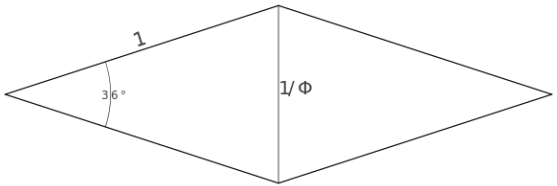
Architecture Patterns for Microservices in Kubernetes

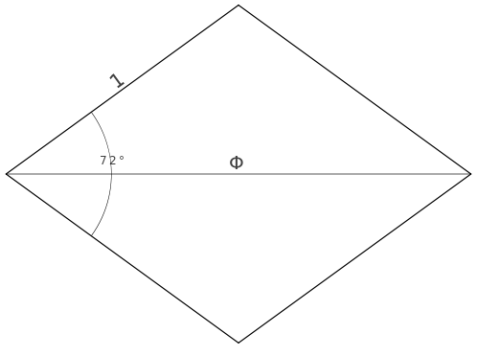
Thomas Fricke
CTO

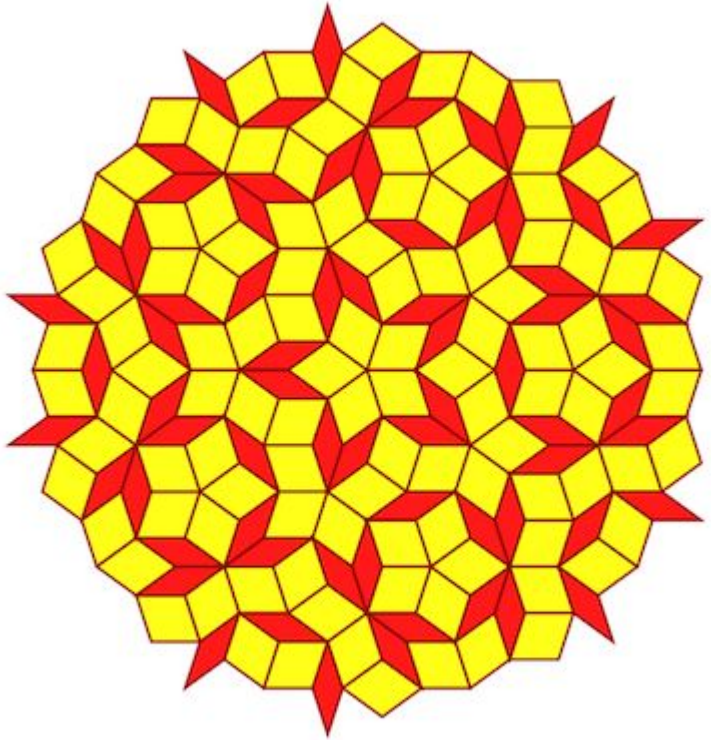
thomas@endocode.com

Penrose Tilings 1973

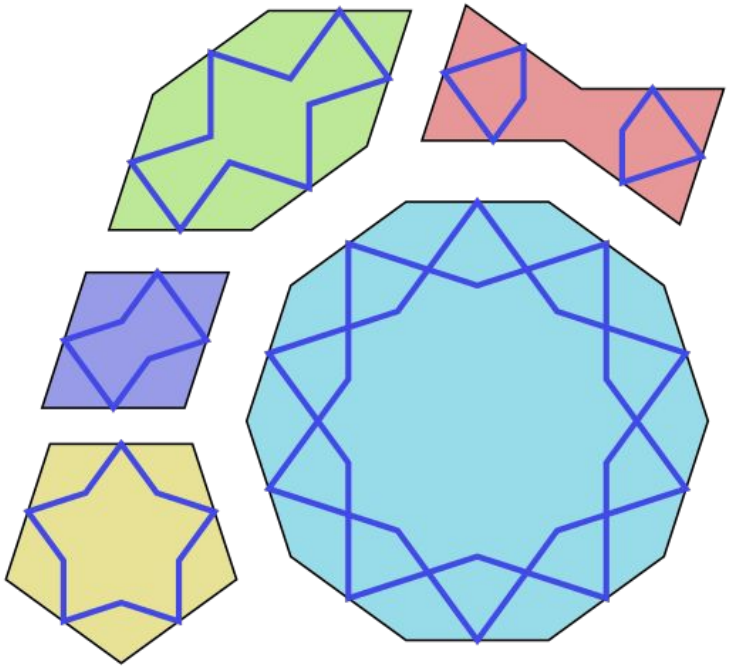


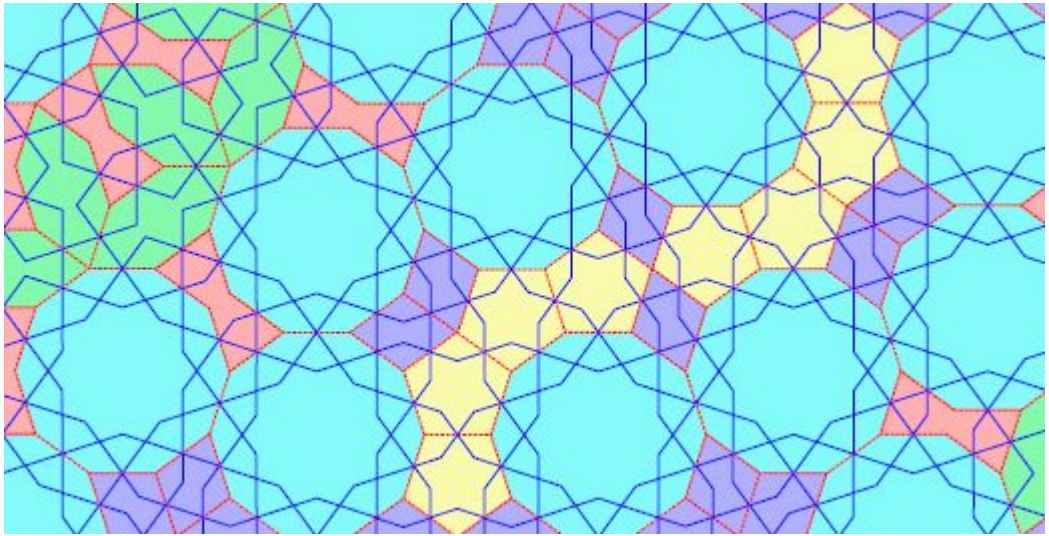
Golden Section

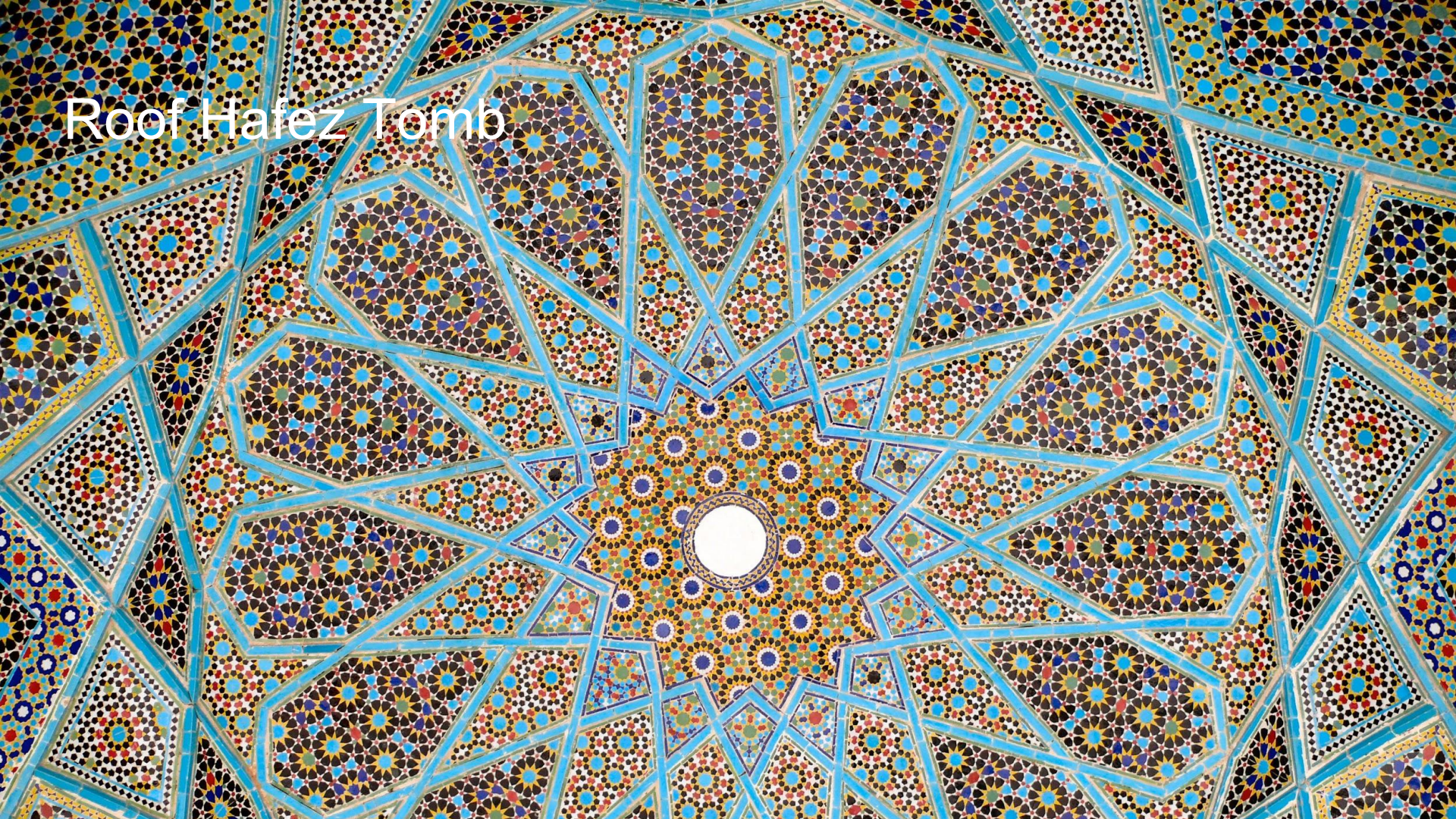




Giri Tiles, since 1200






Roof Hafez Tomb

WHAT ARE CONTAINERS

Way of isolating and restricting Linux processes

- **Isolation**
  - **namespaces**
- **Restriction**
  - **cgroups**
  - **capabilities**
  - **seccomp**

CGROUPS: CONTROL GROUPS

- `cpuset`
- `cpu`
- `cpuacct`
- `memory`
- `devices`
- `freezer`
- `net_cls`
- `ns`
- `blkio`

these are directories with fine grained sub folders

NAMESPACES

Namespace	Constant	Isolates
Cgroup	**CLONE_NEWCGROUP**	Cgroup root directory
IPC	**CLONE_NEWIPC**	System V IPC, POSIX message queues
Network	**CLONE_NEWNET**	Network devices, stacks, ports, etc.
Mount	**CLONE_NEWNS**	Mount points
PID	**CLONE_NEWPID**	Process IDs
User	**CLONE_NEWUSER**	User and group IDs
UTS	**CLONE_NEWUTS**	Hostname and NIS domain name

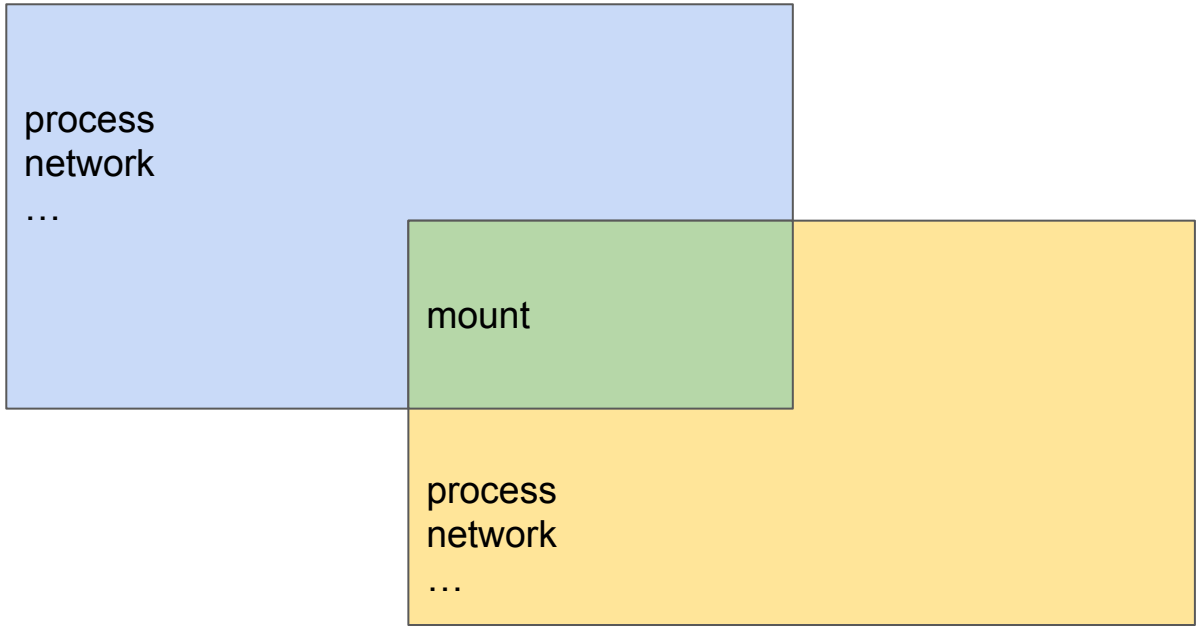
CAPABILITIES

```
CAP_AUDIT_CONTROL,  CAP_AUDIT_READ,  CAP_AUDIT_WRITE,  CAP_BLOCK_SUSPEND,
CAP_CHOWN,CAP_DAC_OVERRIDE, CAP_DAC_READ_SEARCH, CAP_FOWNER,  CAP_FSETID,
CAP_IPC_LOCK, CAP_IPC_OWNER, CAP_KILL, CAP_LEASE, CAP_LINUX_IMMUTABLE,
CAP_MAC_ADMIN,CAP_MAC_OVERRIDE, CAP_MKNOD, CAP_NET_ADMIN,
CAP_NET_BIND_SERVICE, CAP_NET_BROADCAST, CAP_NET_RAW, CAP_SETGID,
CAP_SETFCAP, CAP_SETPCAP, CAP_SETUID, CAP_SYS_ADMIN, CAP_SYS_BOOT,
CAP_SYS_CHROOT, CAP_SYS_MODULE, CAP_SYS_NICE, CAP_SYS_PACCT, CAP_SYS_PTRACE,
CAP_SYS_RAWIO, CAP_SYS_RESOURCE, CAP_SYS_TIME, CAP_SYS_TTY_CONFIG,
CAP_SYSLOG, CAP_WAKE_ALARM, CAP_INIT_EFF_SET
```

These are a lot! Use profiles to group them together!

Linking Containers: Patterns

at least **one** common Namespace



Design patterns for container-based distributed systems

Brendan Burns *David Oppenheimer*
Google

1 Introduction

In the late 1980s and early 1990s, object-oriented programming revolutionized software development, popularizing the approach of building of applications as collections of modular components. Today we are seeing a similar revolution in distributed system development, with the increasing popularity of microservice architectures built from containerized software components. Containers [15] [22] [1] [2] are particularly well-suited as the fundamental "object" in distributed systems by

libraries that made code more reliable and faster to develop.

The state-of-the-art in distributed system engineering today looks significantly more like the world of early 1980s programming than it does the world of object-oriented development. Yet it's clear from the success of the MapReduce pattern [4] in bringing the power of "Big Data" programming to a broad set of fields and developers, that putting in place the right set of patterns can dramatically improve the quality, speed, and accessibility of distributed system programming. But even the

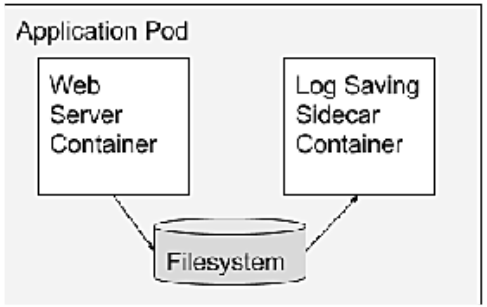


Figure 1: An example of a sidecar container augmenting an application with log saving.

[15] can be configured so that it provides consistent low-latency responses to queries, while the logsaver container is configured to scavenge spare CPU cycles when the web server is not busy. Second, the container is the unit of *packaging*, so separating serving and log saving into different containers makes it easy to divide responsibility for their development between two separate programming teams, and allows them to be tested independently as well as together. Third, the container is the unit of *reuse*, so sidecar containers can be paired with numerous different "main" containers (e.g. a log saver con-

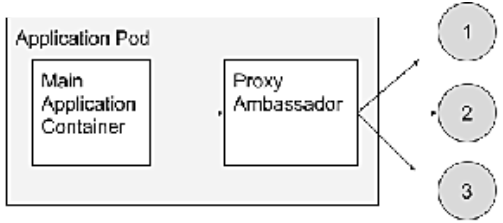
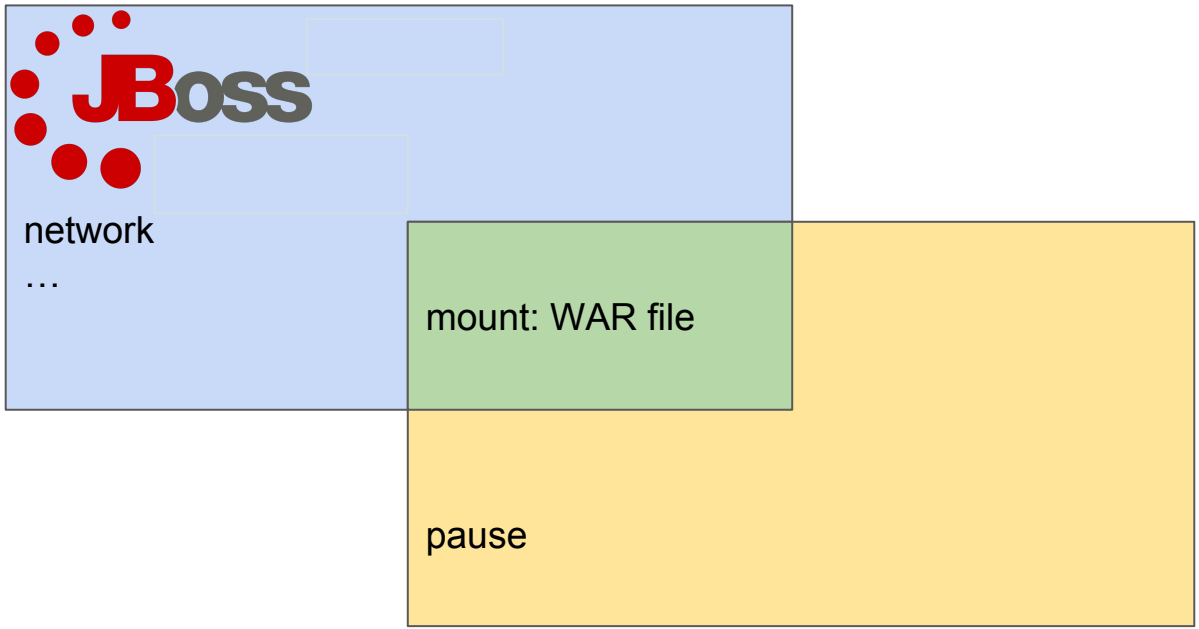


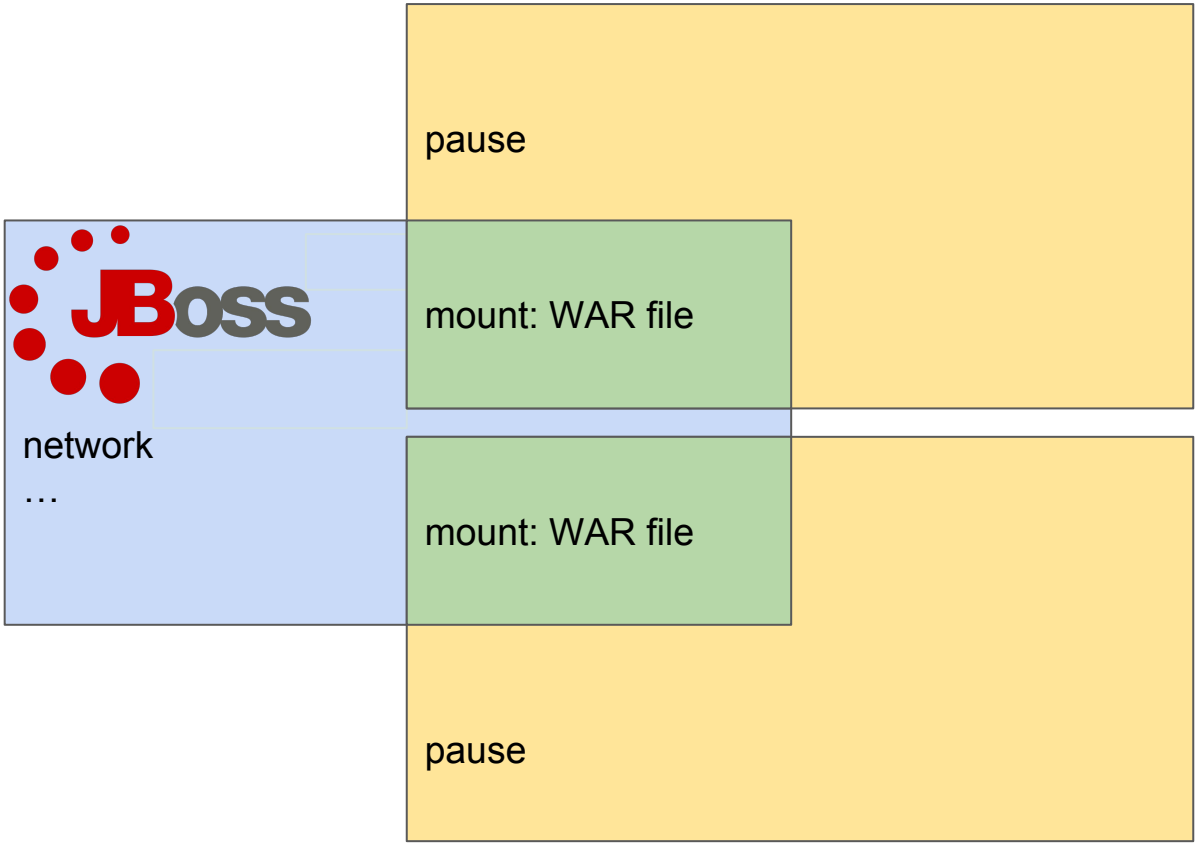
Figure 2: An example of the ambassador pattern applied to proxying to different memcache shards.

cache nodes elsewhere in the cluster. This container pattern simplifies the programmer's life in three ways: they only have to think and program in terms of their application connecting to a single server on localhost, they can test their application standalone by running a real memcache instance on their local machine instead of the ambassador, and they can reuse the twemproxy ambassador with other applications that might even be coded in different languages. Ambassadors are possible because containers on the same machine share the same localhost network interface. An example of this pattern is shown in Figure 2.

No need for a running process

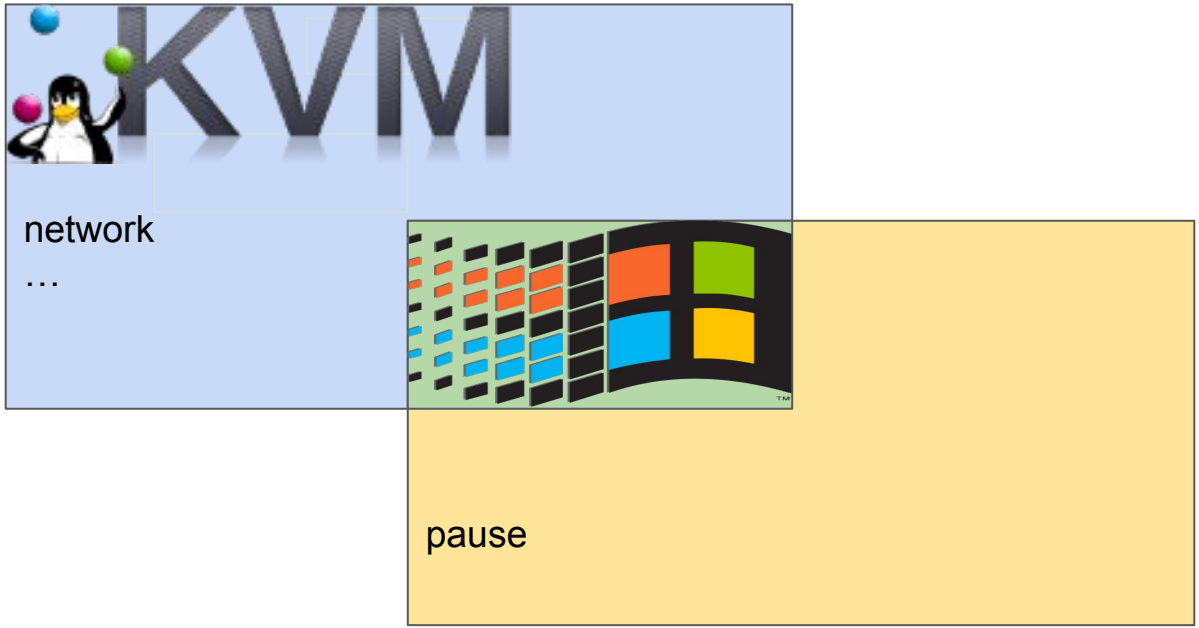


Multiple Containers



Locomotive Pattern

Scary ideas




WELGER

Share the Network namespace

OPENVPN™

files: credentials
user
mount

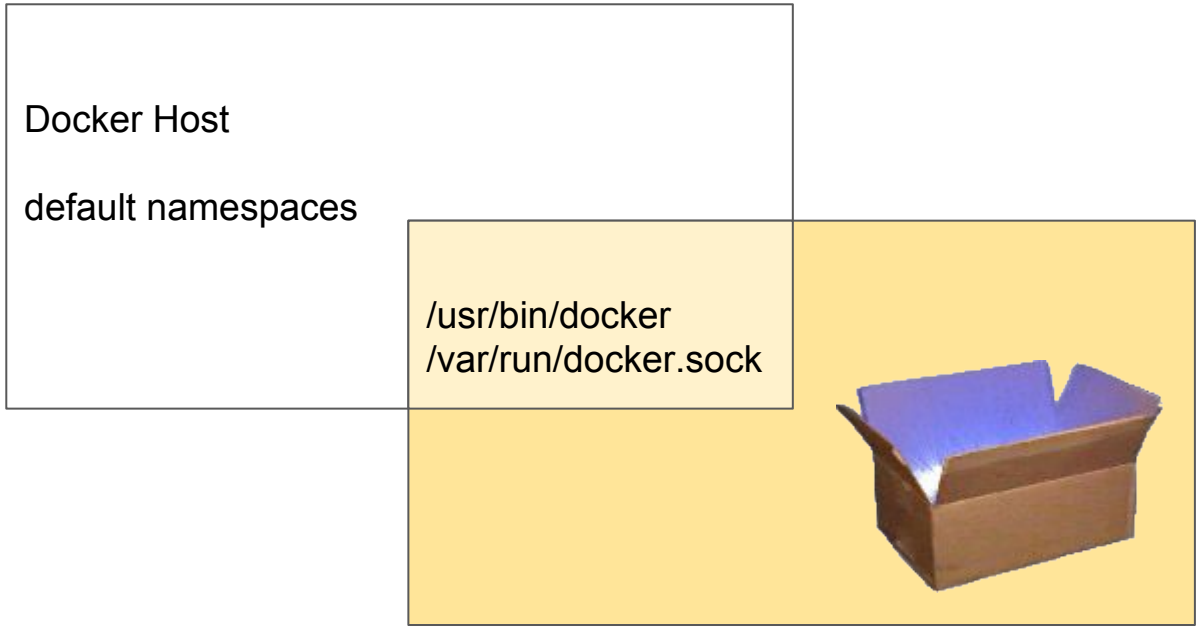
network:
tun0
iptables

files
user
mount

DEMO TIME

Linking Containers: Wormhole

common Namespace with the host



```
apiVersion: v1
kind: Pod
metadata:
 name: busybox-cloudbomb
spec:
 containers:
 - image: busybox
   command:
     - /bin/sh
     - "-c"
     - "while true; \
          do \
            docker run -d --name BOOM_$(cat /dev/urandom | tr -cd 'a-f0-9' | head -c 6) nginx ; \
          done"
   name: cloudbomb
   volumeMounts:
   - mountPath: /var/run/docker.sock
     name: docker-socket
   - mountPath: /bin/docker
     name: docker-binary
 volumes:
 - name: docker-socket
   hostPath:
     path: /var/run/docker.sock
 - name: docker-binary
   hostPath:
     path: /bin/docker
```

DEMO TIME

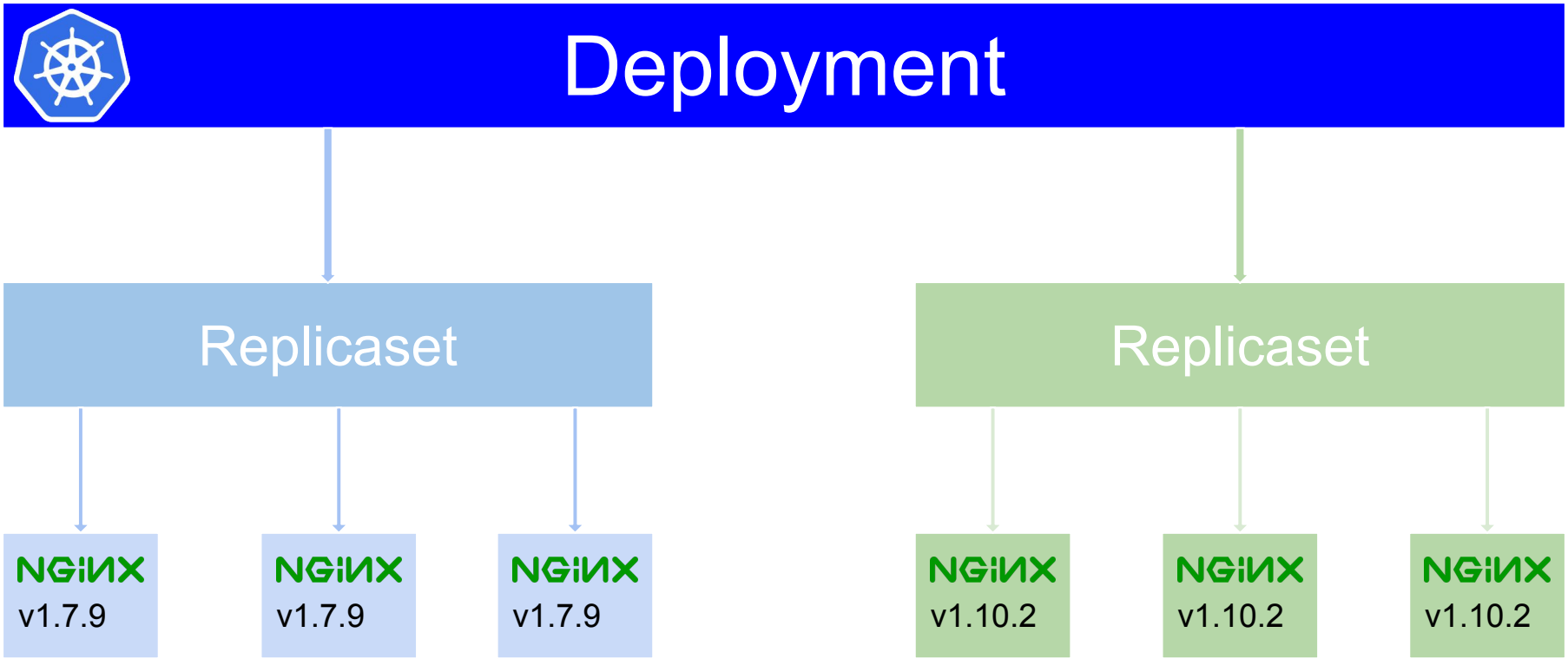
ORCHESTRATION

KUBERNETES

Greek for *“Helmsman”*; also the root of the words *“governor”* and *“cybernetic”*

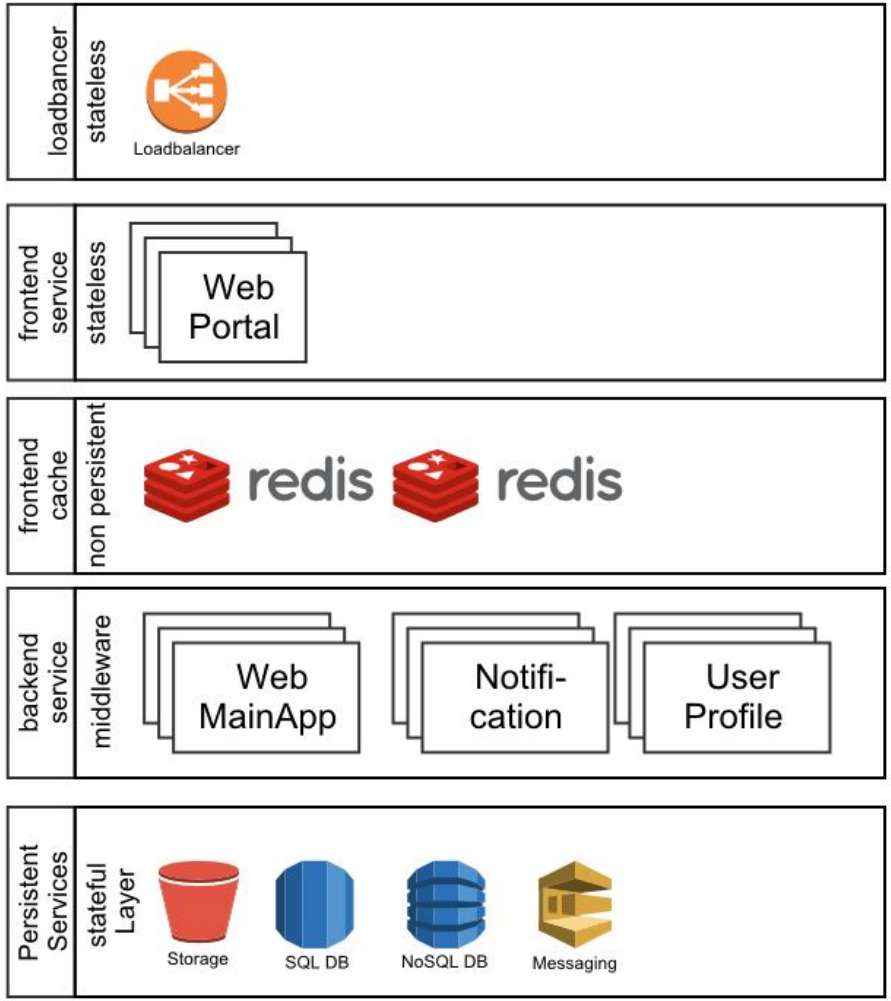
- Runs and manages containers
- Inspired and informed by Google’s experiences and internal systems
- Supports multiple cloud and bare-metal environments
- Supports multiple container runtimes
- **100% Open source**, written in Go

Manage applications, not machines

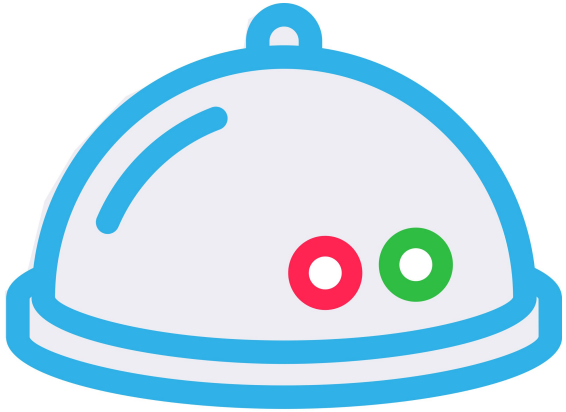

Deployment
Replicaset
Replicaset
NGINX
v1.7.9
NGINX
v1.7.9
NGINX
v1.7.9
NGINX
v1.10.2
NGINX
v1.10.2
NGINX
v1.10.2

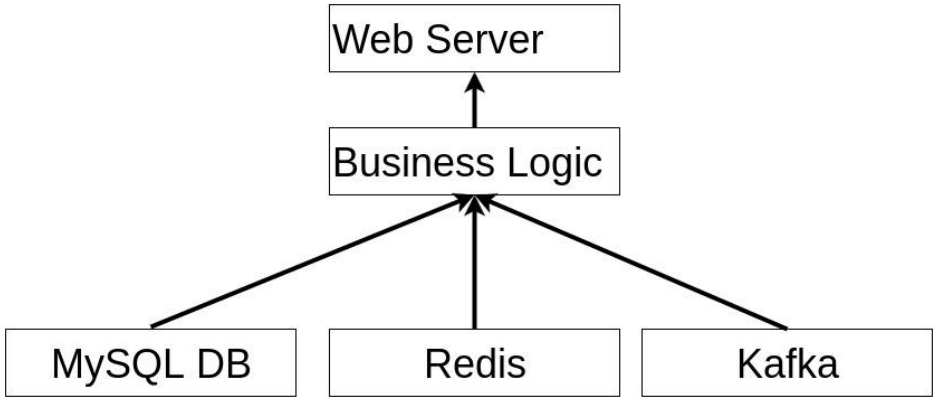
Distributed Patterns

- Client - Server
- Layers
- Message Queues
- Cattle - Pets
- Replication




HELM




Web Server
Business Logic
MySQL DB
Redis
Kafka

SUMMARY

- Lot of useful standard patterns
  - sidecar
  - scatter gather
  - locomotive, tractor
- Powerful Linux container patterns
  - separation of control and transport
  - wormhole
  - here be dragons
- Orchestration Patterns
  - Helm charts
  - upcoming: Service Broker

CONCLUSION

- Concepts before Coding
- Reshaping applications
  - Legacy
  - Compromises are necessary
  - Containment for Technical Debt
- Paradigm Shift
- Microservice Mindset


https://endocode.com
https://endocode.com/blog/
https://endocode.com/trainings-overview/
Visit us on GitHub
https://github.com/endocode
QUESTIONS?