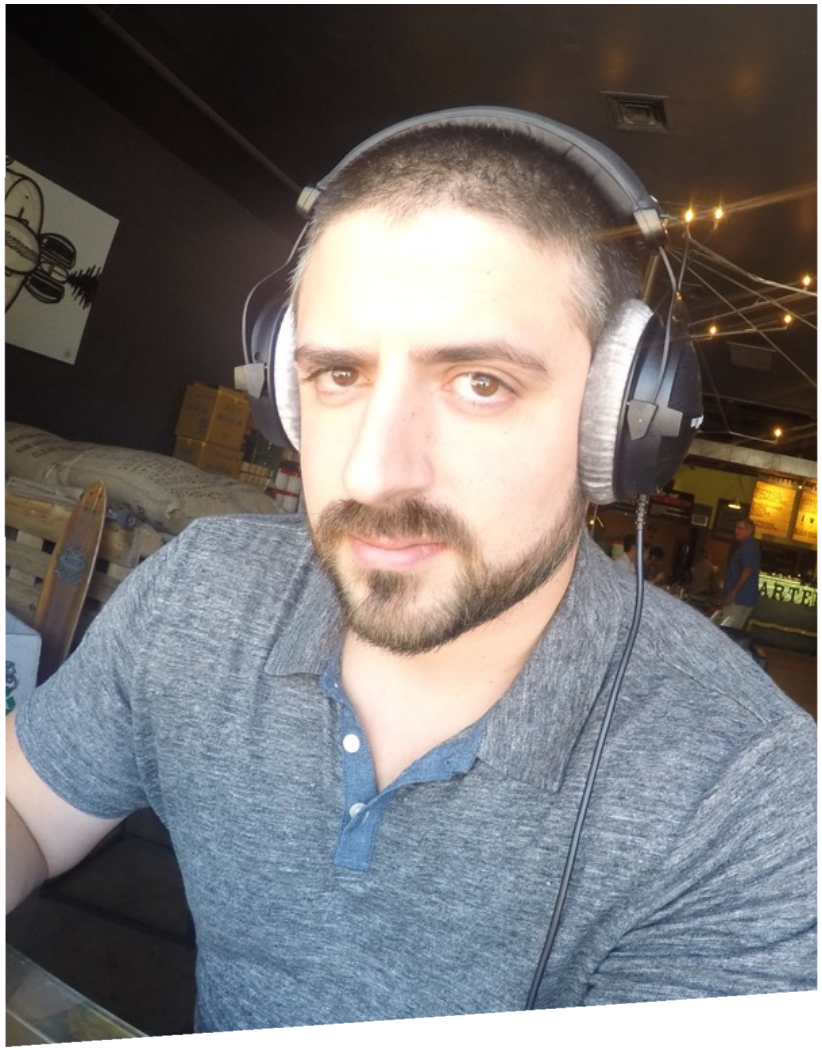


Getting Deep on Orchestration

Jeff Nickoloff

Engineer, Author, and Consultant

dockercon 16

Agenda

What is orchestration?
Abstractions
Examples

Components and patterns
APIs
System of Record
Agents

Demo: Entropy
About failure
Architecture
Break stuff

About Abstractions

High quality abstractions are force multipliers for communication and reasoning about more complex (or potentially variable) ideas and systems.

Using a computer or programming is an act of communication.

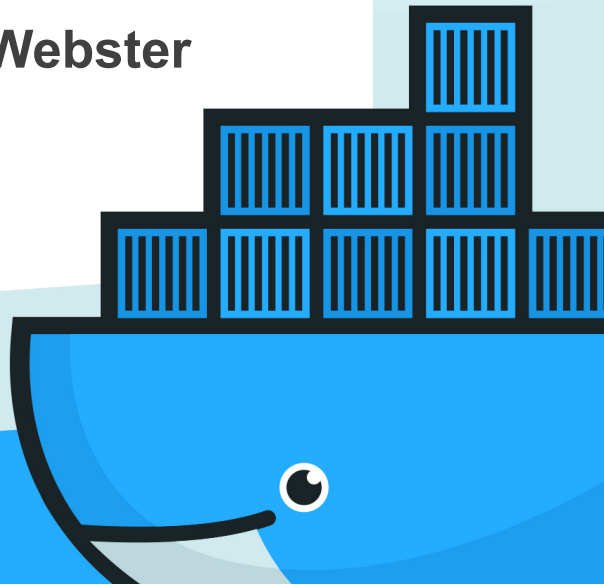
<tba> (Software that provides such an abstraction gives a user amazing power.) Software like Docker.

Orchestration

What is it anyway?

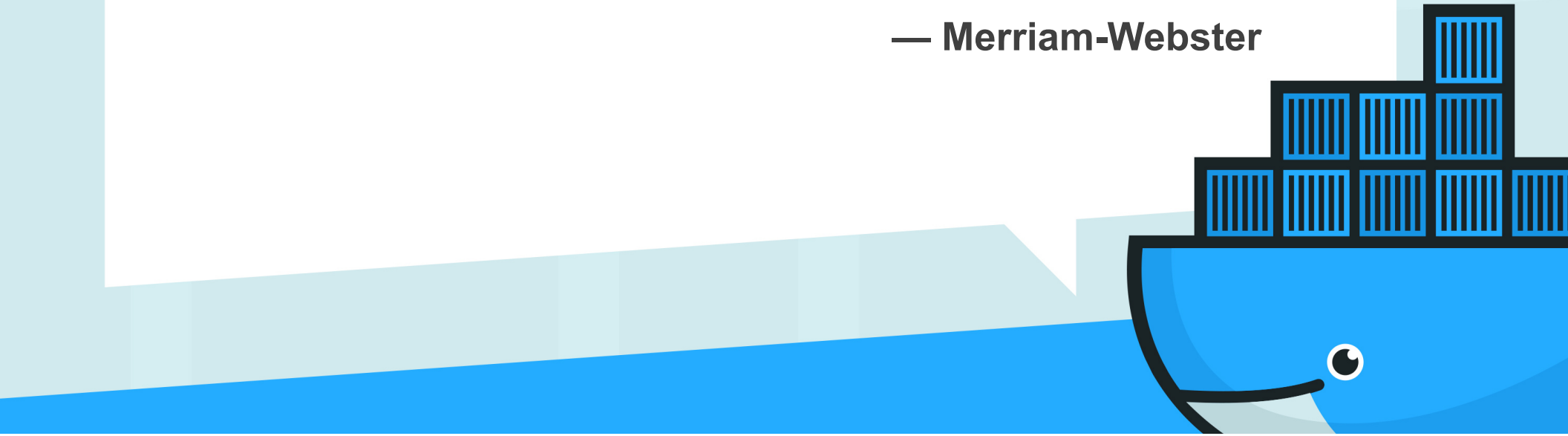
“Orchestrate (v):
to organize or plan
(something that is complicated).”

— Merriam-Webster



“Orchestrating (tv):
to arrange or combine so as to
achieve a desired or maximum effect.”

— Merriam-Webster



Orchestration Platform (Software)

(for orchestral music uses see Podium)

A system that provides organization and planning abstractions for other abstractions.

In the OSS world we have:

- **Swarm**
- **Kubernetes**
- **Mesosphere**

Orchestration Platform (Software)

(for orchestral music uses see Podium)

A system that provides organization and planning abstractions for other abstractions.

In the OSS world we have:

- **Docker + Swarm + Compose + etcd / consul / zk**
- **Docker + Flannel + Systemd + etcd + kubelet + kube (api, scheduler, controller-manager, proxy, DNS)**
- **Mesos + Marathon + Calico + zk**

Abstractions

Force multiplying ideas: a few lower level examples

- Engine (runs containers)
- Cluster (nodes as an engine)
- Network (routing, overlay, etc)
  - and network name resolution
- Volume (named sharable storage)

Abstractions

Force multiplying ideas: higher level examples

- Service (long running event handler)
- Job (process with a linear lifecycle)
- Feed (Job with an input document)
- Report (Job with an output document)
- Request (process a single request)
- Cron Job (run a periodic job)

Architecture

Platform components
and patterns

An Interface

… probably many!

Users and other software compositions need some means to interact with the orchestration platform.

- **One or a collection of APIs**
- **Command Line Program**
- **Web or Native GUI**

System of Record

Managed state of the platform

Accounting for entities (container, pod, service, volume, etc) and their state.

Commonly provide:

- **KV semantics**
- **Record observation (watches)**
- **Update / Delete semantics with fencing tokens**
- **Distributed locks**
- **HA with strong consistency (Paxos / Raft)**

Agents / Control Loops

API's provide interfaces between the platform and users or other tools.

Control loops are the platform's automata.

- **Passively react to state changes**
- **Manage system state based on active monitoring**
- **Might require leadership election for HA**

Agents / Control Loops

Event driven agents that coordinate changes in the system

- Container (re)scheduler
- Cluster node registrar
- Service node registrar
- Low entropy network registrar
- Local supervisor / init
- Service controller
- Job controller
- Report workflow controller
- Feed workflow controller
- Distributed cron controller

Patterns

Event driven agents that coordinate changes in the system

- State Observation
  - Feedback for control loops
- Entity Lifecycle Graph
- Registration and Discovery
  - Route to an IP
  - Engine in a cluster
  - Replica of a service
  - Endpoint of a service

Deep Dive: Cluster

Getting dirty

<insert illustration of a Swarm cluster>
<insert illustration of a Kubernetes cluster>
<insert illustration of a Mesosphere cluster>

Deep Dive: Service

Even more examples

<tba>

An Example

Demo a new abstraction:
Entropy

Failure Injection

A powerful and complex idea

Build confidence in complex distributed systems by injecting realistic failures and comparing operations against a steady state.

- **http://principlesofchaos.org**
- **Failure mode and effects analysis**
- **Netflix / SimianArmy**

Failure Injection

… and container based platforms

All container platforms (clustered or not):

- **enable high-entropy systems**
- **add new components and failure modes**
- **provide new mechanisms for handling failure**

__ extension point, failure injection interface (the kernel), cross cutting concern

Project: Entropy

An orchestration abstraction for failure injection

Features:

- **Probabilistic failure injection policies**
- **Failure modes**
  - **Latency, partition, GC pause, etc**
- **Applied to existing containers filtered by label**
- **An event stream**
- **Notifications**
- **Integrates with the Docker API**

Project: Entropy

An orchestration abstraction for failure injection

Components:

- **Microservice API**
- **CLI**
- **Policy manager (control loop)**
- **Failure injection agents (pluggable)**

On GitHub: __ insert URL

Demo Slide

```
$ # Container platform is running. Start a service.
$ docker-compose up -d -p election -f election.yml
$ # Start the entropy platform
$ docker-compose up -d -p entropy -f entropy.yml
$ # Define a policy
$ entropy -H $DOCKER_HOST run \
  -f 1s -p .1 \
  -t servicename=voter \
  recv_drop
```

Thank you!

Checkout:

github.com/buildertools/entropy