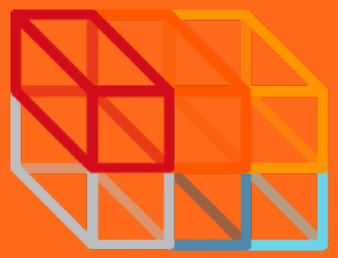
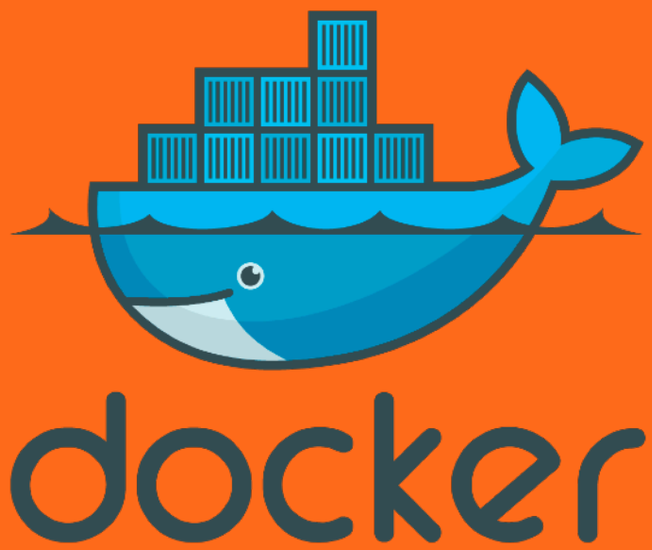
Containerized Node.js Microservices

Wyatt Preul // jsgeek.com/chicago

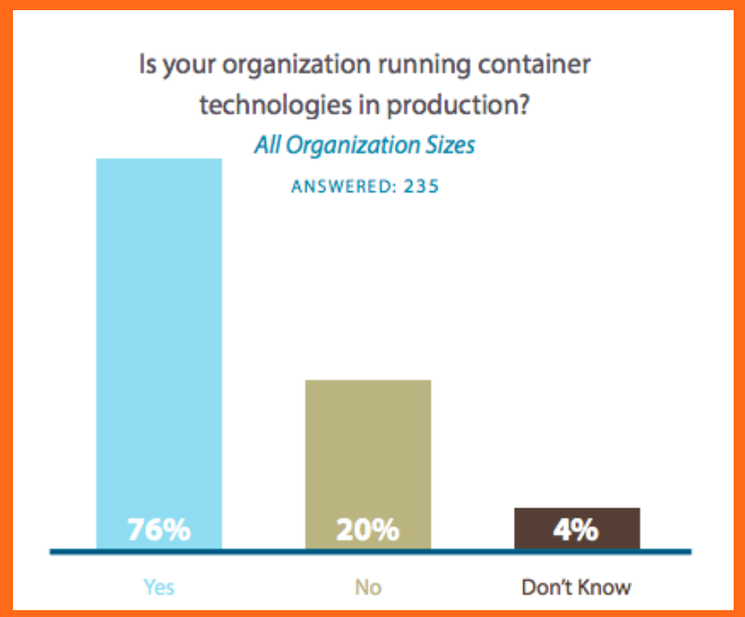


Joyent
TRITON™



Using containers?

… in production?



- 79% are using containers

- 76% are using containers in production

- take-away: more orgs are using containers than previous years

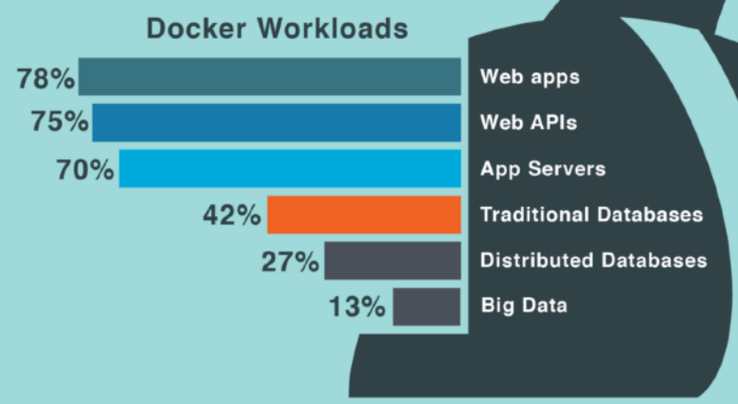
DevOps.com/ClusterHQ: 2016 Container Survey

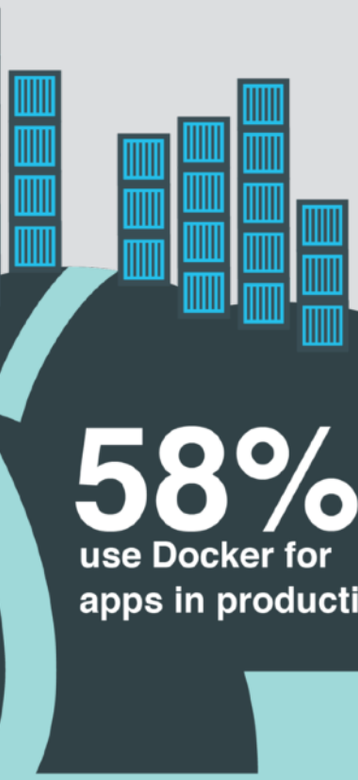
65%
use Docker to deliver development agility.

48%
use Docker to control app environments.

41%
use Docker to achieve app portability.

90%
use Docker for apps in development.





58%
use Docker for apps in production.

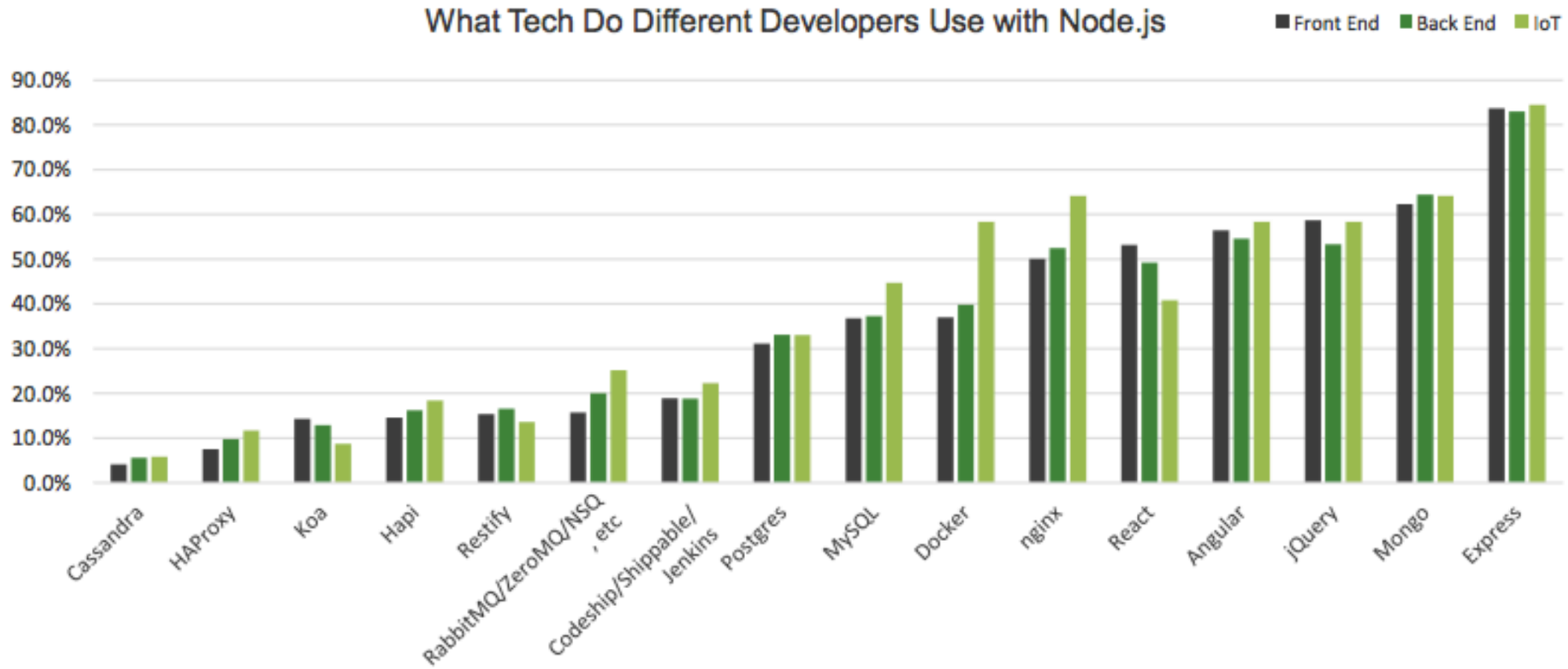
90%
plan dev environments around Docker.

80%
plan DevOps around Docker.

docker

Docker survey results: docker.com/survey-2016

Tech Use with/in Node – Front End, Back End, IoT



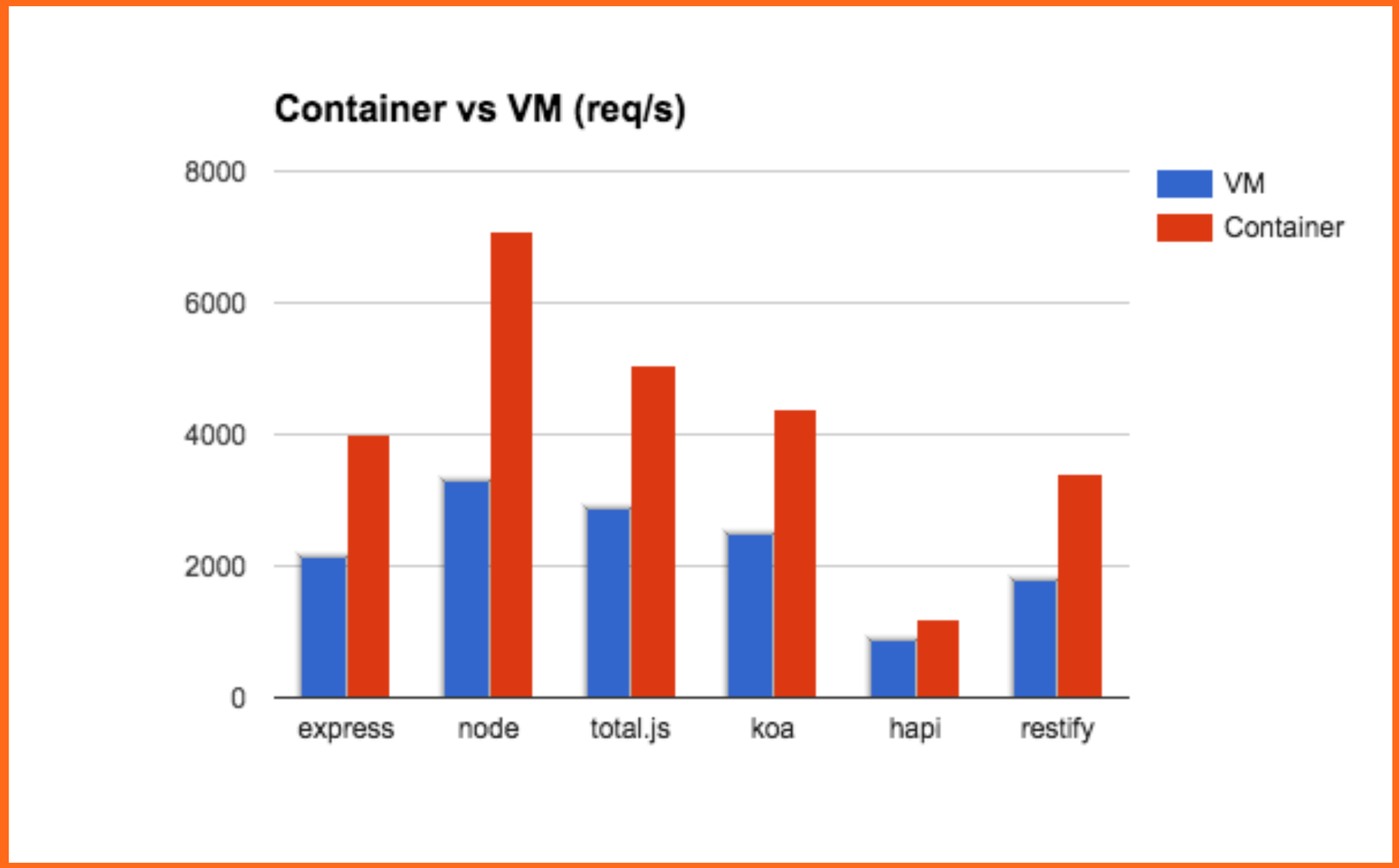
~45% of Developer Respondents use Node.js with Containers

2016 Node.js Survey Report

Benefits of Containers

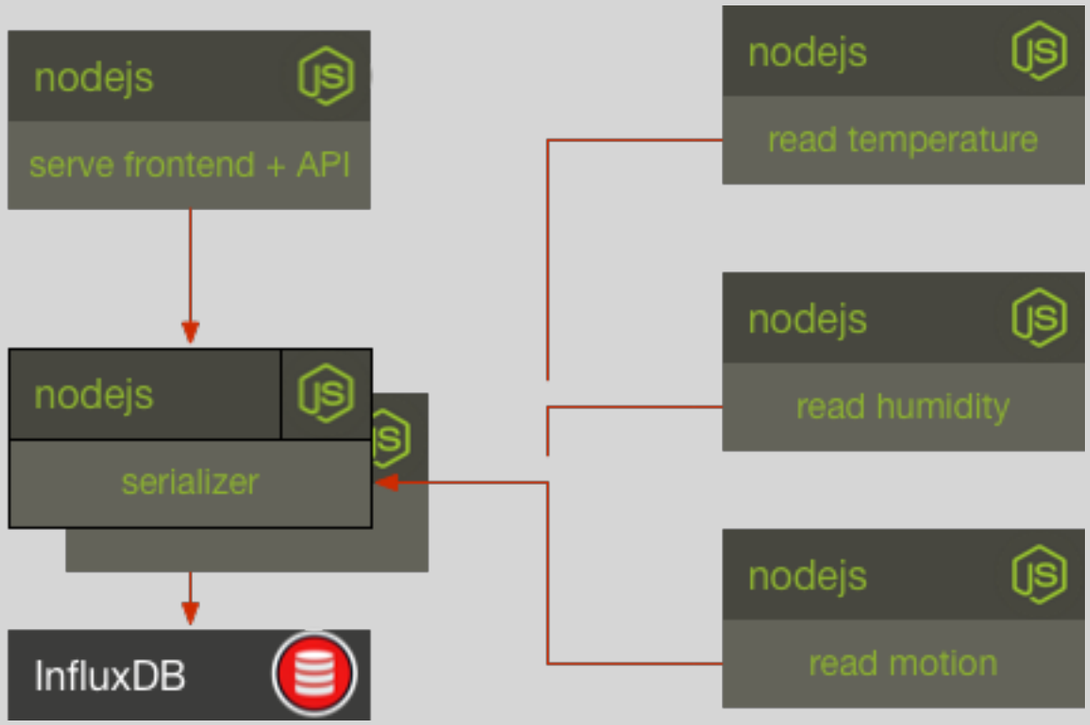
- I heard it was cool
- consistent environments, immutable
- increased developer efficiency
- OS level virtualization, more performant than VM

Hardware vs OS Level Virtualization Performance



CentOS 7, same datacenter, 2gb RAM, Node.js 6.7.0

https://github.com/geek/NodeRestPerfTest



Using microservices?

… in production?

Microservice? NIST 800-180 Draft

- Small, decomposed, isolated and independently deployable services
- Stateless and less fragile when changes are introduced
- Built around [business] capabilities

Microservices Popularity

- 68% of orgs are using or investigating microservices - NGINX 2016 Survey
- ClusterHQ survey indicates orgs are choosing containers to support microservices architecture
- Node.js survey findings indicate that Node.js + containers = perfect combo for microservices architecture

Benefits of Microservices

- align well with Unix Philosophy
- embrace failure, works in spite of external failures
- iterate quickly - disposable services, independently deployable services

Microservices & Containers

- well suited for each other
- disposable, fast, developer friendly
- docker-compose.yml is great for describing a set of microservices

Node.js Microservices & Containers

- tiny, fast, portable
- easily replaceable
- perfect partnership, async i/o services running on the metal in portable containers!

Pitfalls

For Docker,
Microservices

Docker pitfall - PID 1

- bring your own init (BYOI)
- container inits exist: tini, dumb-init, my_init

Docker pitfall - lifecycle

- need setup and teardown hooks in container
- perform initialization before starting
- perform cleanup (finish writes) before container is killed

Docker pitfall - depends_on/links

- depends_on starts services in order, but doesn't account for startup time or time till healthy
- not reliable as mechanism for guaranteeing a service is "ready" before another one
- build resiliency into services (interruptions do occur)

Microservice pitfall - deploy order

- deployments are more complicated if there is an ordering when services need to come online
- services aren't designed to embrace failure
- this is a sign you aren't building microservices (independently deployable)

Microservice pitfall - load balancer

- with lots of microservices and hosts, misconfiguration is likely more common as more configuration is required

- increased latency between services

- slightly more work is required to replicate production env in development

Autopilot Pattern

- Apps that can be deployed and scaled with a single click.
- Apps and workflows that work the same on our laptops as in the cloud (public and private cloud).
- Apps and workflows that aren't married to any specific infrastructure or scheduler.

TRITON ContainerPilot

Addresses previous issues + FOSS

Applications on Autopilot

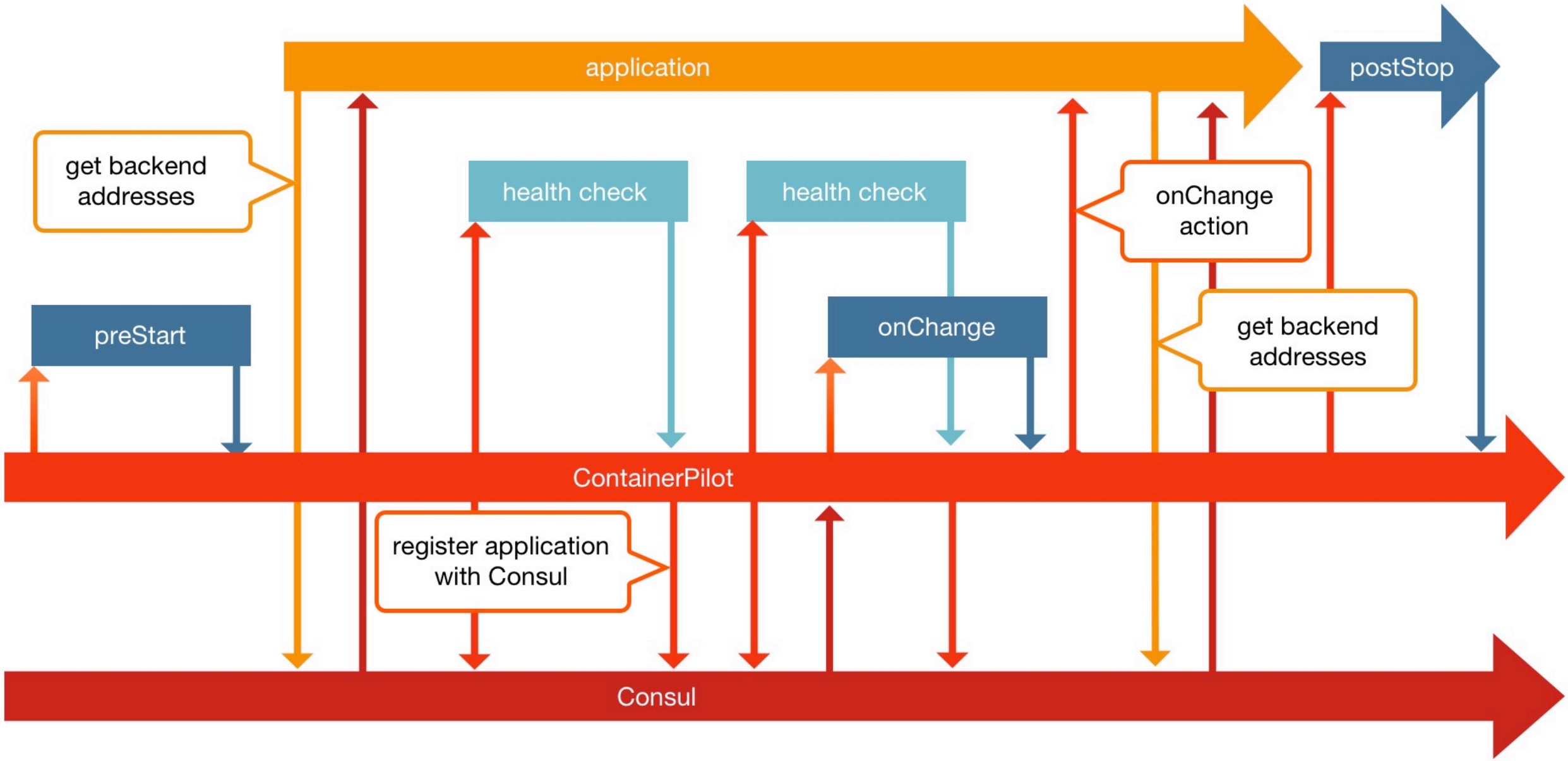
- autopilotpattern.io - describes pattern
- github.com/autopilotpattern - location of solutions using the Autopilot Pattern with ContainerPilot
- MongoDB, MySQL, InfluxDB, Consul, Wordpress, Jenkins, …

ContainerPilot

- tool to automate a container's service discovery, life cycle management, and configuration portable, works anywhere docker does
- capabilities:
    - health checks
    - handles startup and shutdown of services
    - runs as pid 1 in the container
    - register service with and watches consul for dependency changes
    - telemetry reporting
    - automatically reconfigures service upon state change
- open-source, free: github.com/joyent/containerpilot

ContainerPilot 3

- all planning is public in RFD process, see RFD 86
- ability to start service after a dep is healthy
- can have multiple health checks per service
- multi-process containers more straightforward
- + more


application
postStop
get backend addresses
health check
health check
onChange action
preStart
onChange
get backend addresses
ContainerPilot
register application with Consul
Consul

nodejs-example

github.com/autopilotpattern/nodejs-example

smartthings

Browser

Docker Container
ContainerPilot
smartthings

Docker Container
ContainerPilot
natsboard

Docker Container
ContainerPilot
Prometheus

Docker Container
ContainerPilot
traefik

Docker Container
Docker Container
Docker Container
ContainerPilot
NATS

Docker Container
Docker Container
Docker Container
ContainerPilot
Consul

Docker Container
Docker Container
ContainerPilot
nodejs
frontend

Docker Container
ContainerPilot
humidity

Docker Container
ContainerPilot
motion

Docker Container
ContainerPilot
temperature

Docker Container
Docker Container
ContainerPilot
nodejs
serializer

Docker Container
ContainerPilot
InfluxDB

Joyent
TRITON™

Node.js modules

- hapi - web API framework
- Seneca - microservices framework
- Piloted - ContainerPilot integration, relies on consul
- Wreck - simple module for making performant HTTP requests

Code & Demo

```
$ git clone https://github.com/autopilotpattern/nodejs-example.git


$ cd nodejs-example


$ EDITOR .
```

Recap

- Use ContainerPilot with Node.js docker containers (piloted module)
- Use consul for discovery (autopilotpattern/consul)
- Make microservices independently deployable and fault tolerant

Deploying to prod

Triton Provides

- Containers as a Service
    - Docker - The data center is the docker host
- Software for Public and Private deployment
- Open Source!

Joyent

TRITON™

External facing services

- Public API (cloudapi)
- Operator Portal (adminui)

Internal services

- Monitoring (amon)
- DNS (binder)
- Cloud Analytics (ca)
- Compute Node API (cnapi)
- DHCP (dhcpd)
- Firewall API (fwapi)
- Image API (imgapi)
- Network API (napi)
- Packages API (papi)
- Services API (sapi)
- VMs API (vmapi)
- Workflow API (workflow)
- Docker API (docker)

Infrastructure services

- SDC ops/tools (sdc)
- Zookeeper (zookeeper)
- AMQP (rabbitmq)
- Assets (assets)
- Redis (redis, amonredis)

Data tier services

- User auth cache (mahi)
- LDAP Dir Service (ufds)
- Key/Value Store (moray)
- HA Postgres (manatee)

Global zone agents

- amon-agent
- amon-relay
- cainstsvc
- config-agent
- firewaller
- hagfish-watcher
- cn-agent
- net-agent
- cmon-agent
- smartlogin
- ur
- vm-agent

Docker on Triton - Demo

```
$ eval $(triton env)

$ docker-compose up -d

$ open http://$(triton ip nodejsexample_frontend_1)

$ docker logs -f nodejsexample_frontend_1
```

Production vs. Development

- Development against local Docker
  - One host
  - Great for rapid development
- Production against Triton
  - Still one “host”

    The datacenter is viewed as one docker host
  - Standard Docker toolset

    Docker

    Compose
  - Production infrastructure handled for you

    Networking

    Affinity

    Security

Debugging Docker - Demo

```
$ docker exec -it nodejsexample_frontend_1 sh


$ top


# Add tools to path
$ export PATH=$PATH:/native/usr/proc/bin:/native/usr/sbin/:/native/usr/bin/


$ pfiles $(pgrep node)
```

```
# list probes available
$ dtrace -l -p $(pgrep node)


# example, display open files by process
$ dtrace -n 'syscall::open*:entry { printf("%s %s",execname,copyinstr(arg0)); }’

# process names + probe and count
$ dtrace -n 'syscall:::entry { @[execname, probefunc] = count() }'
```

Questions?

Links @ jsgeek.com/chicago