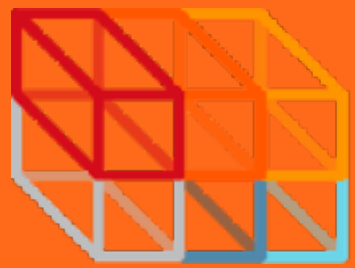




# **Node.js Microservices on Autopilot**

Wyatt Preul // [jsgeek.com/nr](http://jsgeek.com/nr)



Joyent

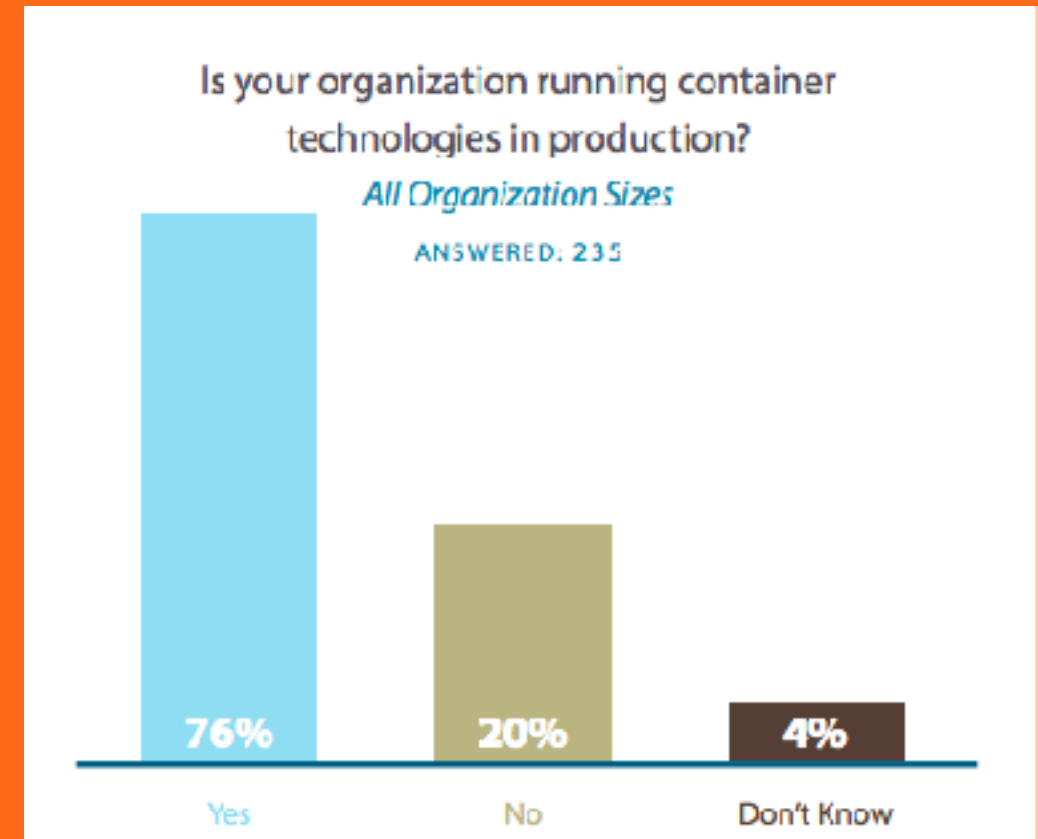
**TRITON**<sup>TM</sup>



docker

**Using containers?**

... in production?



DevOps.com/ClusterHQ: 2016 Container Survey

**65%**

use Docker to deliver development agility.

**48%**

use Docker to control app environments.

**41%**

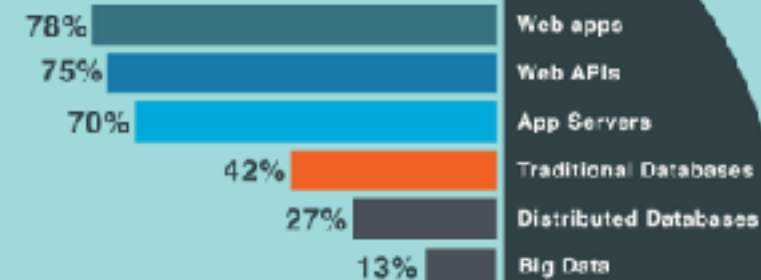
use Docker to achieve app portability.

**90%**

use Docker for apps in development.



Docker Workloads



**58%**

use Docker for apps in production.



**90%**

plan dev environments around Docker.



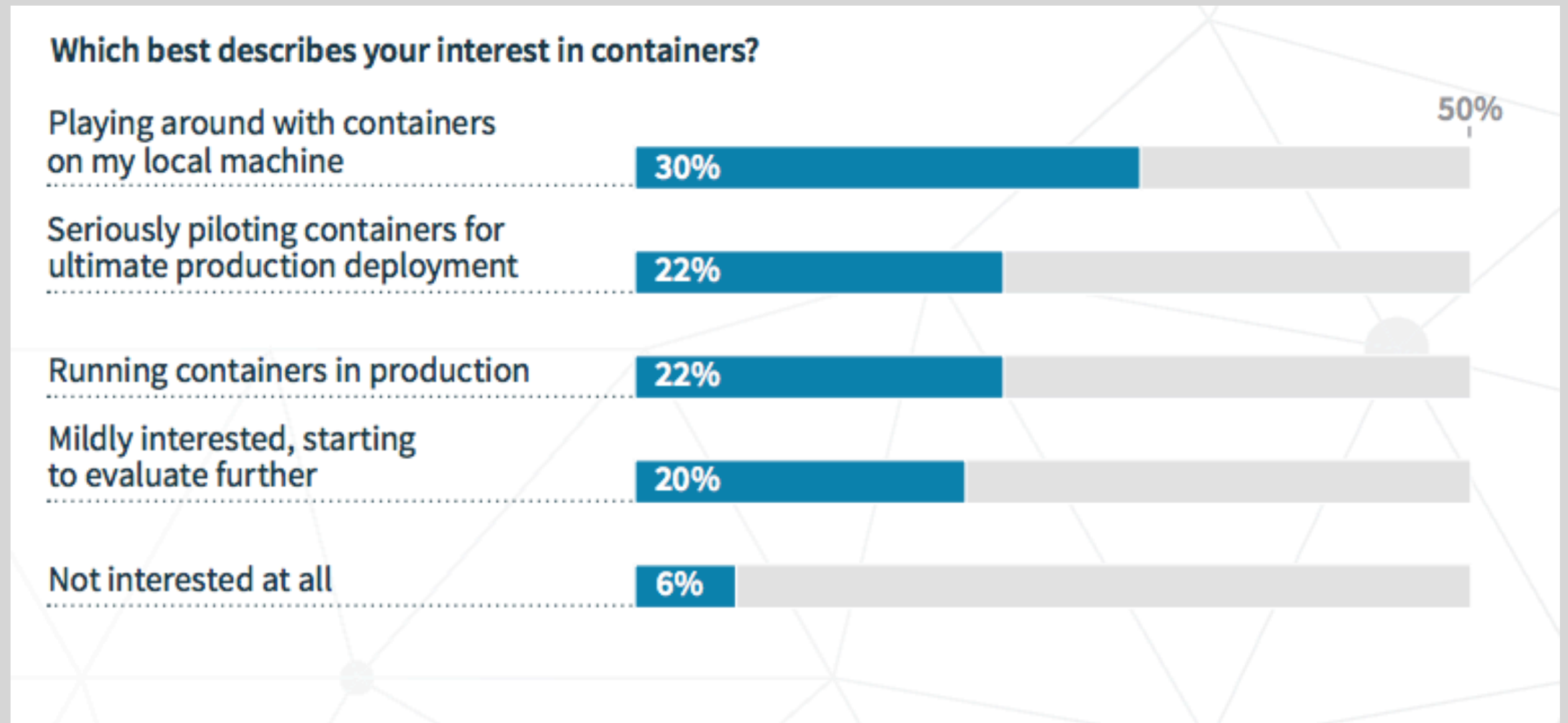
**80%**

plan DevOps around Docker.



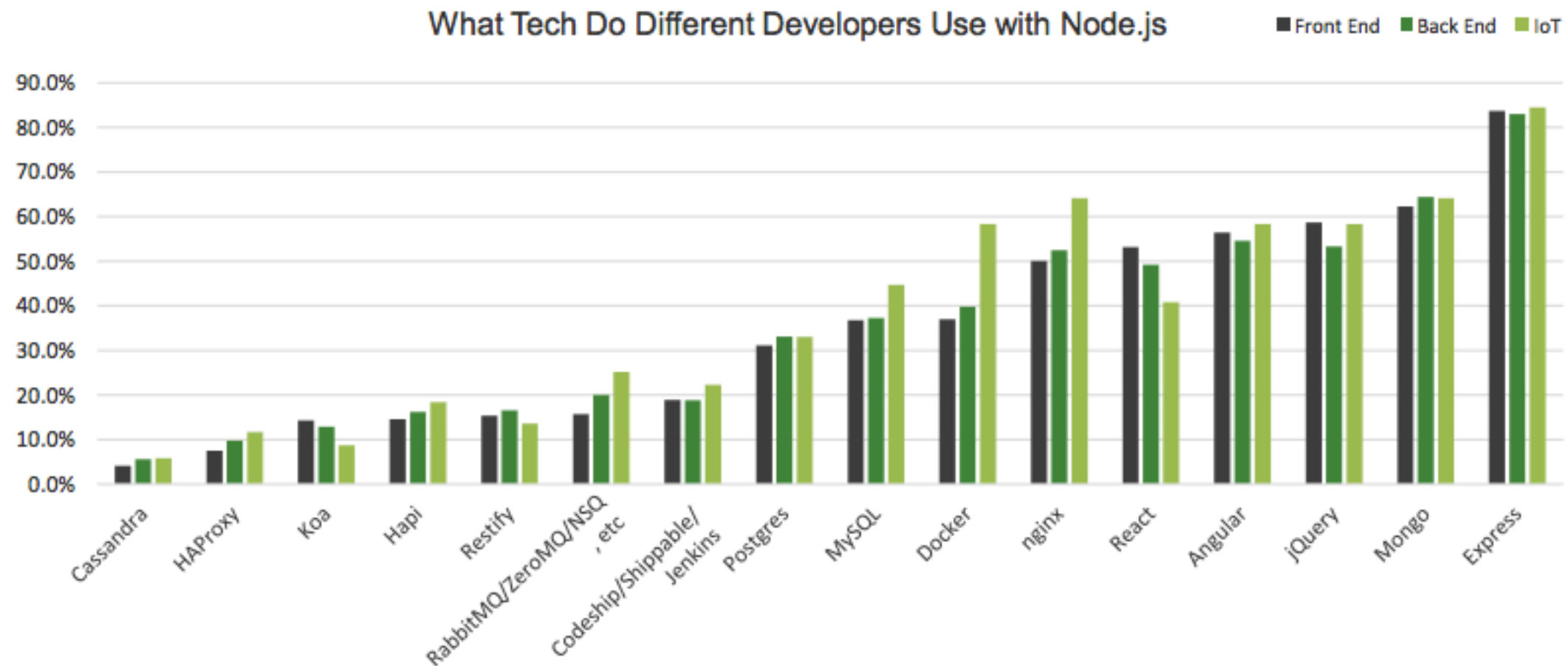
Docker survey results: [docker.com/survey-2016](https://docker.com/survey-2016)

# Java developer interest in containers



Lightbend 2016 Survey of JVM Devs

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



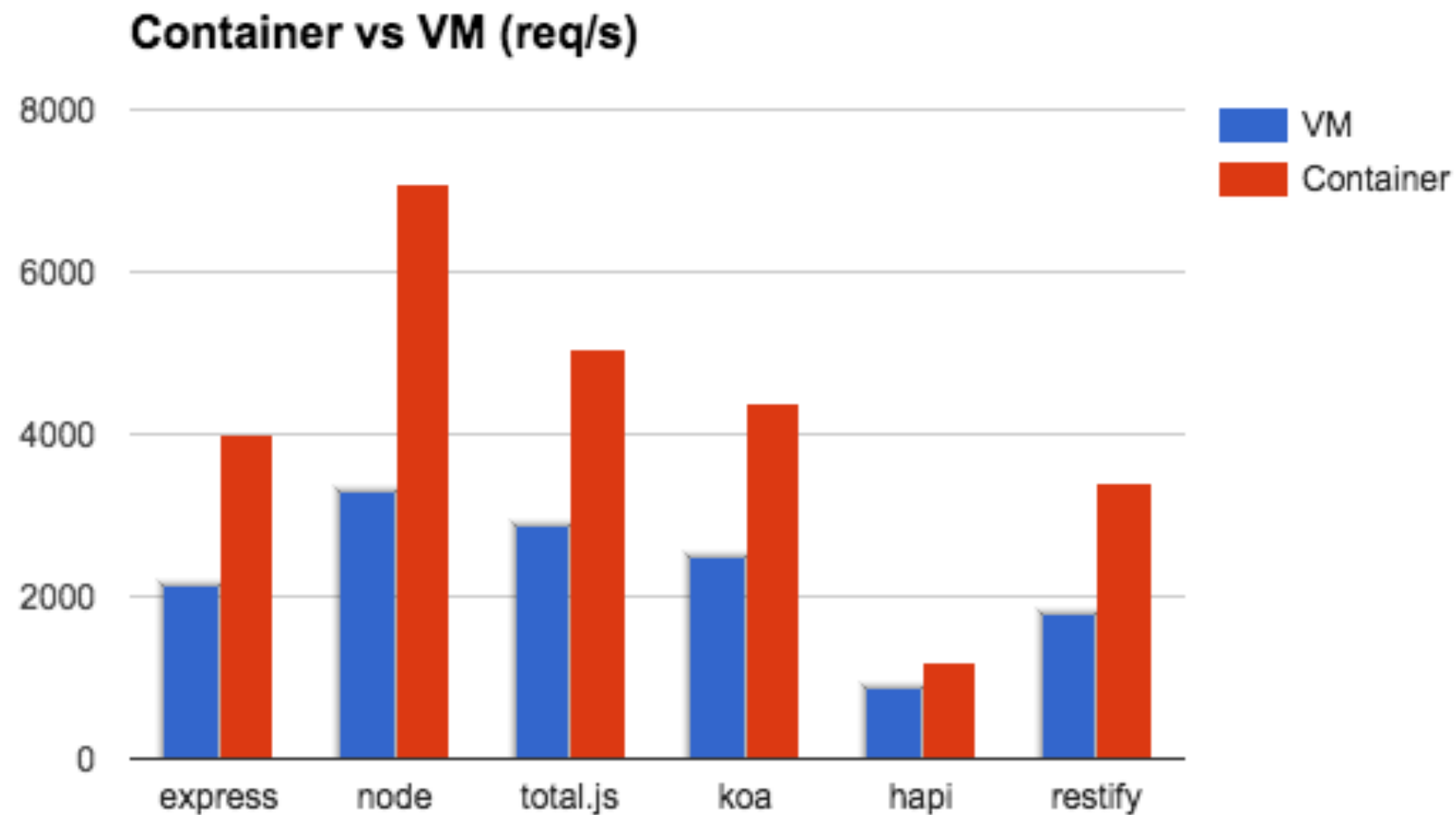
~45% of Developer Respondents use Node.js with Containers

2016 Node.js Survey Report

# Benefits of Containers

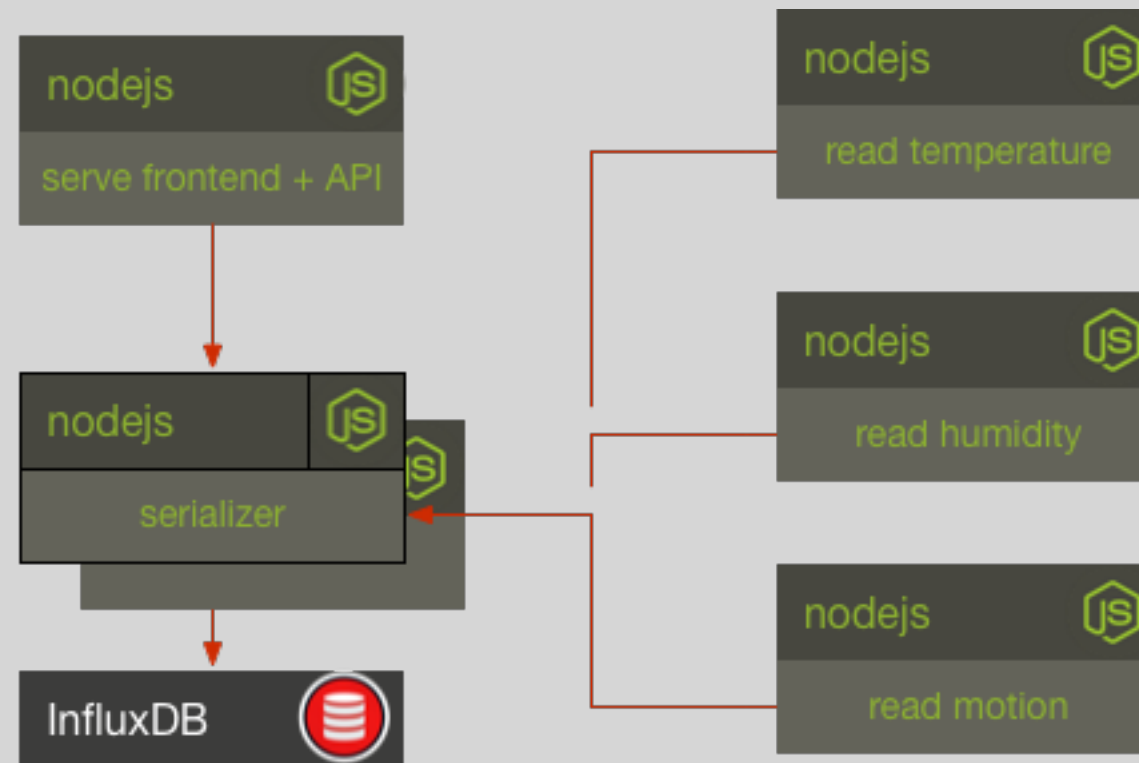
- consistent environments, immutable
- increased developer efficiency
- OS level virtualization, more performant than VM

# Hardware vs OS Level Virtualization Performance



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





# Using microservices?

... in production?

# 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

# Microservices & Containers

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

# Benefits of Node.js

- developer friendly - fun, easy to write
- largest library ecosystem (400k)
- perfect for writing non-blocking i/o code

# Node.js Microservices & Containers

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

# 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 - load balancer

- subdomains setup for environment (qa, stg, prod)... mistakes will happen, not uncommon for a prod service to point to a QA service, oops
- with lots of microservices and hosts, misconfiguration is likely more common
- increased latency between services

# Microservice pitfall - /health

- indicate issue with service, or at least an issue between the load balancer and the service - can be unreliable source of truth
- sometimes perform full checks, db connection, memory usage, exposed as public endpoint (/health) ... can DoS a service



TRITON  
ContainerPilot

Addresses previous issues + FOSS

# 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](https://github.com/joyent/containerpilot)

# Applications on Autopilot

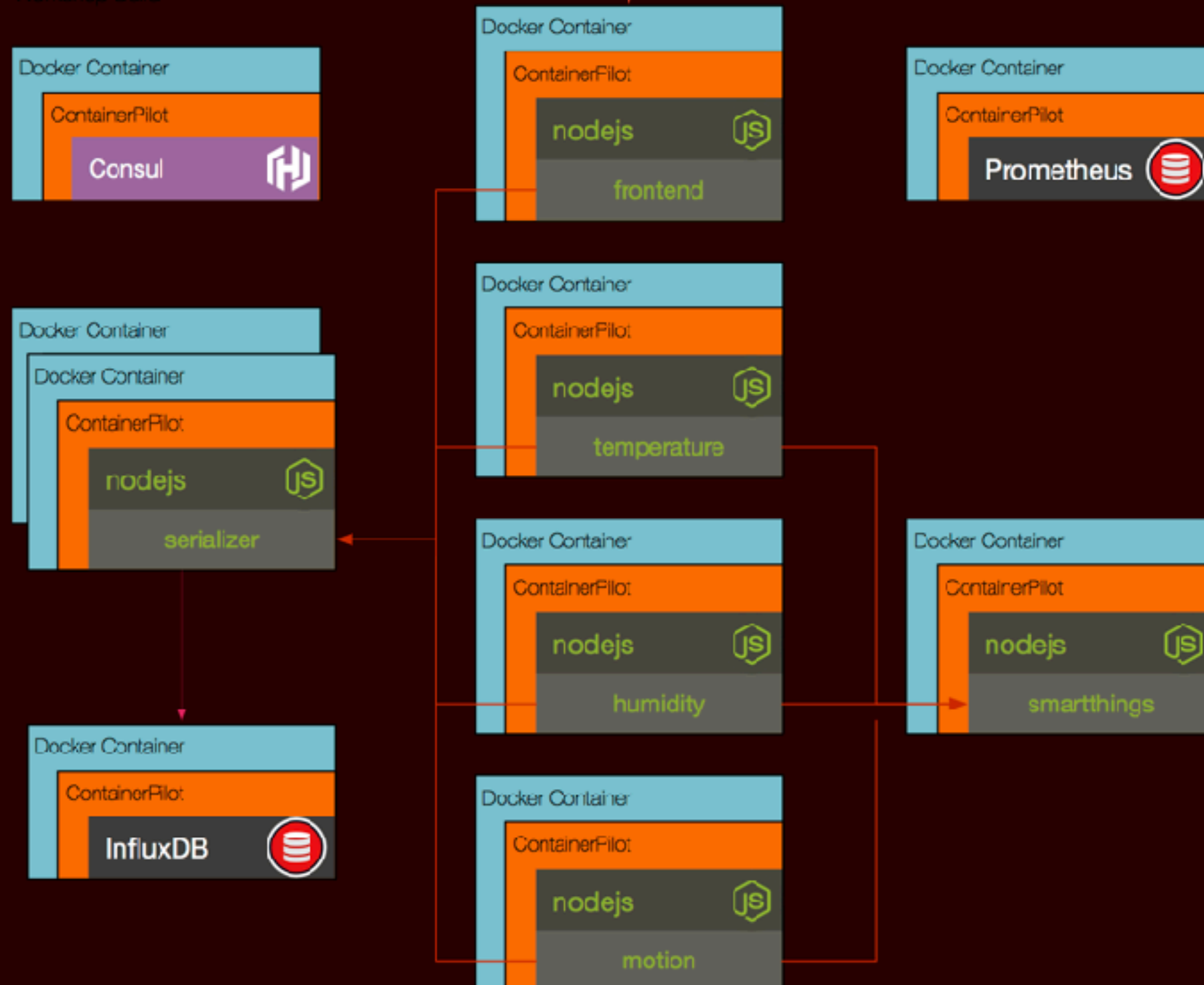
- [autopilotpattern.io](https://autopilotpattern.io) - describes pattern
- [github.com/autopilotpattern](https://github.com/autopilotpattern) - location of solutions using the Autopilot Pattern with ContainerPilot
- MongoDB, MySQL, InfluxDB, Consul, Wordpress, Jenkins, ...



**nodejs-example**

[github.com/autopilotpattern/nodejs-example](https://github.com/autopilotpattern/nodejs-example)

## Workshop Build







# 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 .
```

# 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

# Recap

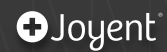
- Use ContainerPilot with Node.js docker containers (piloted module)
- Use consul for discovery (autopilotpattern/consul)
- Make microservices independently deployable and fault tolerant



**Deploying to prod**

# Triton Provides

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



# 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

firewall

hagfish-watcher

cn-agent

net-agent

cmon-agent

smartlogin

ur

vm-agent



# Docker on Triton

---

- Docker Containers = Triton Instances
  - No difference other than how they are managed
    - Docker - via Docker API (docker run etc)
    - Triton Instances - via CloudAPI (triton create)
  - Native networking
    - Each container gets its own IP address(es)
    - No port mapping as such. Firewall rules used to open “mapped” ports
    - Container name service, A Records for groups of services (e.g. consul.srv.us-sw-1.cns.joyent.com)
-

# 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 p tools to path
```

```
$ export PATH=$PATH:/native/usr/proc/bin
```

```
$ pfiles $(pgrep node)
```

```
# Add dtrace to path
```

```
$ export PATH=$PATH:/native/usr/sbin/
```

```
# 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)); }'
```



**Questions?**

Links @ [jsgeek.com/nr](https://jsgeek.com/nr)