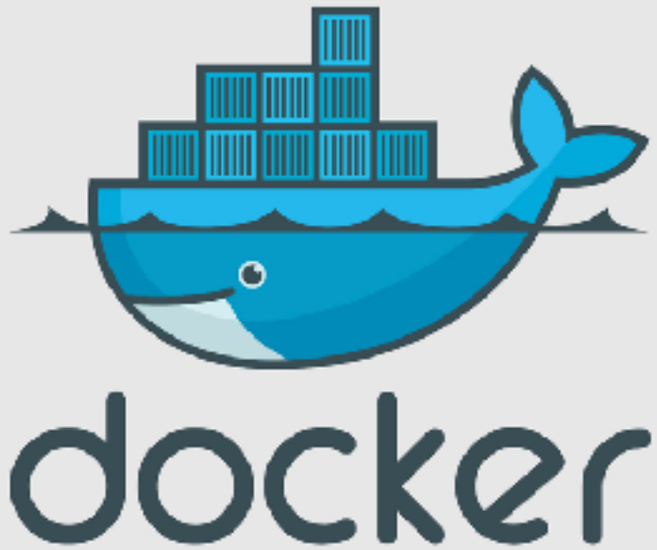
Node.js Containers

Wyatt Preul // jsgeek.com/dallas/



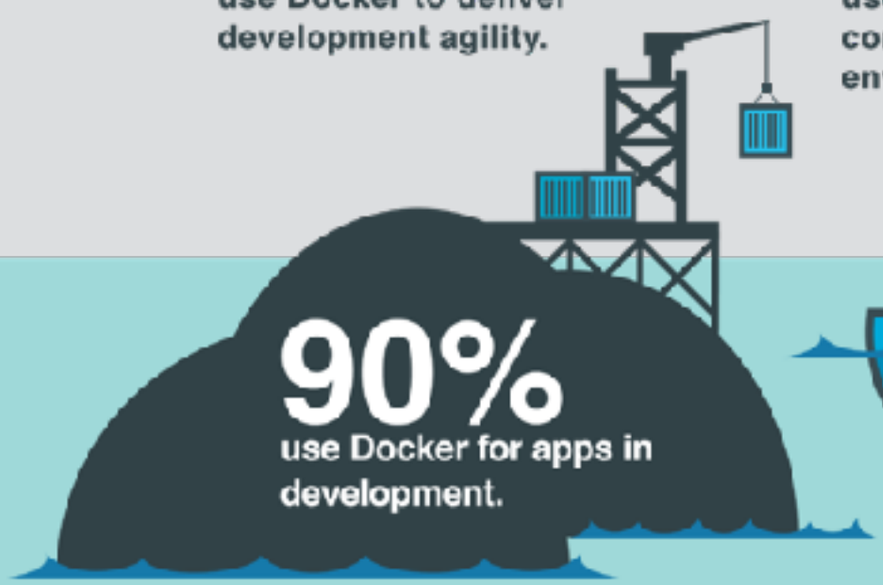
Using docker?

… in production?

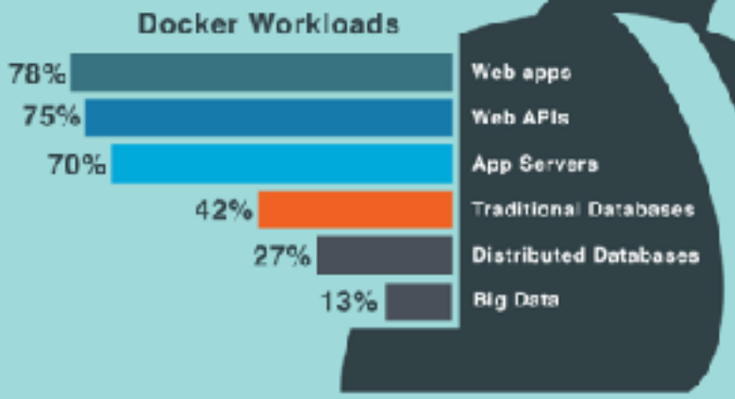
65%
use Docker to deliver **development agility.**

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.

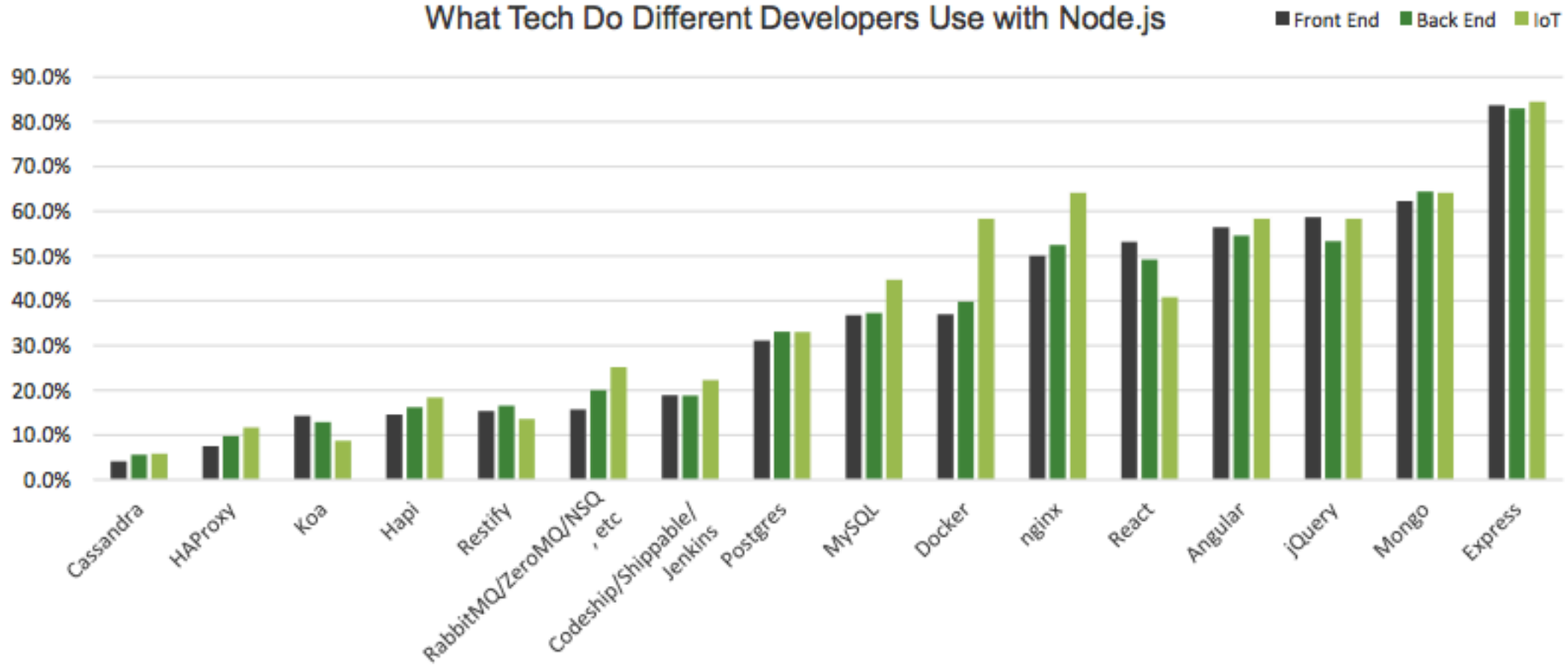
90%
plan dev **environments around Docker.**

80%
plan DevOps around Docker.

docker

Docker survey results: docker.com/survey-2016

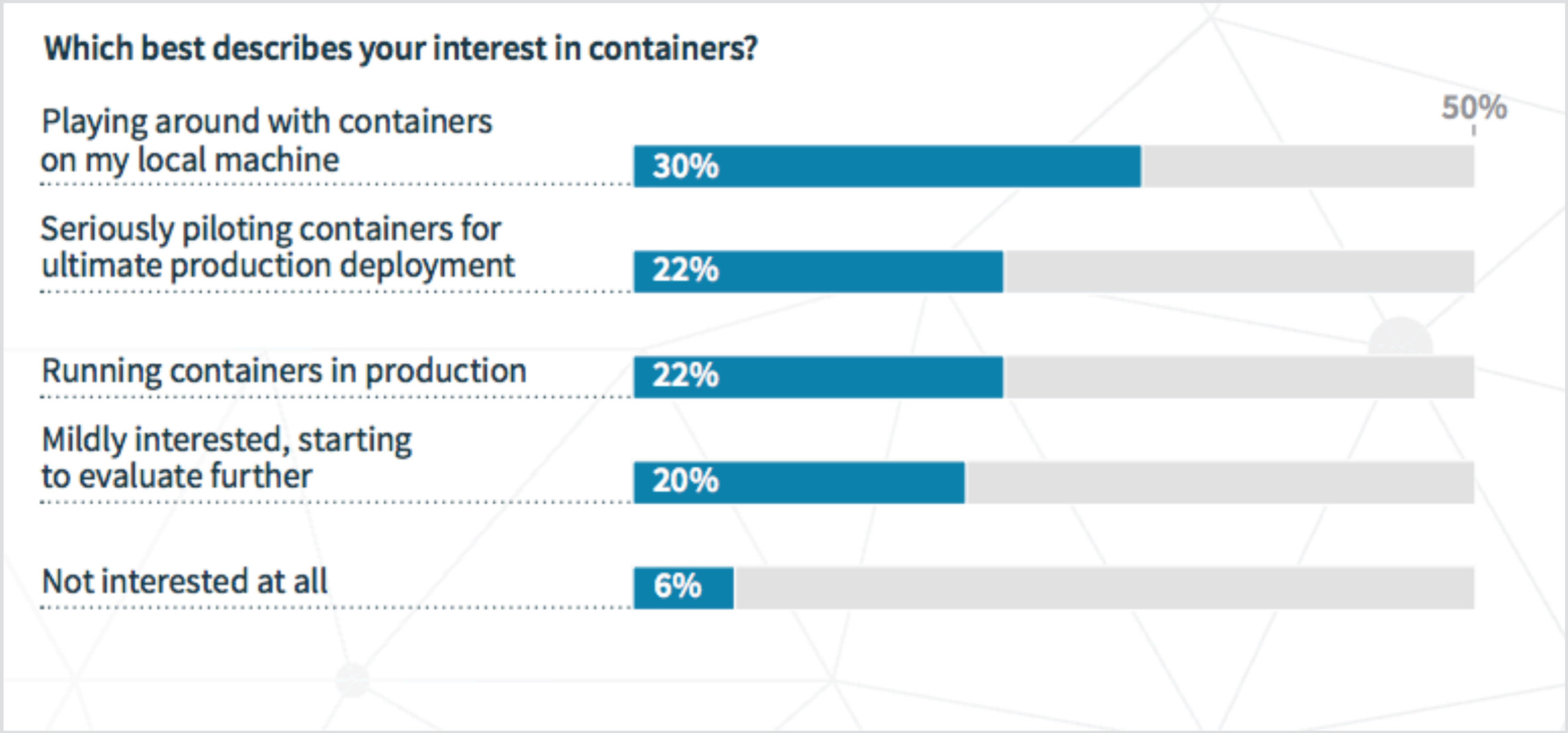
Tech Use with/in Node – Front End, Back End, IoT



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

2016 Node.js Survey Report

Java developer interest in containers

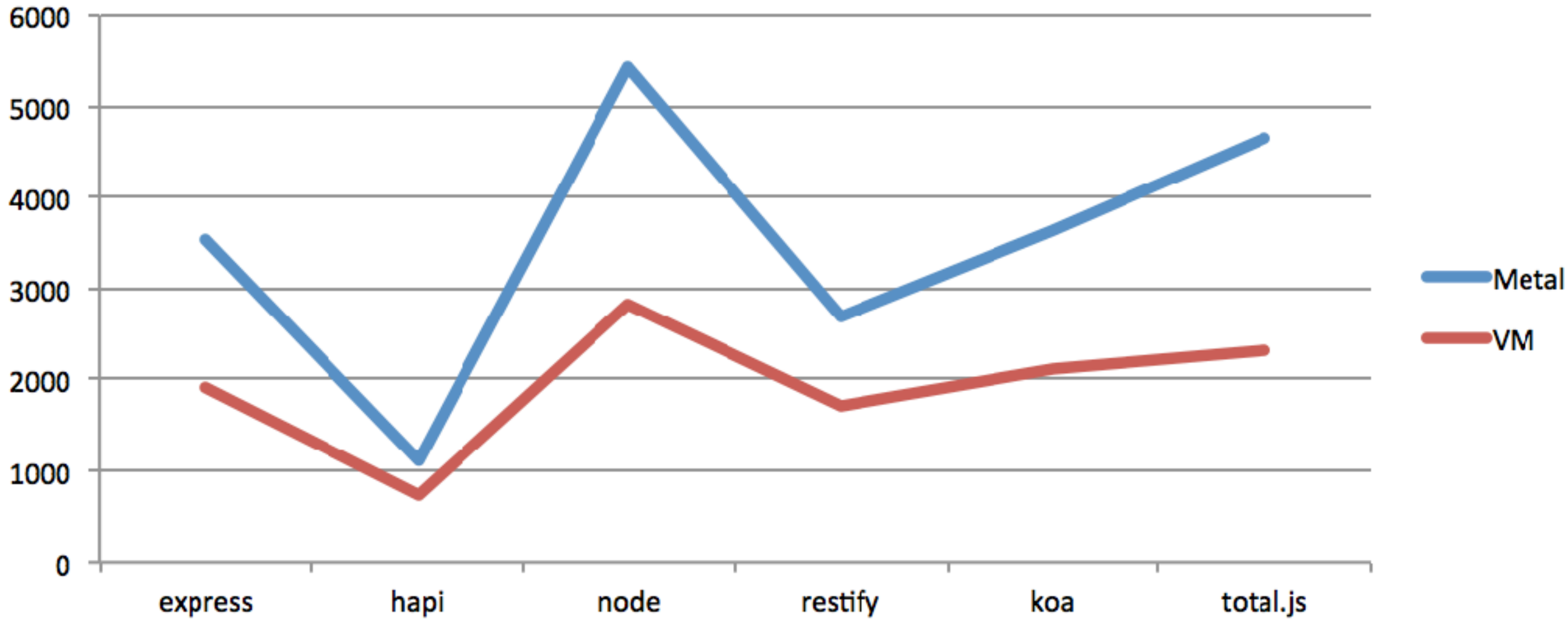


Lightbend 2016 Survey of JVM Devs

Benefits of Containers

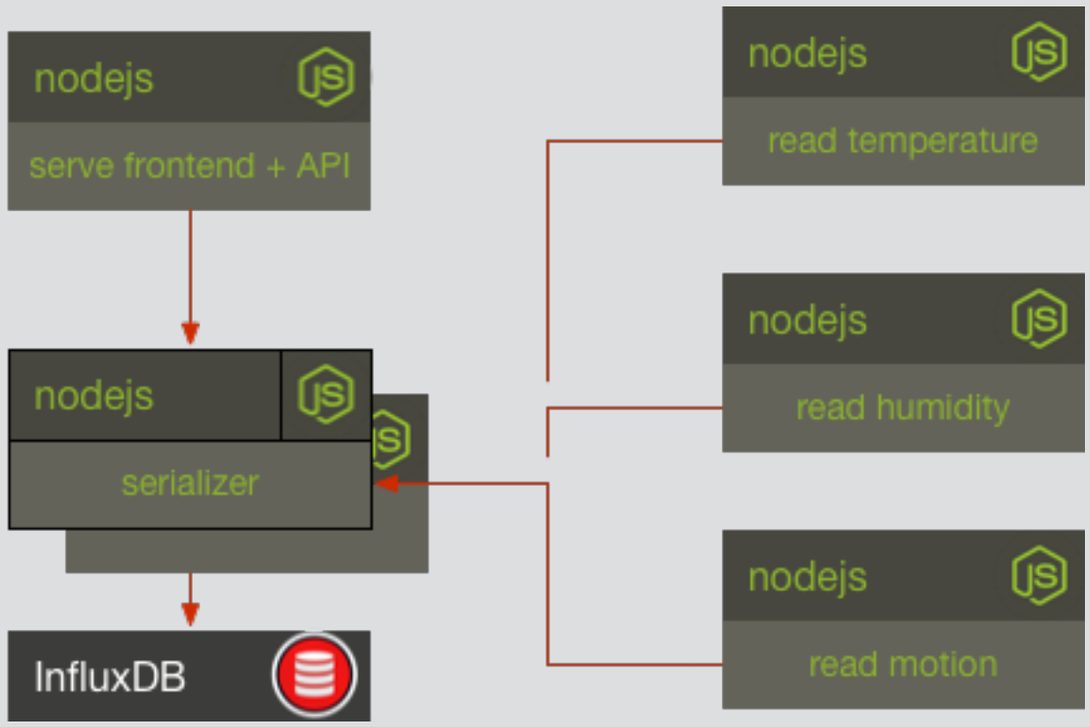
- consistent environments, immutable
- operations that developers can do, speeding up delivery of software
- OS level virtualization, more performant than VM

Hardware vs OS Level Virtualization Perf



CentOS 7, same datacenter, 2gb RAM, node 6.9.2

https://github.com/geek/NodeRestPerfTest



Using microservices?

… in production?

Microservices Popularity

- 68% of orgs are using or investigating microservices - NGINX 2016 Survey
- 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 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

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
    - watches a service catalog for changes in related services
    - consul, etcd, zookeeper, etc.
    - automatically reconfigures service upon state change
- open-source, free: github.com/joyent/containerpilot

nodejs-example

github.com/autopilotpattern/nodejs-example


Joyent
TRITON
Workshop Build
Docker Container
ContainerPilot
Consul
Docker Container
ContainerPilot
nodejs
serializer
Docker Container
ContainerPilot
nodejs
frontend
Docker Container
ContainerPilot
nodejs
read temperature
Docker Container
ContainerPilot
nodejs
read humidity
Docker Container
ContainerPilot
nodejs
serializer
Docker Container
InfluxDB
Docker Container
ContainerPilot
nodejs
reac motion

Node.js modules

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

Joyent®

Code & Demo

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


$ cd nodejs-example


$ EDITOR .
```

Recap

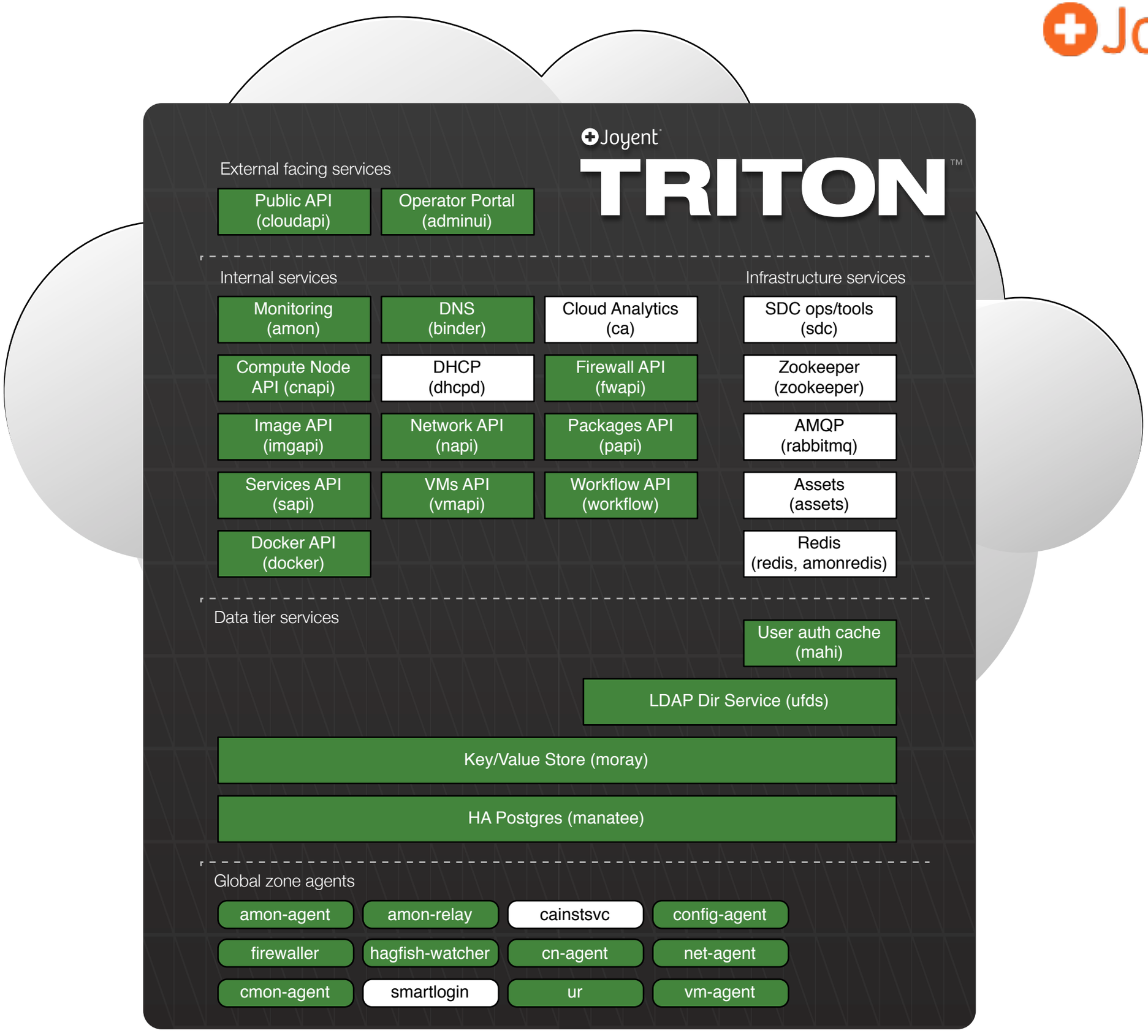
- Use ContainerPilot with Node.js docker containers (piloted module)
- Use consul for discovery (autopilotpattern/consul)
- Make microservices independently deployable and fault tolerant

Deploying to prod

Triton Provides

- Infrastructure as a Service
  - Network, Instance, Container, User and Security Management
- Containers as a Service
  - Docker - The data center is the docker host
- Software for Public and Private deployment
- High Performance
- High Security
- Open Source!

Joyent


Joyent
TRITON™
External facing services
Public API (cloudapi)
Operator Portal (adminui)
Internal services
Infrastructure services
Monitoring (amon)
DNS (binder)
Cloud Analytics (ca)
SDC ops/tools (sdc)
Compute Node API (cnapi)
DHCP (dhcpd)
Firewall API (fwapi)
Zookeeper (zookeeper)
Image API (imgapi)
Network API (napi)
Packages API (papi)
AMQP (rabbitmq)
Services API (sapi)
VMs API (vmapi)
Workflow API (workflow)
Assets (assets)
Docker API (docker)
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

- Docker Containers = Triton Instances
- No difference other than how the are managed
  - Docker - via Docker API (docker run etc)
  - Triton Instances - via CloudAPI (triton create)
- Based on LX instances
- Native networking
  - Each container get's it's 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. users.srvc.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
    - No need to manage multiple hosts/routing

Debugging Docker - Demo

```
$ docker exec -it nodejsexample_frontend_1 sh


$ top


# Add p tools to path
$ `


$ 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/dallas