Performance | Management | Security | Networking | Introspection | Utilization

Joyent®

7 characteristics of container-native infrastructure

Performance | Management | Security | Networking | Introspection | Utilization

container-native all the things

HI!
MY NAME IS

Casey Bisson

Performance | Management | Security | Networking | Introspection | Utilization

Public Cloud

Triton Elastic Container Service. We run our customer's mission critical applications on container native infrastructure

Private Cloud

Triton Elastic Container Infrastructure is an on-premise, container run-time environment used by some of the world's most recognizable brands

Public Clo

Triton Elasti

customer's mi
container nativ

joyent / sdc

Unwatch 83 | Unstar 346 | Fork

Main SmartDataCenter Project — Edit

97 commits | 2 branches | 0 releases | 17 contributors

branch: master | sdc / +

RELENG-613: deprecate sdc-zookeeper — latest commit 40c7667838

trentm authored 5 days ago

assets	arch diagram	6 months ago
docs	RELENG-613: deprecate sdc-zookeeper	5 days ago
etc	use the same form for the git urls as the other repos	6 months ago
tools	setup.bat: VMware Workstation path.	25 days ago
.gitignore	update repos.md, start reviewing notes	8 months ago
LICENSE	TOOLS-607 I for one, welcome my new open source license	8 months ago
Makefile	joyent/sdc#33: make clone-all-repos	5 months ago
README.md	doc: fix download link to coal~setup.bat .	25 days ago
package.json	TOOLS-623 Add repos list	8 months ago

README.md

SmartDataCenter

optimized to deliver next

Code | Issues | Pull requests | Wiki | Pulse | Graphs | Settings

SSH clone URL

git@github.

You can clone
or Subversion.

Clo

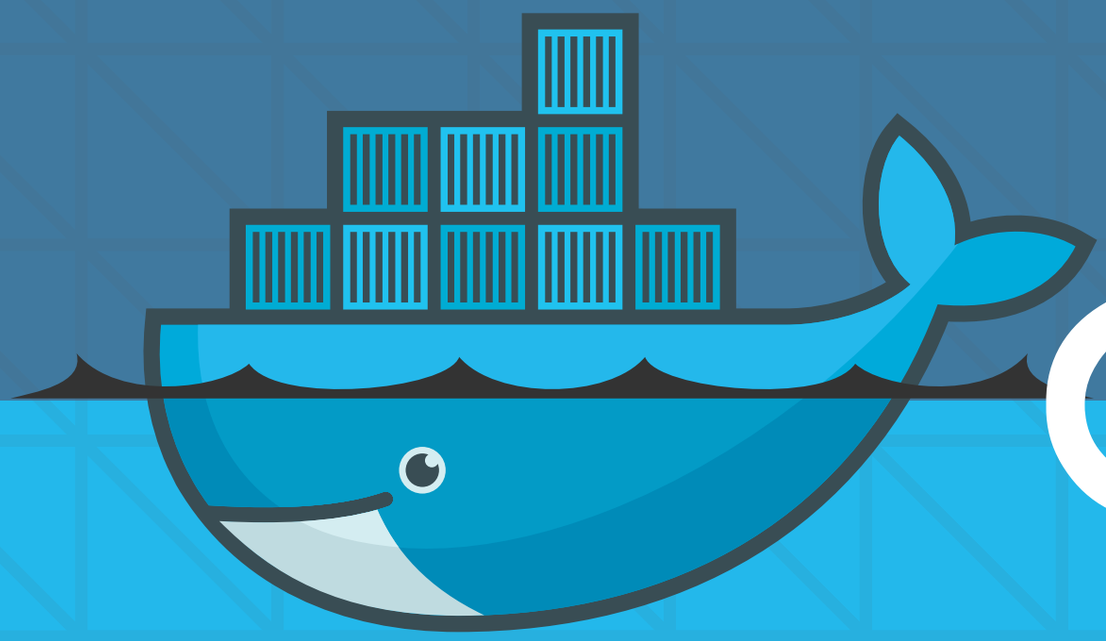
D

it's open source!

fork me, pull me: https://github.com/joyent/sdc

Joyent

The best place to run Docker



docker

Portability
From laptop to any public or private cloud

Great for DevOps
Tools for management, deployment & scale

Productivity
Faster code, test and deploy

some background

Infrastructure containers

- We use them like VMs
- They perform like bare metal
- We can pack 3x more containers than VMs on a single compute node
- They require no changes in application architecture
- First appeared in FreeBSD as jails in 1999-ish; defined for Solaris in 2002; Linux implementation in progress (user namespaces mainlined in 2013)
- Joyent has been using them to power our public cloud for almost a decade

Virtualization and Namespace Isolation in the Solaris Operating System (PSARC/2002/174)

John Beck, David Comay, Ozgur L., Daniel Price, and Andy T.
Solaris Kernel Technology

Andrew G. and Blaise S.
Solaris Network and Security Technologies

Revision 1.6 (OpenSolaris)

September 7, 2006

SmartOS container hypervisor

- Forked from Open Solaris in 2010
- Part of the Illumos community
- Open source
- Includes the core compute, network, and storage virtualization components:
  - **Zones** = truly secure containers
  - **Crossbow** = in-kernel network virtualization, offers unique NIC(s) for each container
  - **ZFS** = fast and secure virtualized filesystems

Triton (was SmartDataCenter)

- **Open source**, Mozilla licensed
- Complete **data center automation**; including…
- Automated boot and configuration of **compute nodes**
- Multi-tenant **management layer** for compute, network, and storage across N compute nodes
- Turns an entire data center into a single, **massive container host**
- Customer-facing services, including Docker and CloudAPI to support **self-service provisioning** and management
- Bring your own application scheduling and orchestration

Infrastructure containers

Like bare metal virtual machines

SmartOS

Secure container hypervisor

Triton

Container infrastructure automation

but...


docker

and

application containers

are

different

Container spectrum

infrastructure containers

application containers

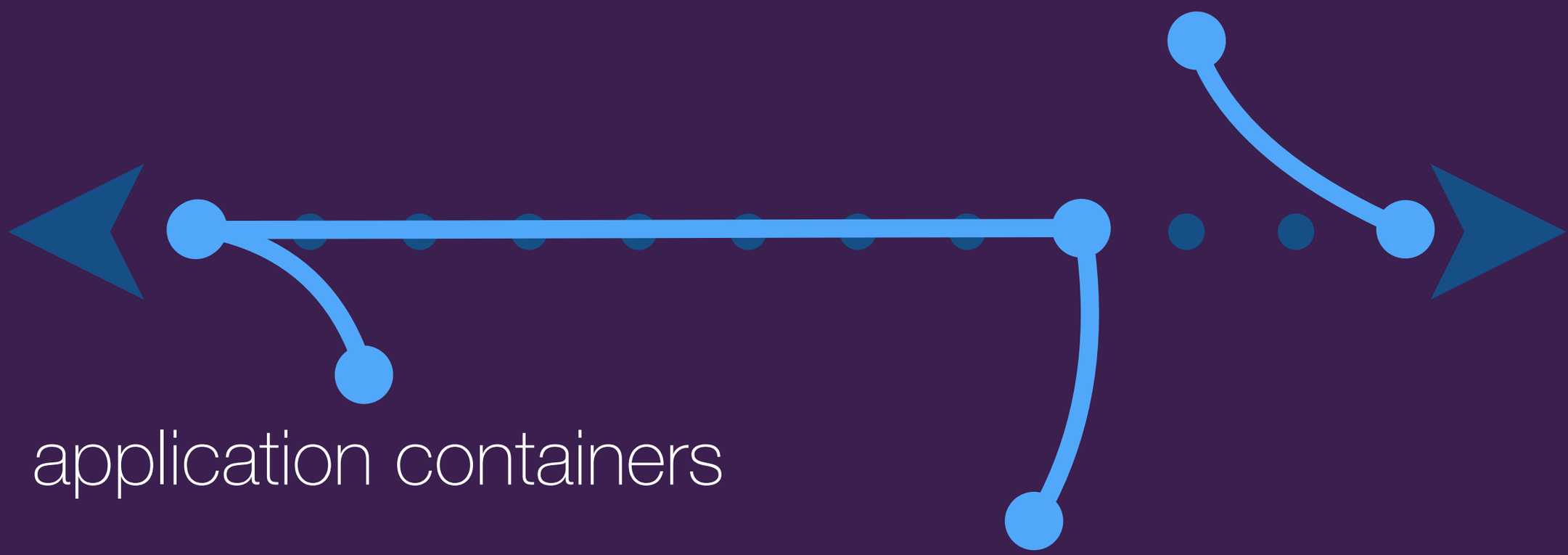
Container spectrum

slimmed-down
infrastructure containers

infrastructure containers

application containers

Container spectrum



application containers
are an opinionated
implementation

Docker container advantages

- Convenient imaging toolchain
- Easy sharing and re-use of images
- Dockerfiles are **machine+human readable** documentation for everything needed to build and run the application
- Docker Compose yamls are machine+human readable documentation for **how to assemble a set of containers** into a larger application
- Docker begs us to **automate discovery** and configuration
- Docker allow us to **imagine deployment and execution as a single step**

Docker
commoditizes
the possibility of

- immutable infrastructure
- automated, testable, repeatable deploys
- the kind of apps we all dream of

How 500px serves up over 500TB of high res photos

500px

General Architecture

The architecture of 500px can be thought of as a large **Ruby on Rails** monolith surrounded by a constellation of microservices. The Rails monolith serves the main 500px web application and the 500px API, which powers the web app, mobile apps and all our third party API users. The various microservices provide specific functionality of the platform to the monolith, and also serve some API endpoints directly.

The Rails monolith is a fairly typical stack: the App and API servers serve requests with **Unicorn**, fronted by **Nginx**. We have clusters of these Rails servers running behind either **HAProxy** or LVS load balancers, and the monolith's primary datastores are **MySQL**, **MongoDB**, **Redis** and **Memcached**. We also have a bunch of **Sidekiq** servers for background task processing. We currently host all of this on bare metal servers in a datacenter.

The microservices are a bit more interesting. We have about ten of them at the moment, each centered around providing an isolated and distinct business capability to the platform. Some of our microservices are:

- Search related services, built on **Elasticsearch**
- Content ingestion services, in front of **S3**
- User feeds and activity streams, built on **Roshi** and **AWS Kinesis**
- A dynamic image resizing and watermarking service
- Specialized API frontends for our web and mobile applications

We run our microservices in **Amazon EC2** and in our datacenter environment. They are mostly written in **Go**, though there are a couple outliers which use **NodeJS** or **Sinatra**. However, regardless of the language in use, we try to make all of our microservices good **12-factor** apps,

And we're not done with this problem yet! There are almost certainly more optimizations to be found, and we hope to keep pushing those response times down further and further in the future.

Workflow

We use **Github** and practice continuous integration for all of our primary codebases.

For the Rails monolith, we use **Semaphore** and **Code Climate**. We use a standard rspec setup for unit testing, and a smaller set of Capybara/Selenium tests for integration testing. Fellow 500pxer and professional cool guy **Devon Noel de Tilly** has written **at length** about how we use those tools, so I won't try to out do him -- just go check it out.

For our Go microservices, we use **Travis CI** to run tests and to create Debian packages as build artifacts. Travis uploads these packages to S3, and then another system pulls them down, signs them, and imports them into our private Apt repository. We use **FPM** to create packages, and **Aptly** to manage our repos. Lately, though, I've trying out **packagecloud.io** and I really like it so far, so we may be changing how we do this in the near future.

For deployments, we use a combination of tools. At the lowest level we use **Ansible** and **Capistrano** for deploys and **Chef** for configuration management. At a higher level, we've really embraced **chatops** at 500px, so we've scripted the use of those tools into our beloved and loyal **Hubot** friend, **BMO**.

Anyone at 500px can easily deploy the site or a microservice with a simple chat message like `bmo deploy <this thing>`. BMO goes out, deploys the thing, and then posts a log back into the chat. It's a simple, easy mechanism that has done wonders to increase visibility and reduce complexity around deploys. We use **Slack**, which is where you interact with BMO, and it makes everything really nicely searchable. If you want to find a log or if you forget how to do something, all you have to do is search the chat. Magical.

Other Important Apps

We monitor everything with **New Relic**, **Datadog**, ELK (**Elasticsearch**, **Logstash** and **Kibana**), and good old **Nagios**. We send all our emails with **Mandrill** and **Mailchimp** and we process payments with **Stripe** and **Paypal**. To help us make decisions, we use **Amazon's Elastic MapReduce** and **Redshift**, as well as **Periscope.io**. We use **Slack**, **Asana**, **Everhour**, and **Google Apps** to keep everyone in sync. And when things go wrong, we've got **Pagerduty** and **Statuspage.io** to help us out and to communicate with our users.

The Future, Conan?

photos
or
it didn't
happen

~~photos~~

or

~~it didn't happen~~

repo
or
it doesn't
work

public repo
or
it doesn't
work

public repo with

1. Dockerfile
2. docker-compose.yml
3. documentation, etc…

but...

pegleg sturgeon
@philsturgeon

Follow

Meanwhile, in my hometown of Bristol

Tom Jowitt
@tjbyte

Follow

@philsturgeon This is like working with Docker in production.

RETWEETS	FAVORITES
5	5

deploying containers in VMs slows performance

deploying
containers in VMs
adds complexity

deploying
containers in VMs
slows scaling

deploying containers in VMs wastes power

deploying
containers in VMs
causes global warming

deploying
containers in VMs
defeats the purpose

source

source

source

source

VMs cause complexity by…

- Requiring pre provisioning and configuration before they can host containers
- Mixing VM and container life cycles
- Reducing surface area for deployments, making pockets of unused capacity inevitable
- Reducing performance due to hardware virtualization
- Adding layers to networking
- Allowing room for us to turn them into pets
- Requiring re-packing of containers when scaling down

but...

containers
on bare metal Linux
struggle with
up-front costs

containers on bare metal Linux struggle with elasticity

containers on bare metal Linux struggle with data center automation

containers on bare metal Linux struggle with multi-tenant security

Docker has changed its security status to

It's complicated

–Docker's Jérôme Petazzoni

Bare metal Linux complexities

- No data center automation
- No security domain
- No performance isolation
- No elasticity…where do you scale to?

but...

there is

another way

most
infrastructure

pick

two

scenario

pick two:

- ~~elasticity~~

bare metal

- security
- performance

pick two:

hardware virtual machine

- elasticity
- security
- ~~performance~~

container-native infrastructure

pick three scenario

pick three:

bare metal containers

- elasticity
- security
- performance

So...
how do you secure it for bare metal?

Container anatomy

Container anatomy

Container anatomy

Container anatomy

Container anatomy

Container anatomy

Container anatomy

So...

can i run my Linux images on Triton?

yes!

ubuntu insights

Certified Ubuntu images now optimized for Joyent Triton containers

By Canonical on 18 June 2015

- *Joyent Expands Triton Elastic Container Infrastructure Beyond Docker, Adds Support for Container-Native Linux on Bare Metal*
- *Partners with Canonical to provide certified and supported Ubuntu images, optimized to run natively on bare metal on Triton Infrastructure Containers*

ubuntu
Supported by Canonical

Joyent

so...

i promised you a list

1:
the unit of compute
is a container

2:
you provision containers

3:
the containers run on bare metal

4:
the kernel offers real security

that means
no escape
no incursion

5:

the containers are protected from noisy neighbors

what's the point of

48 cores of bare metal

if a single container can dominate them all?

6:
every container gets a VNIC

6:
every container gets a VNIC (or two)

6:
every container gets a VNIC (or more)

because a
well-connected container
is a happy container

7:

You pay for containers

7:

You pay for containers (not VMs)

this is not

science fiction

this is

available

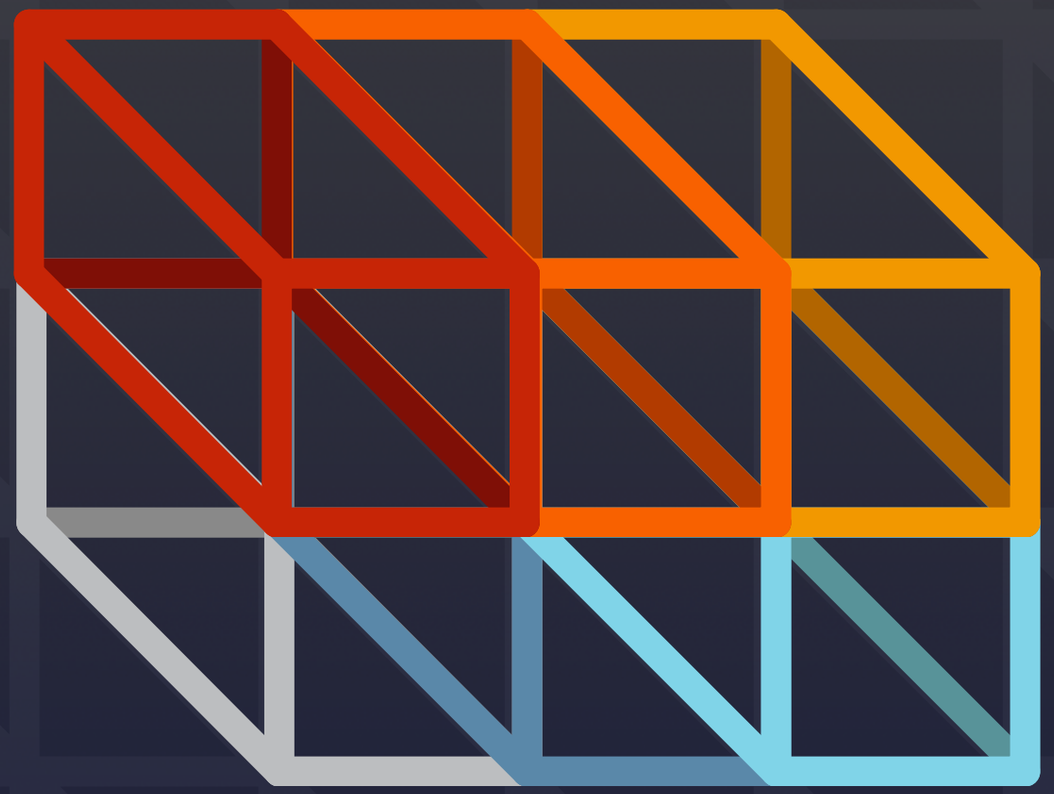
now

Performance | Management | Security | Networking | Introspection | Utilization

Joyent®

TRITON™

The best place to run containers.
Making Ops simple and scalable.



Demo time

Container-native everything

Application architecture

- Design for automated discovery and configuration

Workflow

- Embrace containerization in every step, from dev to deploy

Infrastructure

- Automates the infrastructure from the silicon to the container
- Enforces a single lifecycle: the container
- Eliminates pet VMs and servers
- Makes provisioning and execution a single step
- Delivers the promise of containers: automatable, testable, repeatable deploys

thank you

The 7 characteristics of container-native infrastructure

1. The unit of compute is a **container**, not a VM
2. You **provision containers**, not VMs
3. The containers run on **bare metal**
4. The containers are multi-tenant bare metal **secure**
5. Every container gets its **share**
6. Every container gets one or more **VNICs**
7. You **pay for containers**, not VMs