RSAConference2015

San Francisco | April 20-24 | Moscone Center

CHANGE

Challenge today's security thinking

SESSION ID: CSV-T07R

Something Awesome on Cloud and Containers

Christofer Hoff

VP, Security CTO
Juniper Networks
@Beaker

Rich Mogull

Analyst and Chief Executive Officer
Securosis, LLC
@rmogull

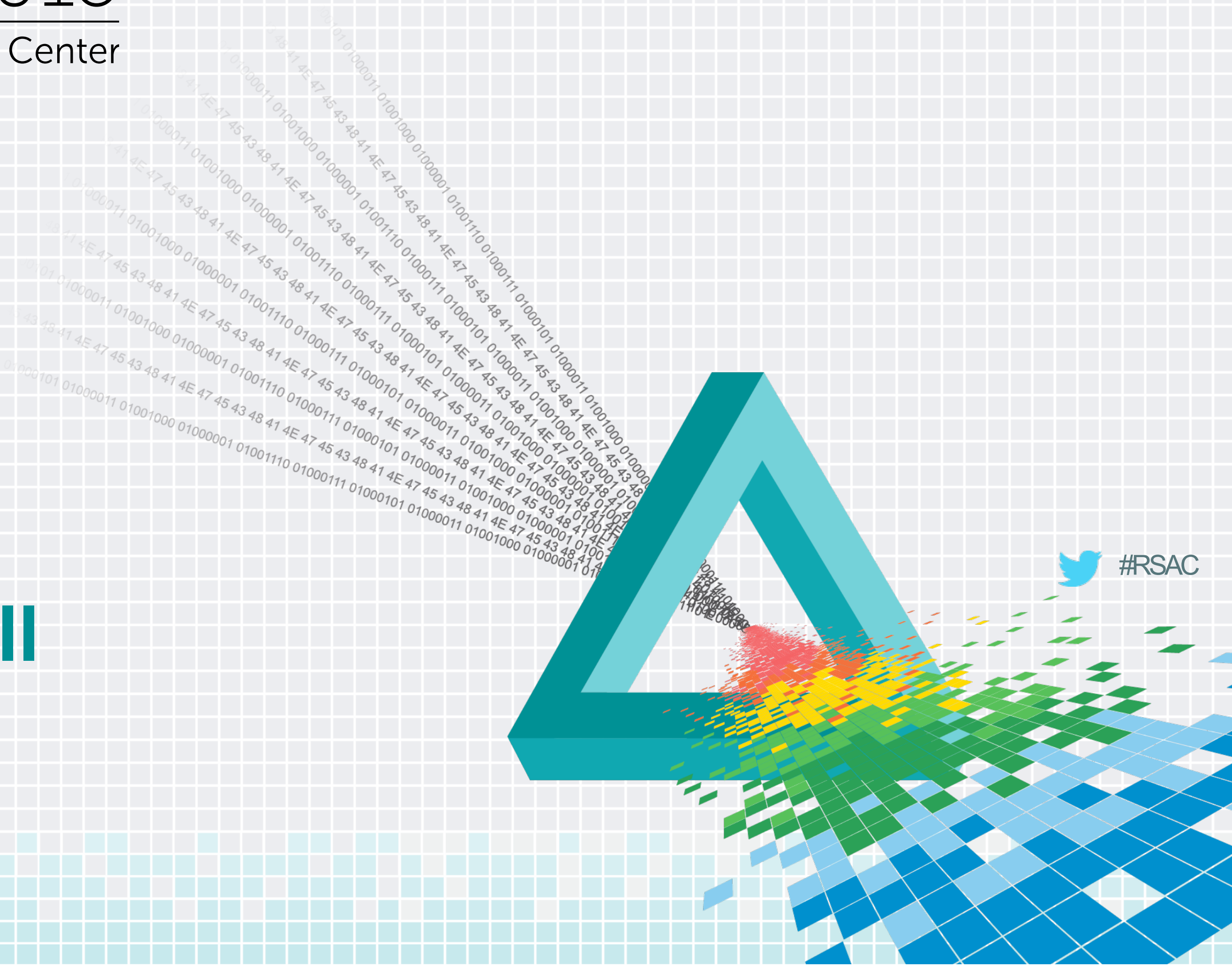
#RSAC

Expanded Agenda

- Cloud in a nutshell
- Software is eating (and digesting) the world…
- Containers & why they matter
- Applying this knowledge

RSA Conference2015

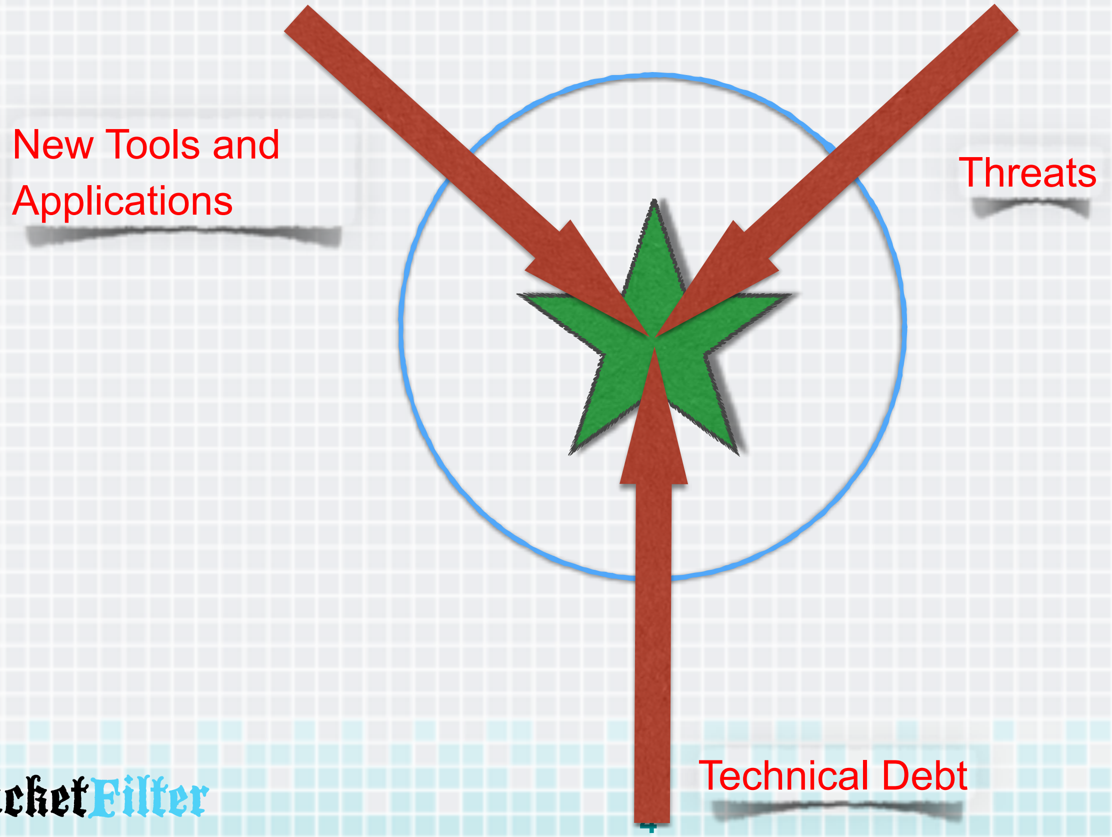
San Francisco | April 20-24 | Moscone Center



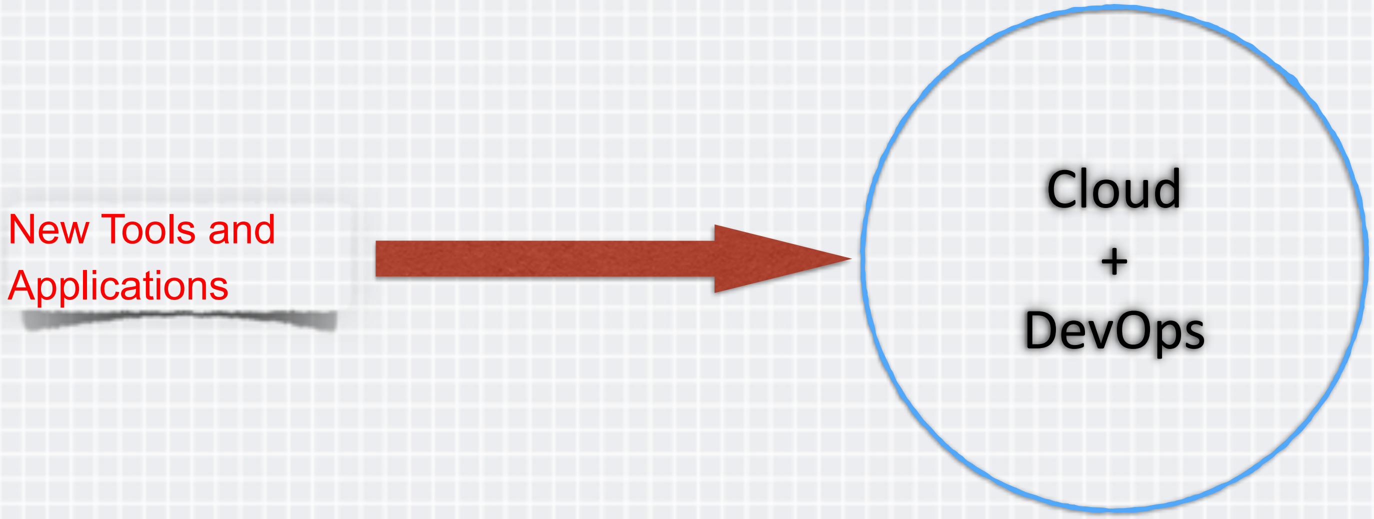
#RSAC

Cloud In a Nutshell

Security's Eternal Challenge



The Problem



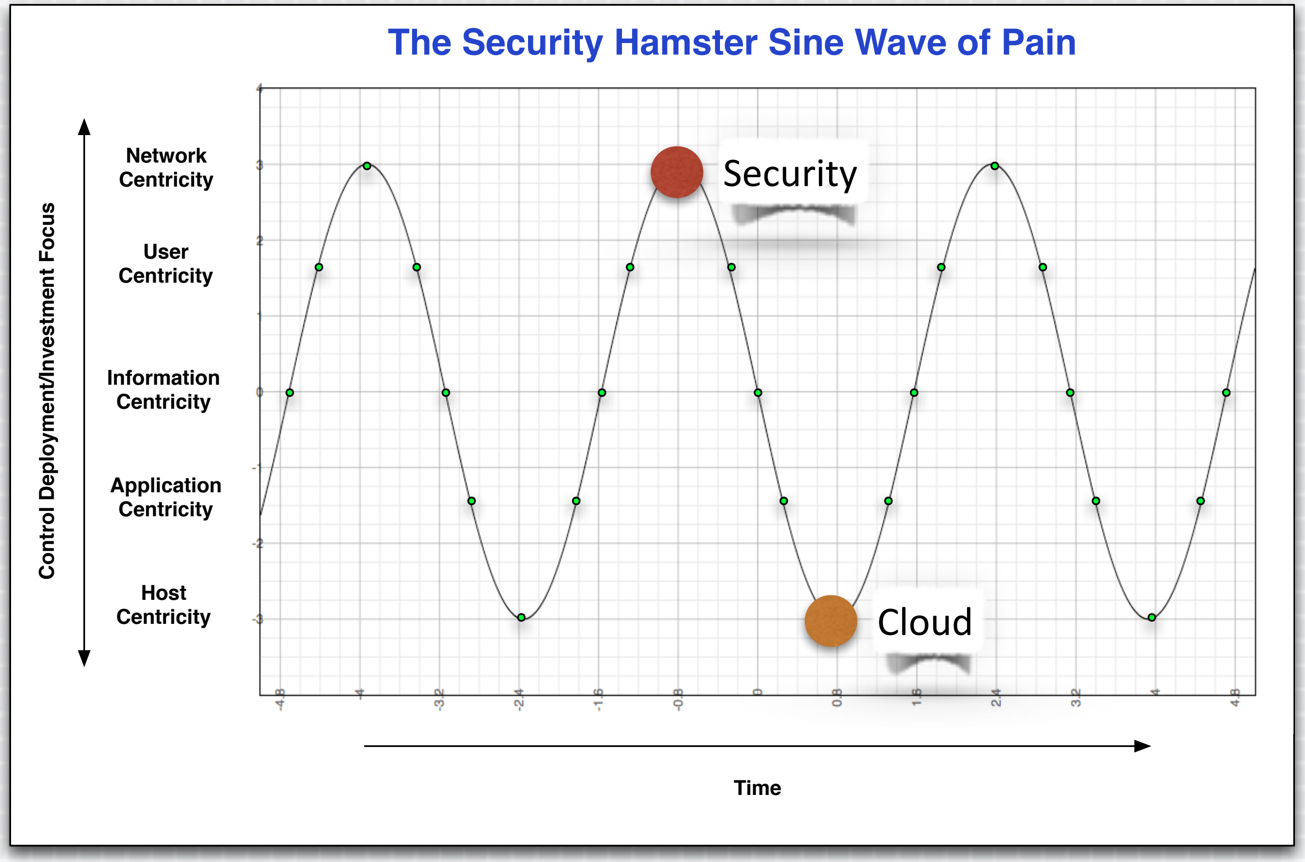
The “Enterprise” vs the “Cloud” Models

- Cloud is an **operational model**
- DevOps represents an **operational framework**
- Both enjoy their own definitional perversion
- Enterprises are adopting Cloud in various forms; Public/Private/Hybrid, IaaS/PaaS/SaaS
- The traditional silos and organizational dynamics of enterprises — driven by arbitrary economic models — are having a rough time with “DevOps”
- Why? Because **people are conflating the differences in the operational models with the need to adapt their frameworks for servicing it**

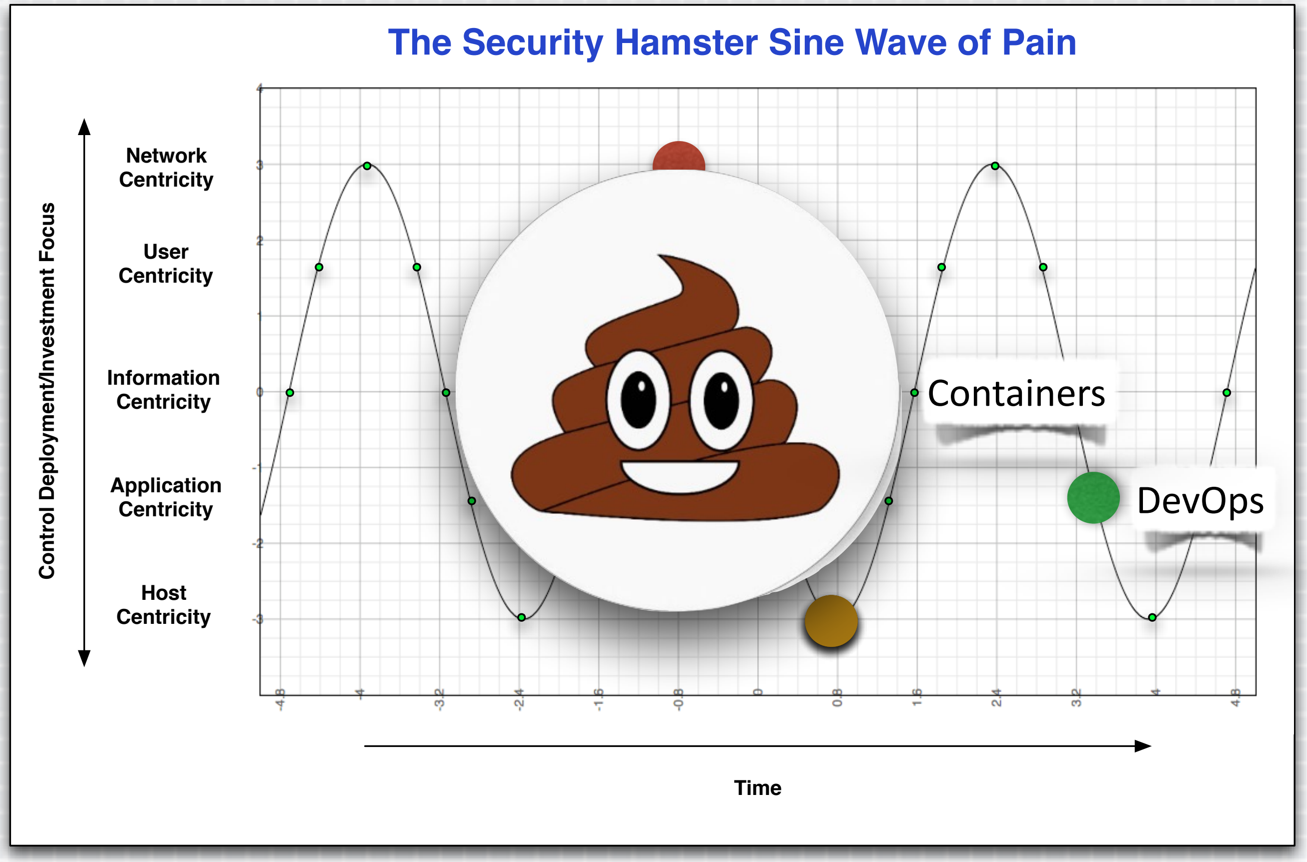
Physics Isn't Helping

- Velocity is static speed
- Acceleration is the increase of velocity over time
- Cloud and DevOps are still accelerating
- We haven't hit terminal velocity, and we can't predict when we will

2008 - 2013


The Security Hamster Sine Wave of Pain
Control Deployment/Investment Focus
Network Centricity
User Centricity
Information Centricity
Application Centricity
Host Centricity
Security
Cloud
Time

2015


The Security Hamster Sine Wave of Pain
Control Deployment/Investment Focus
Network Centricity
User Centricity
Information Centricity
Application Centricity
Host Centricity
Containers
DevOps
Time

Not Just for Unicorns Anymore

- Cloud, SaaS, PaaS, and IaaS, are becoming the norm
- Nearly all orgs we work with, from very large to small, are exploring public cloud deployments.
- These tend to be discreet projects.
  - That quickly expand

Natives vs. Tourists

- Cloud natives reap tremendous benefits in resiliency, economics, and agility.
- Cloud tourists deploy their existing operational models and frameworks onto a cloud service, losing most of the benefits of cloud.
- Typically due to lack of knowledge, institutional momentum, and arbitrary economic models.

IT Deconstructed

Infostructure	Content & Context - Data & Information
Applistructure	Apps & Widgets - Applications & Services
Metastructure	Glue & Guts - IPAM, IAM, BGP, DNS, SSL, PKI & Abstraction Layers
Infrastructure	Sprockets & Moving Parts - Compute, Network, Storage

Cloud Provider Innovation

Infostructure	Aurora, DynamoDB, DocumentDB, Elastic File System
Applistructure	Machine Learning, Kinesis, Scheduler, SQS, SNS, EMR
Metastructure	Config, Azure AD, CloudFormation, App Insights
Infrastructure	Route53, ECS, Glacier, Traffic Manager, VPC, ELB

Security as a Competitive Advantage

Infostructure	Cloud HSM, KMS, EKM, Key Vault
Applistructure	API Management, Cognito
Metastructure	IAM, CloudTrail, CloudWatch Alarms, MFA, SAML 2.0
Infrastructure	PFS, Security Groups, ACLs, Default Encryption

And this is how your vendors feel…

Lockin

- Infrastructure is cloud provider table stakes
  - Not that it doesn't matter (just try to choose a vLAN-based provider)
- The greatest opportunity for differentiation are in *infostructure*, *metastructure*, and especially, *applistructure*.
- No consistency across providers. No incentive for portability.
  - *However:* APIs = interoperability

Agility
Applistructure

The Security Challenge…

- If we don't have consistency in standards/formats for workloads & stack insertion, we're not going to have consistency in security
- Inconsistent policies, network topologies, and application service dependencies make security service, topology & device-specific
- Fundamentally, we need reusable and programmatic security design patterns; Controls today are CLI/GUI based
- Few are API-driven or feature capabilities for orchestration, provisioning as the workloads they protect

What’s Missing?

- Instrumentation that is inclusive of security
- Intelligence and context shared between infrastructure and applistructure layers
- Maturity of “automation mechanics” and frameworks
- Standard interfaces, precise syntactical representation of elemental security constructs < We need the “EC2 API” of Security
- An operational security methodology that ensures a common understanding of outcomes & “DevOps” culture in general

The Golden Rule of Cloud

If your security sucks now, you will be pleasantly surprised by the lack of change when you move to Cloud

RSAConference2015

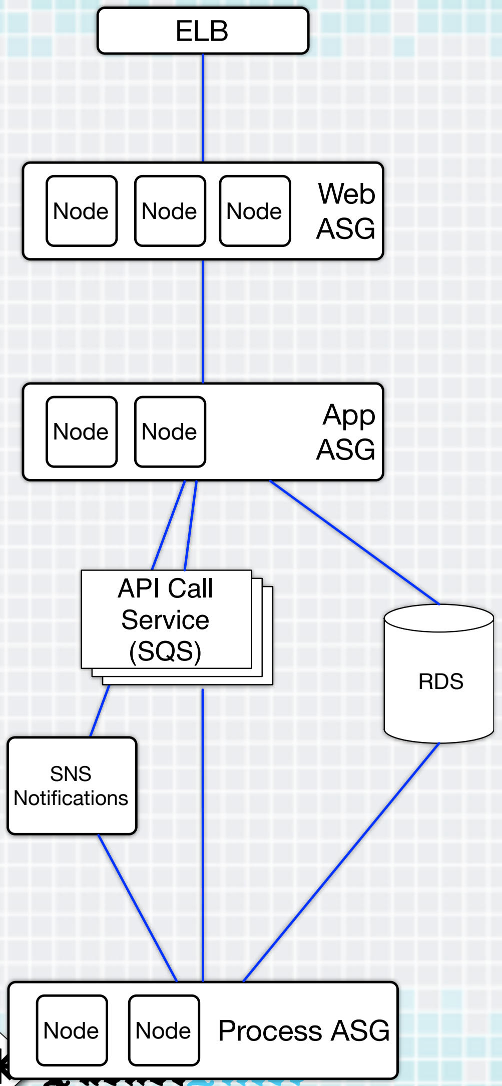
San Francisco | April 20-24 | Moscone Center

Software Is Eating (and Digesting) the World

#RSAC

Software Is Eating the World

- Software development is fundamentally disrupting the way companies operate and innovate
- Distributed scale-out application-centric architecture deployed via Platform-as-a-Service atop Infrastructure-as-a-Service is driving agility, reducing TTM and expanding reach (i.e. Cloud)
- Not just about creating the same sorts of apps faster, but moving from monolithic applications and packaging, to decomposed microservices (apps as services comprised of collections of APIs) This is the new perimeter
- Open: source, standards, and APIs are now mission critical components
- Agile, CI/CD and DevOps are the operational manifestations of these models and frameworks



- No fixed servers.
- No fixed connections.
- Workloads scattered across IaaS and PaaS.
- All components elastic.
- eak.
- S/IPS breaks.
- Assessment and monitoring break.

If you try to apply the traditional security operational framework to the new operational model!

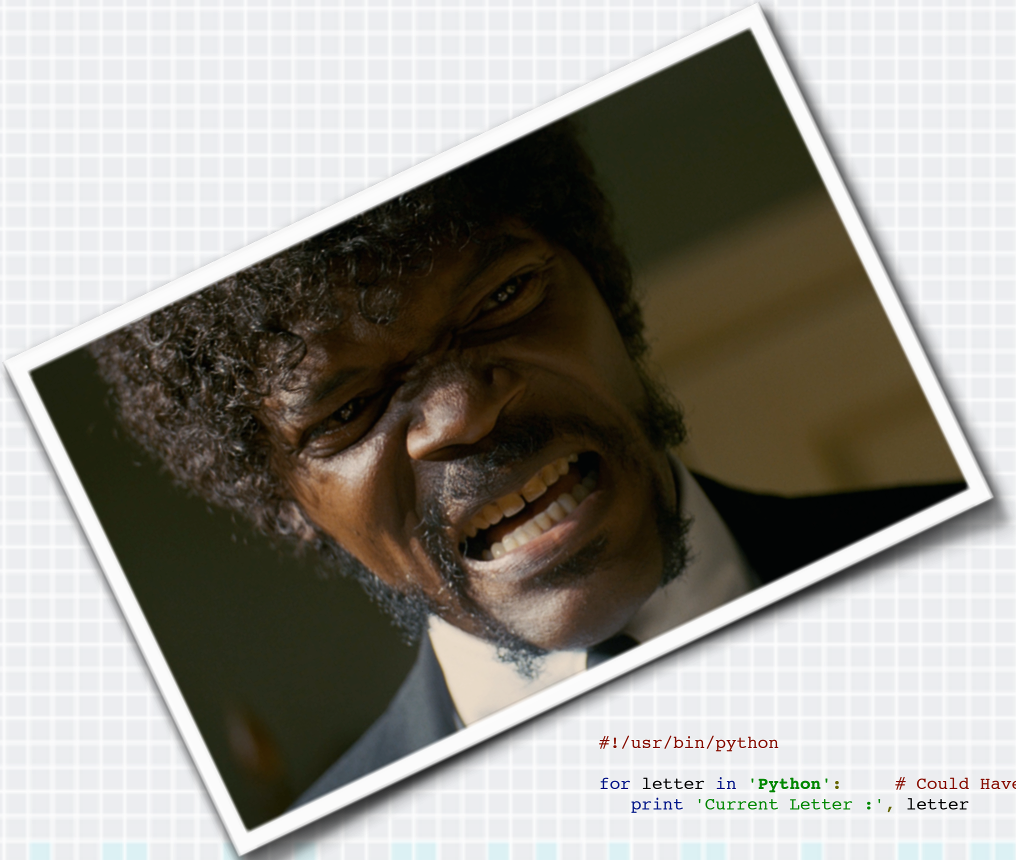
Security Says: “ENGLISH…Do You Speak It!?”

- DART, Ceylon, GO, F#, OPA, Fantom, Zimbu, X10, Haxe, Chapel
- Django, Pylons, Mojolicious
- CouchDB, Hadoop, Neo4J, MongoDB, Cassandra

- Python
- Ruby
- node.js
- Erlang
- Scala
- Clojure
- Groovy

The path of the righteous security man is beset on all sides by the inequities of the selfish and the tyranny of evil developers. Blessed is he who, in the name of scalability and good will, shepherds the weak through the valley of downtime darkness, for he is truly his brother's keeper and the finder of lost vulnerabilities. And I will strike down upon thee with great pwning vengeance and furious anger those who would attempt to poison and destroy my perimeter. And you will know My name is the Compliance Lord when I lay My stateful packet filtering vengeance upon thee.

Developers Say: “CODE…Do You Write It!?”



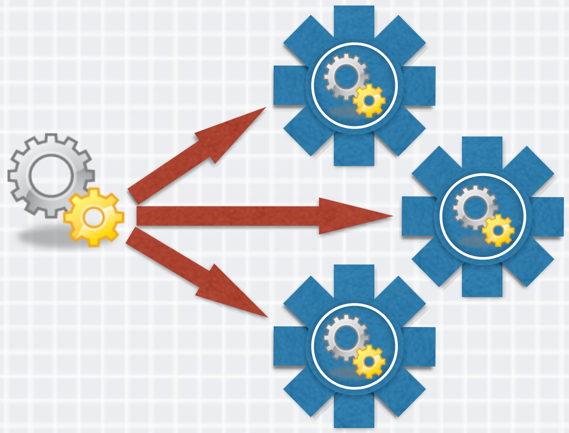
Say 'Python' again.

Say 'Python' again, I dare you, I double dare you...say ‘Python’ one more time!

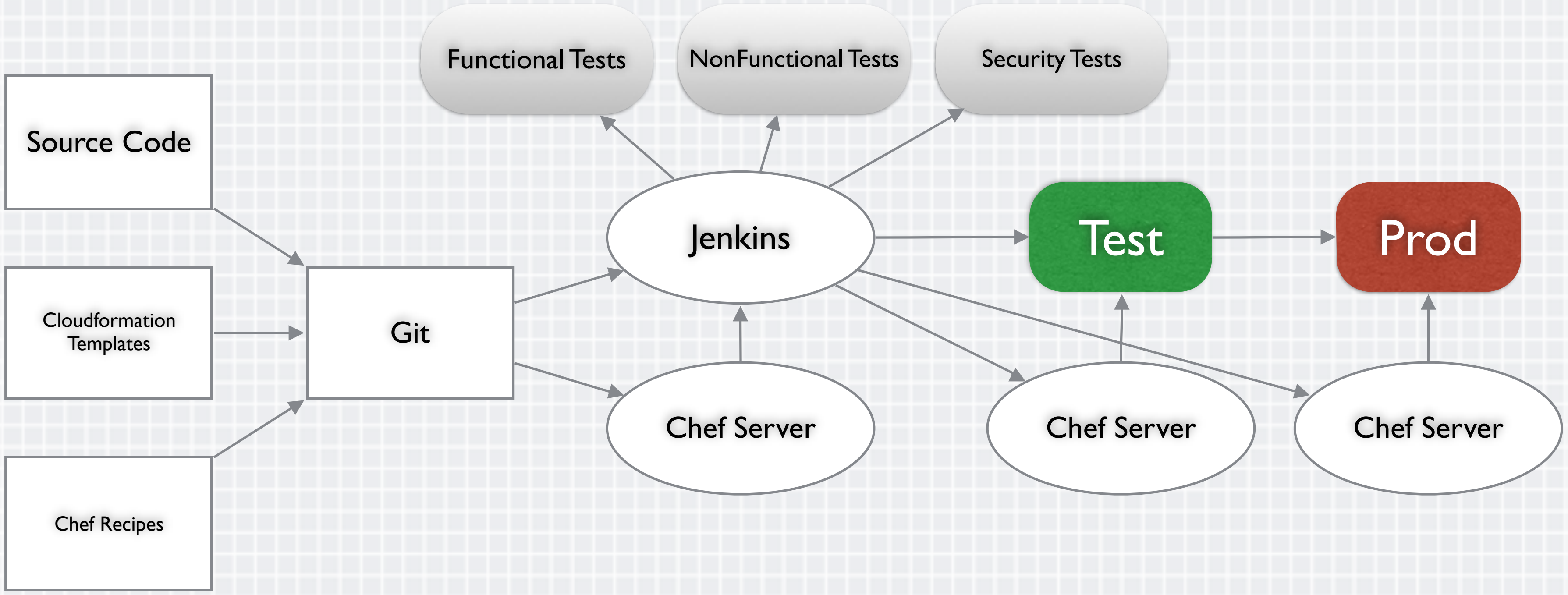
```
#!/usr/bin/python

for letter in 'Python':      # Could Have Lived This Way
   print 'Current Letter :', letter
```

Microservices and Workload-Centric

- Workload runs on the most appropriate platform or infrastructure.
- PaaS vs. IaaS vs. SaaS irrelevant - workload runs on best fit for the job.
- Workloads distribute based on real-time policies: performance, cost, capabilities.
- The smaller the service, the better the workload can adapt to the environment for performance and economic efficiency.



Continuous Deployment Disrupts Operations



Don’t Worry, Ops Feels the Same

The Golden Rule of Continuous Deployment

If your security sucks now, you will be pleasantly surprised by the lack of change when you move to Continuous Delivery

RSAConference2015

San Francisco | April 20-24 | Moscone Center

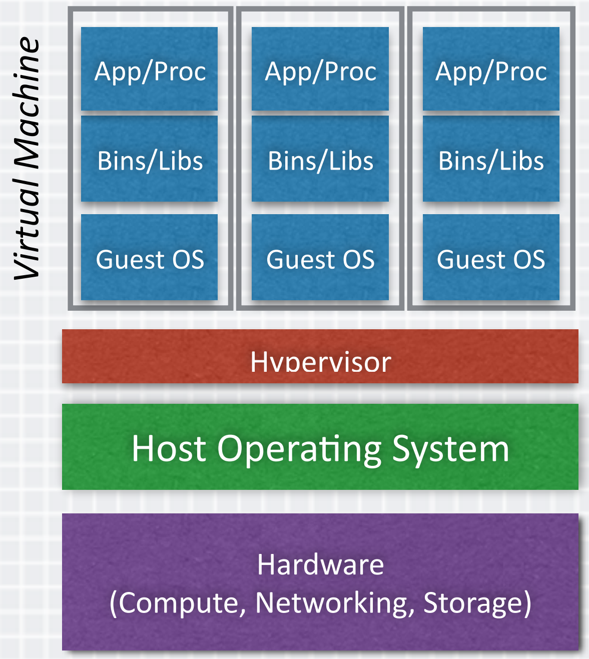
#RSAC

Containers & Why They Matter

What are Containers? And LXC?

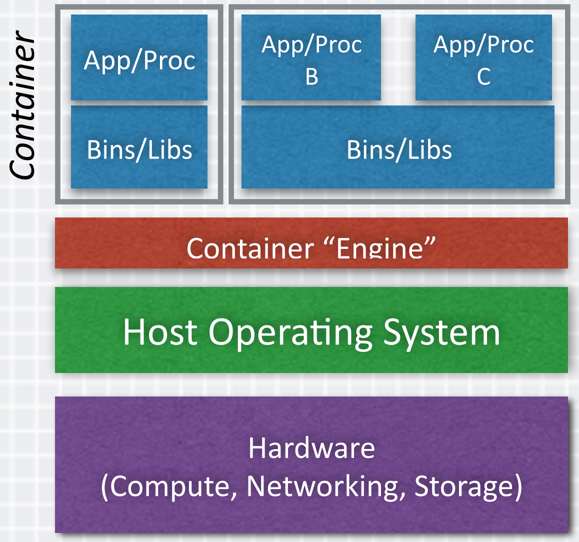
- "Containers" have existed in many forms for years: FreeBSD Jails, OpenVZ, Solaris Zones, and LXC for example.
- LXC (Linux Containers) is a userspace interface for the Linux kernel containment features to enable multiple isolated Linux processes to co-exist on a single Linux host
  - *"LXC containers are often considered as something in the middle between a chroot and a full fledged virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel."*
- This is enabled by the use of two Linux kernel process resource management solutions:
  - **Cgroups** (control groups) are a resource management solution providing a generic process-grouping framework which limits and prioritizes system resources (CPU, memory, I/O, network, etc.)
  - **Namespaces** allow for lightweight process virtualization and enables processes to have different views of the system (mnt, pid, net, pic, uts, user)

Comparing Virtual Machines…



- Virtual Machines (VMs) are best used to emulate and allocate chunks of hardware resources.
- The isolation enabled by hypervisors generally represent less of an attack surface than exposing the entire host OS to the VMs themselves and provides an abstracted/protected layer
- In the case of a Type-1 hypervisor, there is no underlying host OS
- Each VM includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may entail many gigabytes of storage and must be patched independently

…and Containers?



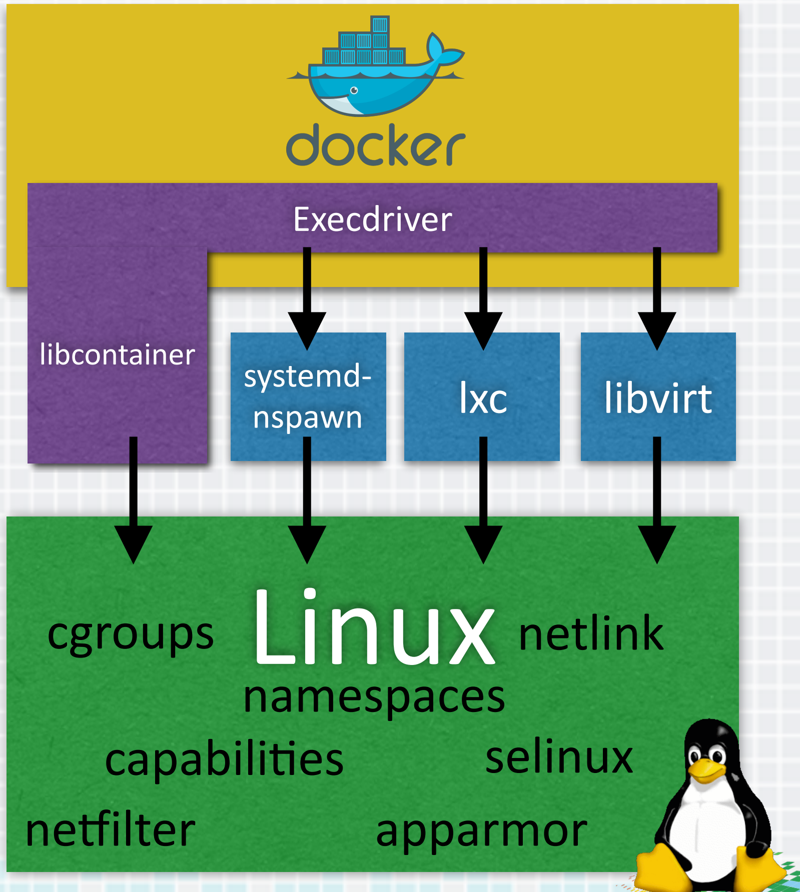
- Containers operate at the process level, which makes them very lightweight and perfect as a unit of software delivery.
- A container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers.
- Thus, it enjoys *many* of the resource isolation and allocation benefits of VMs but is much more portable and efficient.

So What Is Docker? [non-technical]

- Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications.
- Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments.
- Docker is evolving into a platform that includes tools such as Machine, Swarm and Compose to enable simpler, integrated docker engine deployment, clustering, and distributed multi-application orchestration

What Is Docker [more technical]

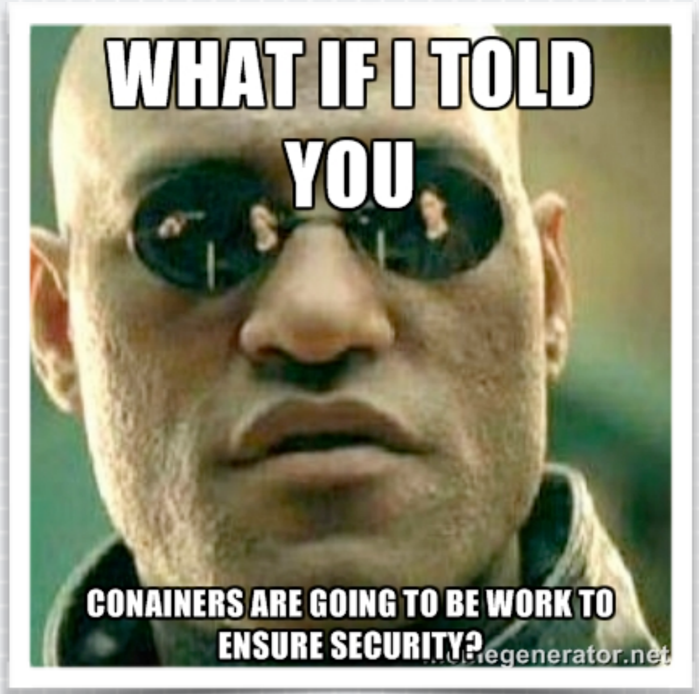
- Docker is an open-source project written in Go, that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux.
- Docker implements a high-level API to provide lightweight containers that run processes in isolation, building on top of facilities provided by the Linux kernel (primarily cgroups and namespaces)
- A Docker container does not require or include a separate operating system. It relies on the kernel's functionality and uses resource isolation (CPU, memory, block I/O, network, etc.) and separate namespaces to isolate the application's view of the operating system.
- Docker accesses the Linux kernel's virtualization features either directly through the provided libcontainer library or indirectly via libvirt, LXC or systemd-nspawn.
- Libcontainer enables containers to manipulate Linux namespaces, control groups, capabilities, AppArmor security profiles, network interfaces and firewalling rules in a consistent and predictable way.



Are Containers "Secure?"

That is the wrong question.

So? TL;DR…



- As you will see, to adequately secure environments that utilize containers, you will be required to know more about the internals and hardening of the underlying host OS to ensure the integrity and security of said platforms **<u>more</u>** than you might with a hypervisor and VMs.
- <u>Or</u> you should/can find a PaaS/Container management solution that includes security capabilities so you don't have to.
- <u>Or</u> you should/can just "give up" and run your containers within VMs…

4 Dimensions of Container Security

1. Underlying Host's Linux distribution kernel and its support for hardening, namespaces, cgroups and capability mapping including capabilities and what you do to harden the OS
2. The Container platform layer/engine & APIs: e.g. LXC or Libcontainer
3. Security of the access to the control plane of the host OS, the container engine, scheduler(s) and application deployment platform components
4. Security of the process(es) within the Container

Related: The grouping of containers into trust "zones" (logical/physical) and the networking capabilities to do so (i.e. br0 interfaces insufficient)

But First, A Selfie…

Some Important Questions:

1. Do you run a Linux OS in production?
2. Do you allow developers to run processes atop Linux?
3. Do you allow multiple processes per host?
4. Do you allow processes running atop linux to do so as root?
5. Do you know what 'setenforce 1' means and where you would implement it and why?
6. How do you define a security boundary?

If the vessel isn't secure, how can the containers be?

"Containers Don't Contain" & "Tupperware Don't Tupper"

- Run Docker Engine with AppArmor or SELinux atop a GRSEC kernel to provide containment where isolation is appropriately scoped
- Don't run untrusted processes with root privileges & enable user namespaces
- Map groups of mutually-trusted containers to separate machines; and
- If you really, really care about isolation and reducing attack surface, run containers in VMs or one container per machine…

The Golden Rule of Containers

If your security sucks now, you will be pleasantly surprised by the lack of change when you move to Containers

RSAConference2015

San Francisco | April 20-24 | Moscone Center

Applying This Knowledge

#RSAC

Apply Slide

- Return to your host-based security fundamentals that you thought you could get rid of because of Cloud…
  - …which you never did because you still have to armor your guest OS and apps anyway, right? Right!?
- Leverage DevOps and automation so that security is integrated (Cloud and Containers) into your least common denominator and per-unit of deployment (bare metal, VM, Container, Micro-service…)
- Don't try this sober.