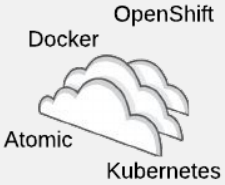
Containerization: Migrating Existing Applications

Chris Collins (@ChrisInDurham)
Sr. Automation Engineer, Duke University

Scott McCarty (@fatherlinux)
Sr. Principal Technical Marketing Manager, Red Hat

Container Journey



Evaluate Technology

Experiment

Quick Win

Inventory Applications

Determine Technology

Containerize

Real World Experience

Quick Win: Combatting a Denial of Service Attack

PROBLEM

- DDOS Attack targeting Duke.edu
- flooding load balancers
- all load- balanced services impacted
- Duke.edu down

SOLUTION

- Duke.edu container image
- AWS Docker hosts
- External DNS for duke.edu pointed to AWS
- Internal traffic kept inside Duke

THE RESULT

- Duke.edu unaffected for internal customers
- Duke.edu traffic handled by AWS for external customers/DDOS
- 30-minute migration!
- Attack removed from load-balancers
- Other load-balanced services back to normal

Inventory: Large Scale Web Applications

PROBLEM

- WordPress websites needed at large scale
- Individual server doesn't scale with load
- Multi-server states hard to synchronize
- Development environment difficult to replicate

SOLUTION

- Containerized service
- Site code distributed in images
- Single-service containers are easy to swap
- Development & production built from images

THE RESULT

- Enterprise-level - ~8,800 sites and ~29,000 users
- Quickly scalable, cloudburst-able
- Interchangeable front-end containers = automated sites with custom DNS and HTTPS
- Developers have taken ownership of deployment, can update, roll-back, clone w/o sysadmins

Inventory: Research Computing, Multiple Stacks

PROBLEM

- Researchers want custom tool chains
- IT wants researchers on shared infrastructure
- Researchers need to be able to reproduce/share environment

SOLUTION

- Run every job in a custom Docker formatted container
- Keep archive of old container images with log of which version was used for which job run

THE RESULT

- Self-service: Researchers at Duke are starting to build their own Docker formatted container images to run their analysis
- Cloudburst-able: Workloads scale out of on-prem HPC to AWS/Azure cloud
- Processing job shipped to data, not vice-versa

Inventory: IDM in a Container

PROBLEM

- Legacy IDM Apps
- Unpredictable behavior after patching
- Result: infrequent patching
- Inability to easily upgrade
- Result: ancient hardware

SOLUTION

- Build IDM apps in containers
- Jenkins builds every 4 hours w/latest patches
- Automated testing notifies of failures
- Last “known good” image kept

THE RESULT

- “Known Good” image always available; uptime assured
- Breaking patches can be investigated while “Known Good” images are kept in use
- Extremely portable
- Hardware independent
- Other environment can be setup, tested, torn down in minutes

Inventory: Red Hat IDM in a Container

PROBLEM

- Dozens of daemons, their own data volumes
- Libraries share config files
- Installer logic expects single machine
- Knowledge embedded in init/systemd
- Extensive initial setup needed

SOLUTION

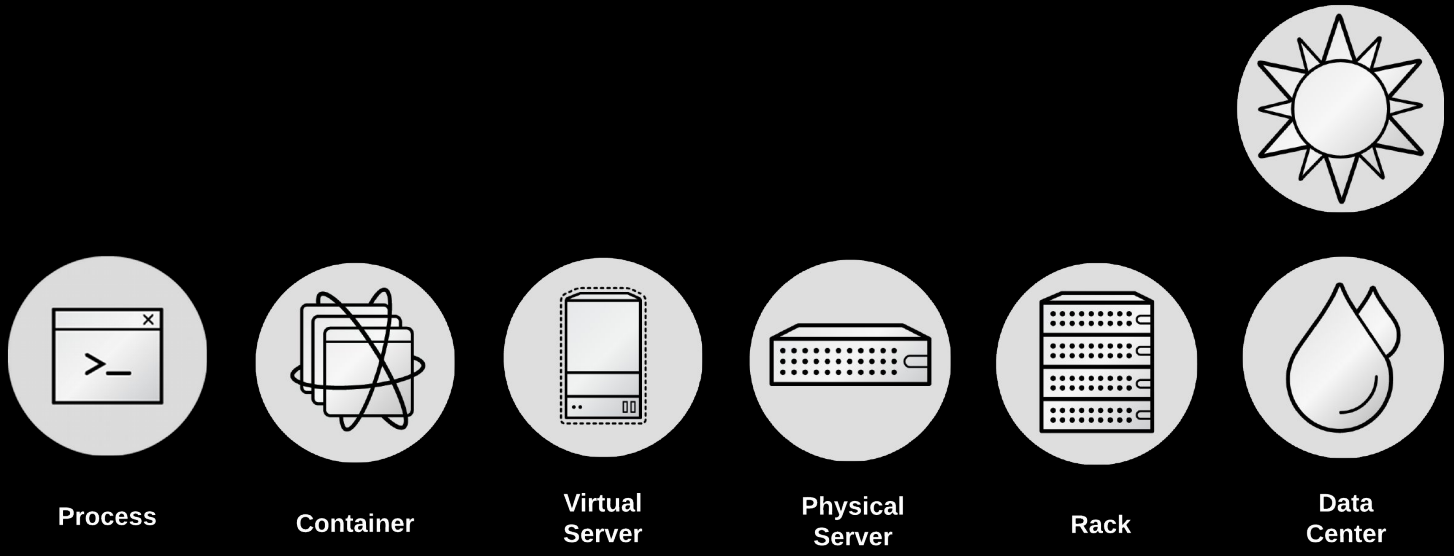
- Build IDM (aka FreeIPA) in single container
- Simplified install, upgrade, rollback
- Minimize data volumes
- Template based data volume population

THE RESULT

- Better software delivery
- Better technical demarcation between vendor and customer
- Extremely portable
- Environments can be setup, tested, torn down in minutes
- Same pattern for other products and services: OpenShift, OpenStack (yes, OSP), sssd, etc.

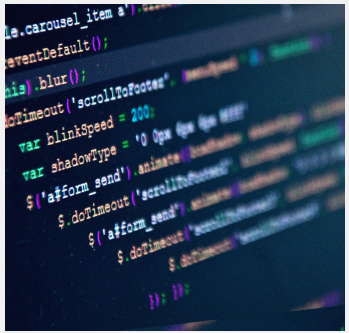
Workload Characterization Guidelines

The Tenancy Scale



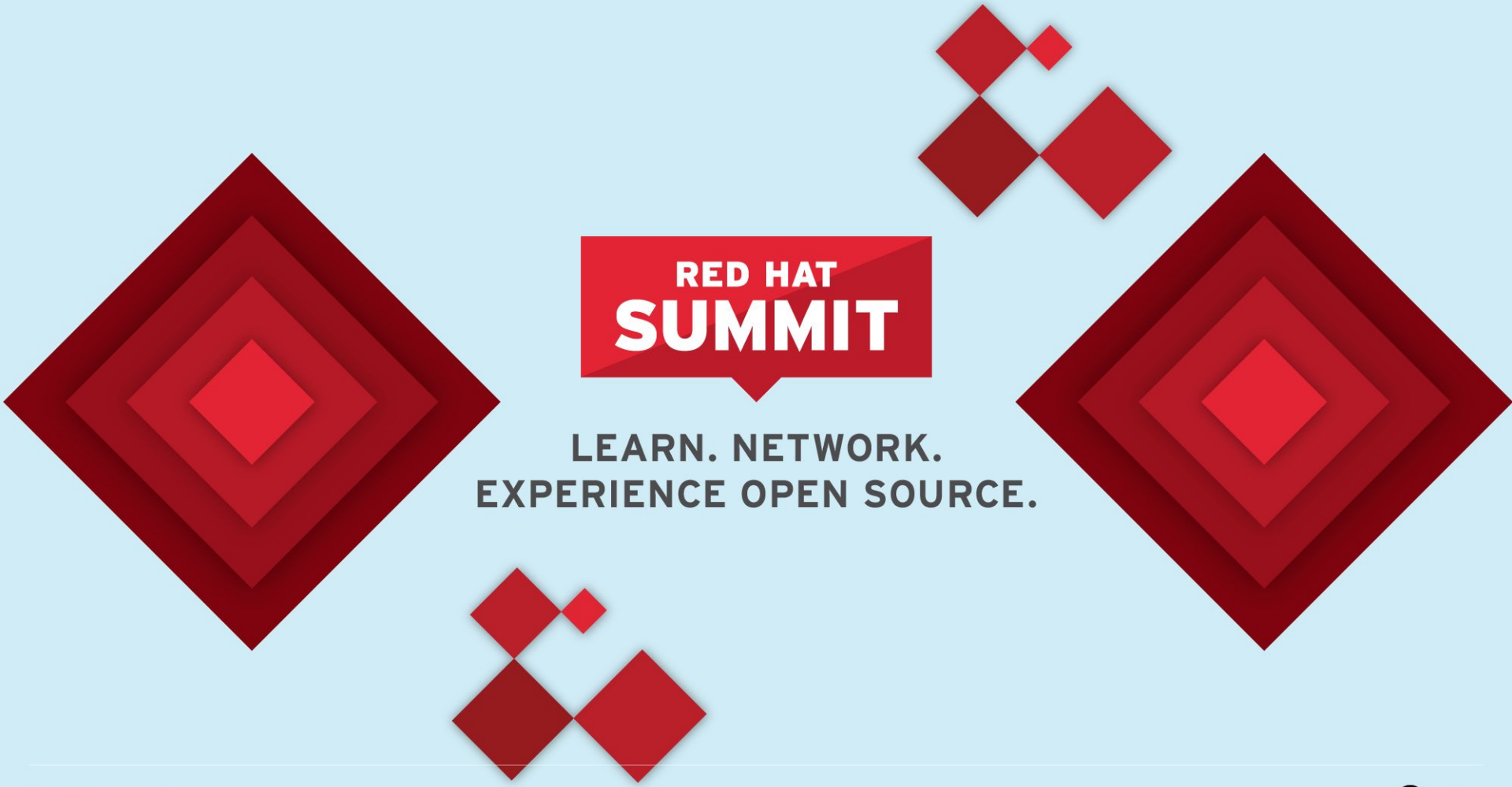
Application Containers

1. Code: mysqld
2. Configuration: /etc/my.cnf
3. Data: /var/lib/mysql
4. Other stuff :-)



Level of Effort

	Easy	Moderate	Difficult
Code	Completely Isolated (single process)	Somewhat Isolated (multiple processes)	Self Modifying (e.g. Actor Model)
Configuration	One Configuration File	Several Configuration Files	Configuration Anywhere in Filesystem
Data	Data Saved in Single Place	Data Saved in Several Places	Data Anywhere in Filesystem
Secrets	Static Files	Network	Dynamic Generation of Certificates
Network	HTTP, HTTPS	TCP, UDP	IPSEC, Highly Isolated
Installation	Packages, Source	Installer and Understood Configuration	Installers (install.sh)
Licensing	Open Source	Proprietary	Restrictive & Proprietary



Further Reading & Citations

- Container Tidbits: When Should I Break My Application into Multiple Containers? http://red.ht/22xKw9j
- Architecting Containers Part 4: Workload Characteristics and Candidates for Containerization: http://red.ht/1SBw9ql