

A Greybeard's Worst Nightmare

How Kubernetes and Containers are re-defining the Linux OS

Daniel Riek, Red Hat
April 2017

Greybeard

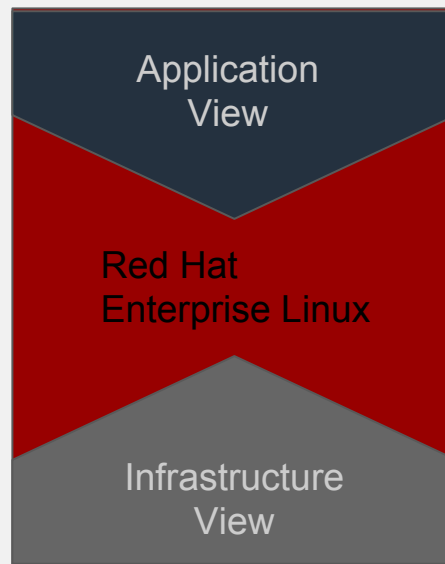


Greybeards fight Balrogs. They hate systemd. They fork distributions.

The Role of the Linux OS

Infrastructure or Application Platform?

- In abstract representations of the modern software stack, the OS is often considered part of the Infrastructure.
- However, an alternative, application-centric view would consider it's primary role to provide a common runtime for applications, abstracting from infrastructure.

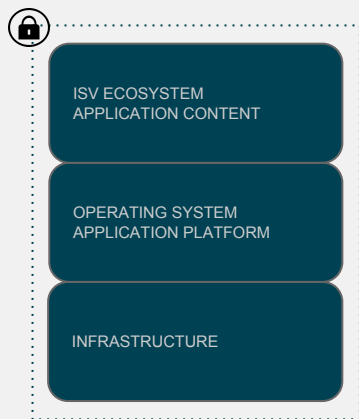


Historic Role of Linux

Breaking the vertical lock-in of Mainframe, Mini-Computers, and UNIX

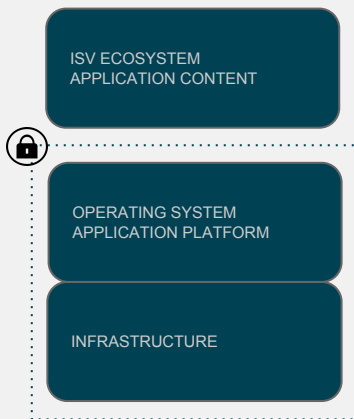
MAINFRAME

Complete vertical integration
Vendor-controlled
HW/OS/Ecosystem.



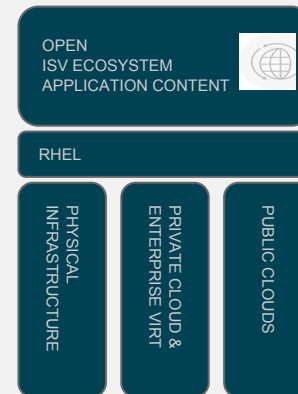
UNIX

Vertical integration of
infrastructure & app platform
Semi-open ecosystem.



RHEL

Completely Open HW and ISV
ecosystem with RHEL as the
neutral enterprise app
platform

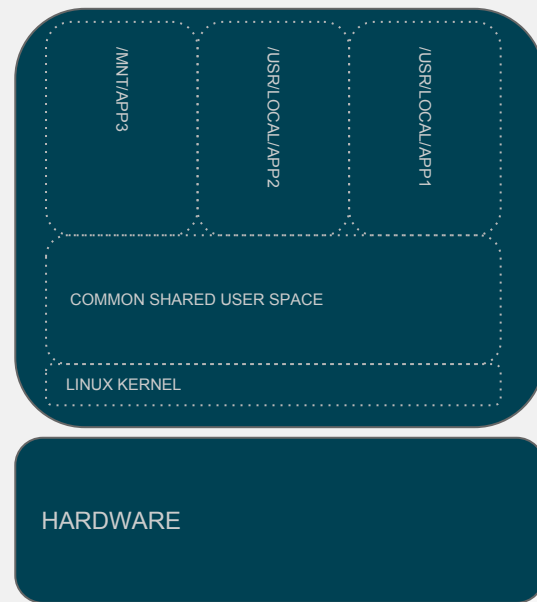


Early Linux Software Stack Management

In the beginning there was /usr/local/ - and stow, and binaries mounted on NFS.

- Servers were special pets. - They were dog-show exhibits.
 - Inherited from Unix host tradition.
- Software often compiled on the production machine.
- High-maintenance.
- Fragile due to dependencies on each host's environment:
 - Application behaviour depends on the state of the individual machine.
 - Not efficient for managing artifacts.
- Late-binding based on source-level API.

Doesn't scale in distributed environments (aka PCs).

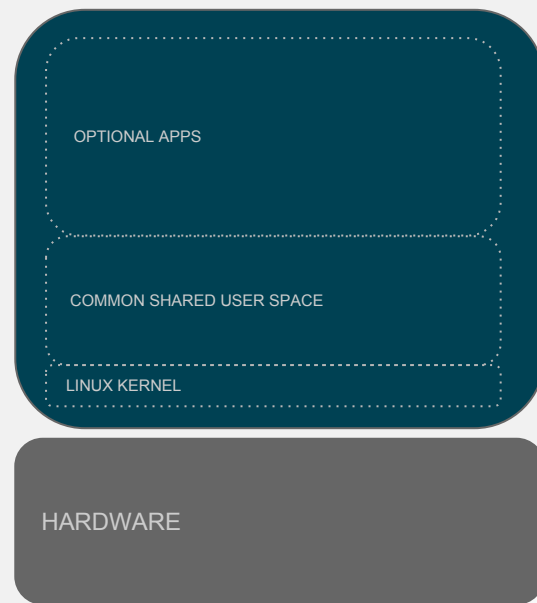


Scalability Through Binary Packaging

Then, There Be RPM and up2date, yum, dpkg, and apt..

- Frozen binary distribution, reproducible builds.
 - Build once, distribute binary across multiple Linux servers.
 - Metadata, signatures.
 - Predictable behavior, dependency management.
 - Management of installed artifacts, updates.
 - Transport for a curated content stream from a trusted source.
- Implicit lock into single instance, single version monolithic userspace.
- Implements a late-binding model for deploying software in Ops based on an ABI contract.

Welcome to Dependency Hell.

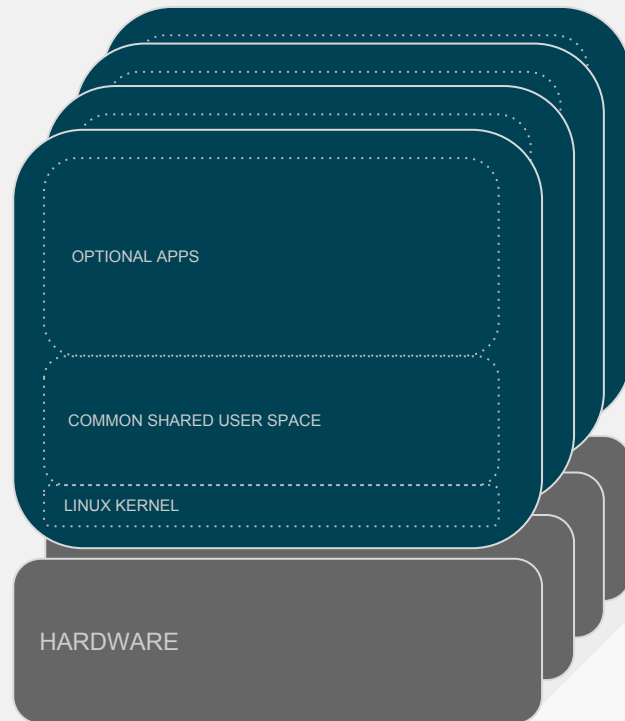


Efficiency Through Central Control

Finally kickstart, satellite, cfengine, and the likes...

- Mass deployment and recipes
- Efficiency through automation. Binary distribution at scale.
- Volatility of late-binding dependency resolution, conflicts & compatibility.
- Automate the stack composition on machines.
- Manage the lifecycle of the software stack.
- Centralize management control.
- Components move across dev/test/ops independently.
- Still in Dependency Hell.

Model still largely used today, sometime with the same components plus newer tools.

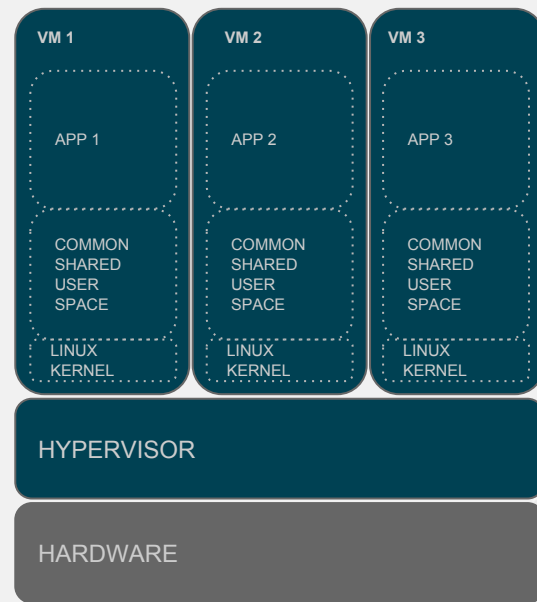


A Whiff Of Freedom

Virtualization, Appliances, The Cloud - Everything is a VM

- Common model: Deploy as pre-built images, operate as pet.
- Predictable initial stack behaviour.
- Existing tools continue to work - it's just virtual HW.
- Multiple instances, multi-tenant.
- Still monolithic inside the VM, still dependency conflicts in VM

Less Dependency Hell - Hello VM Sprawl and inconsistent management.

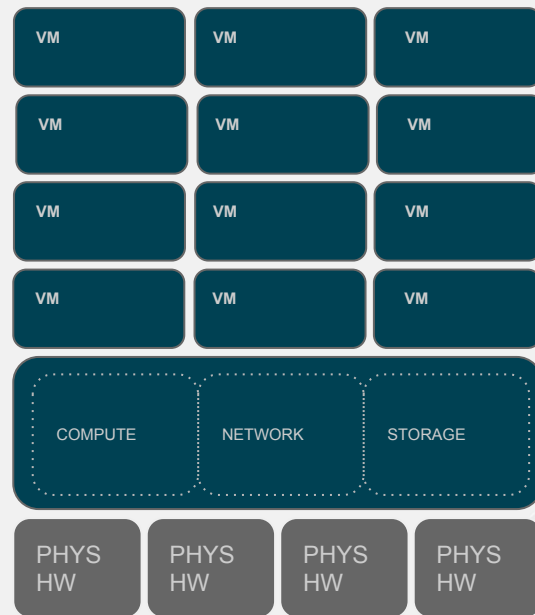


Enterprise Virtualization & Cloud

Infrastructure Elasticity

- Efficient sharing of physical HW due to sharing infrastructure.
- Often Linux inherited one VM per service from Windows.
 - Multi-tier applications consisting out of multiple service.
 - Heavyweight compared to running multiple processes in a single instance.
- Efficient cluster management on VM-level, 'Software Defined' Datacenter
- Potentially the a single artifact to move across DEV/TEST/PROD if integrated into a full image-based lifecycle.
- Did we say Sprawl, but The Cloud takes the ops problems off your hands.

Move towards service aggregation, vertical integration.



Shifting Paradigms

MACRO TRENDS

- “Software is eating the world”
- Business-value driven developers gaining influence over traditional IT
- Open source is the default; driving rapid growth in content volume and stack complexity

PREFERENCES & BEHAVIOR

- Move towards Cloud Native behaviors
- Aggregation of services replaces monolithic systems
- Preference to consume most current versions
- Shift from a broadcast-model to an on-demand model, SaaS

TECHNIQUES & TOOLS

- DevOps enables developers to manage rapid pace of change
- Containers creates application-centric runtimes that allow maximum flexibility with minimal overhead

BIFURCATED DEVELOPMENT ENVIRONMENTS

Customers are increasingly operating in both environments; we must remain relevant in both

Ops-Centric Environments

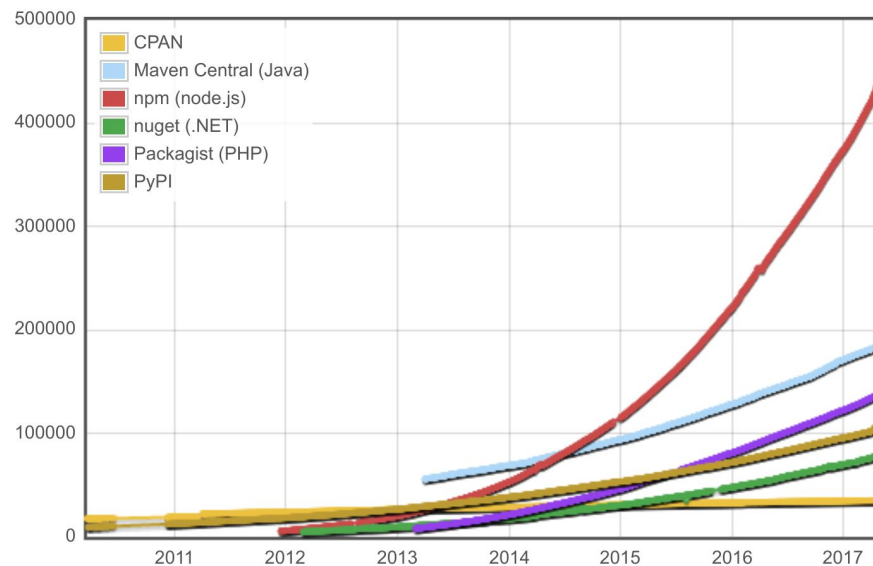
- Single-stream and generational-based distribution
- Stability through control
- Approval-based governance
- Generational, planned appdev
- Optimized for the traditional hardware-centric, monolithic host deployment model
- **Download to install and update in place**

Application Environments

- On-demand distribution
- Stability through validation
- Continuous and process-based governance
- Rapid prototyping and iterative development
- Optimized for cloud-ready applications where infrastructure is ubiquitous
- **Download to build**

Software Stack Complexity Keeps Growing

Module Counts



Traditional Distro Challenges in the App-Space

Diminishing Returns at Growing Complexity

Traditional binary software distribution great for foundational platform components... But:

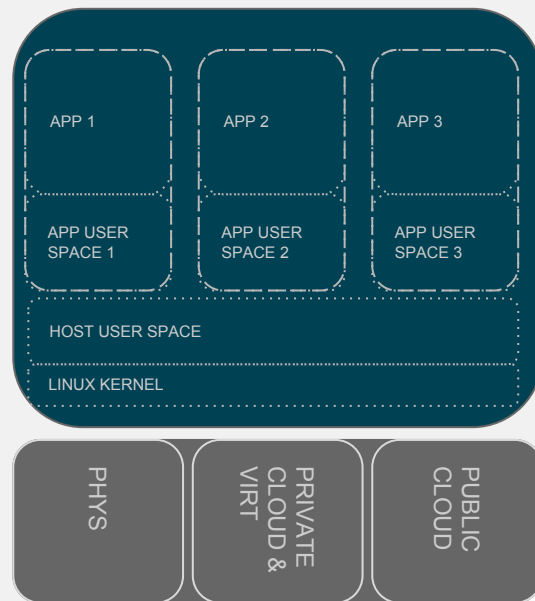
- Modern software stacks have become too complex to be mapped into a common, monolithic namespace.
- As a developer, I have to go to native packaging (e.g. npm) anyways because the distribution does only provide a small part of what I need to build my application.
- Slow delivering new versions to app developers.
- The higher in the stack, the bigger the issue.
- Re-packaging, frozen binary distribution offers little value for the App developer.
- Upstream binary/bytecode formats sufficient, they compile their software anyways, lock-in for hybrid environments.
- Testing is more valid if done with the actual application, using it.

Liberation: Containers

Expanding use of containers, from VServer over LXC to OCI

- Separate the application runtimes from system runtime.
 - Like chroot but with an epstein drive.
- Multi-instance, multi-version environment with possible multi-tenancy: each service has it's own binary runtime.
- Light-weight - at the end, it's just linux processes separated by kernel features: CGroups, Namespaces, SELinux

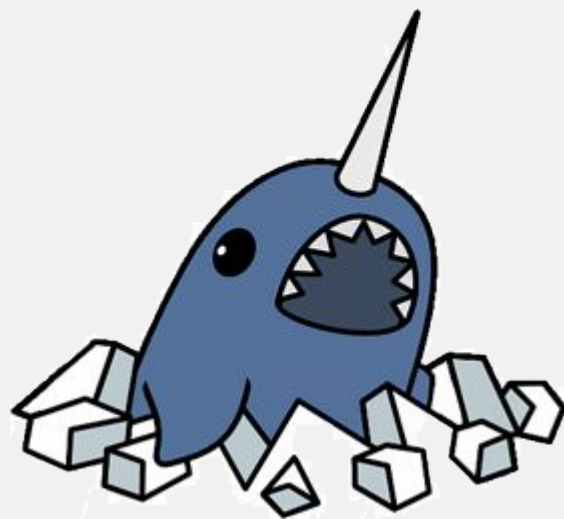
Good bye Dependency Hell



Enter: The Container Revolution

OCI Containers provide the package format for Application-Centric IT

- Aggregate packaging deployed into containers.
 - Initiated by the project previously known as ‘Docker’.
 - Combine existing Linux Container technology with Tar + overlays -> Unicorns
- Frozen binary distribution, reproducible builds.
 - Build once, distribute binary across multiple Linux servers.
 - Metadata, signatures.
 - Management of installed artifacts, updates.
 - Transport for a curated content from a trusted source.
- Fully predictable stack behaviour, life cycle, lightweight.
- Implements an early-binding model for deploying applications packaged by a developer.



The best of both worlds.

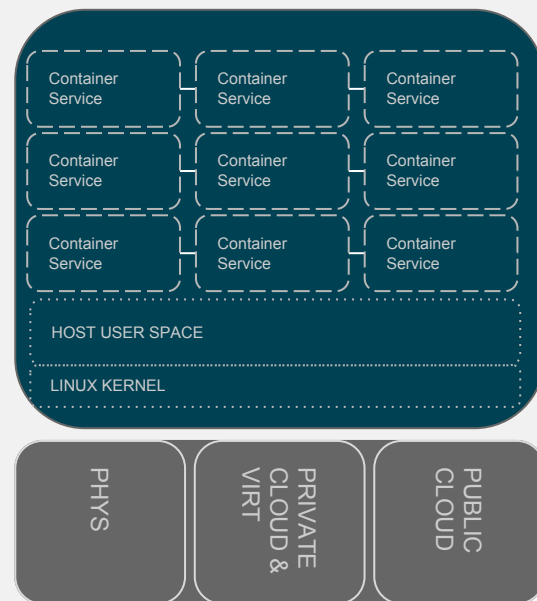
Source: http://www.clipartpanda.com/clipart_images/narwhal-facts-by-whispered-4184726

Multi Container Apps

In reality, most applications consist of multiple containerized services.

- Ideal container is only a single binary.
- Applications are aggregated from multiple containerized services.
- Ideal for cloud native applications.
- From multi-tier applications to micro services.
- Static linking or dynamic orchestration.

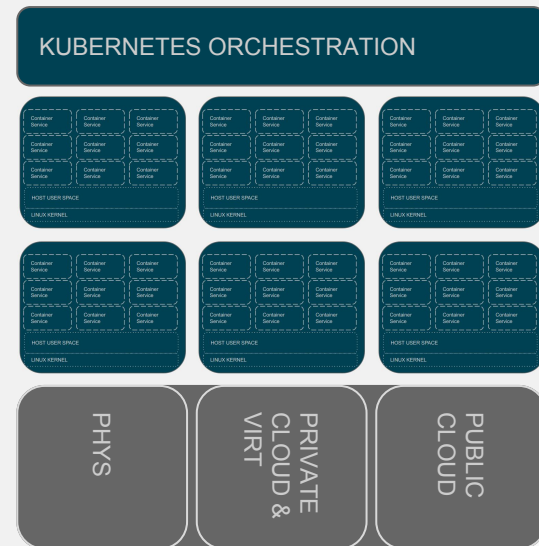
Great to solve dependency hell, but how to make sure my frontend knows which database to talk to?



The Cluster Is The Computer

By default , everything is a cluster

- Kubernetes manages containerized services across a cluster of Linux nodes.
- Application definition model, describing the relationship between services in abstraction from the individual node.
 - Abstraction from the underlying infrastructure: compute, network, storage.
 - Same application definition remains valid across changing infrastructure.
- Whole stack artefacts move across dev/test/ops unmodified.
- Scale-out capabilities and HA are commoditized into the standard orchestration.
- Often built around immutable infrastructure models.



Container Use Cases

Three Major Types of Container Use Cases

FULLY ORCHESTRATED MULTI-CONTAINER APPLICATION

- Container build service
- Multi-container
- Kubernetes
- OpenShift
- Cloudforms mgmt engine
- Ansible used against higher-level APIs

LOOSELY ORCHESTRATED CONTAINER APPLICATION

- Container build service
- One or multi-container
- No Kubernetes
- Static relationships configured on the host
- Atomic tool
- Ansible used against lower-level APIs

PET CONTAINER

- Environment bootstrapped by container
- Run YUM, NPM, etc inside container instance (NOT pre-built in build service)
- Ansible inside container (similar to VM use case)

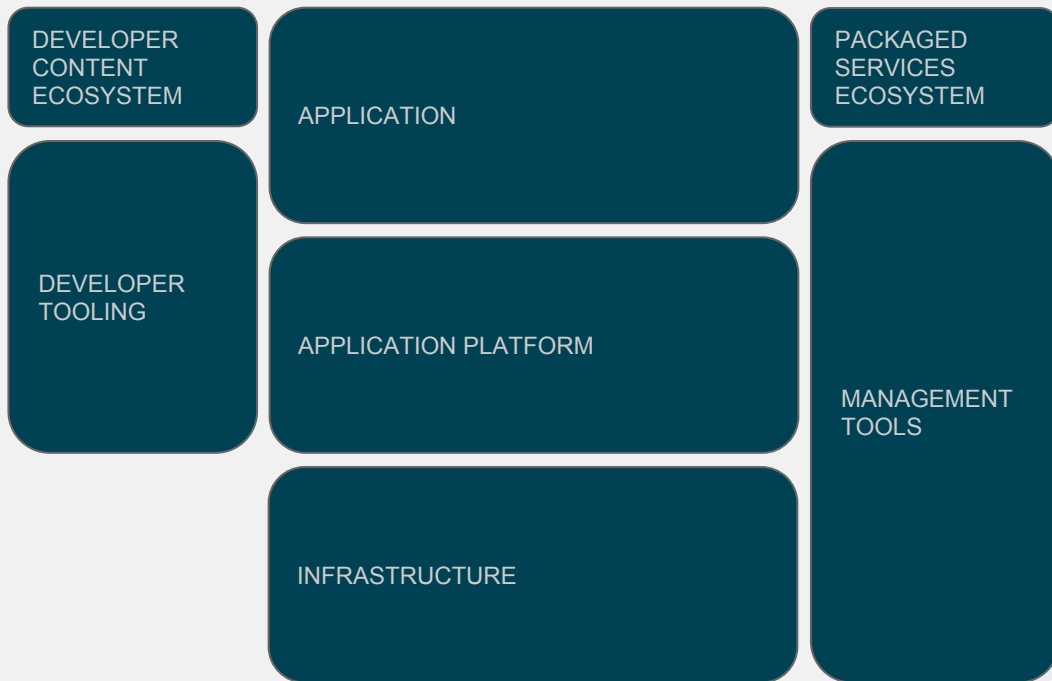
The New App-Centric Platform

Built around the Application Platform and Application Content.

Enabled by developer tools and hybrid cloud management.

Fed by packaged service and developer content ecosystems.

Supported by infrastructure integration.



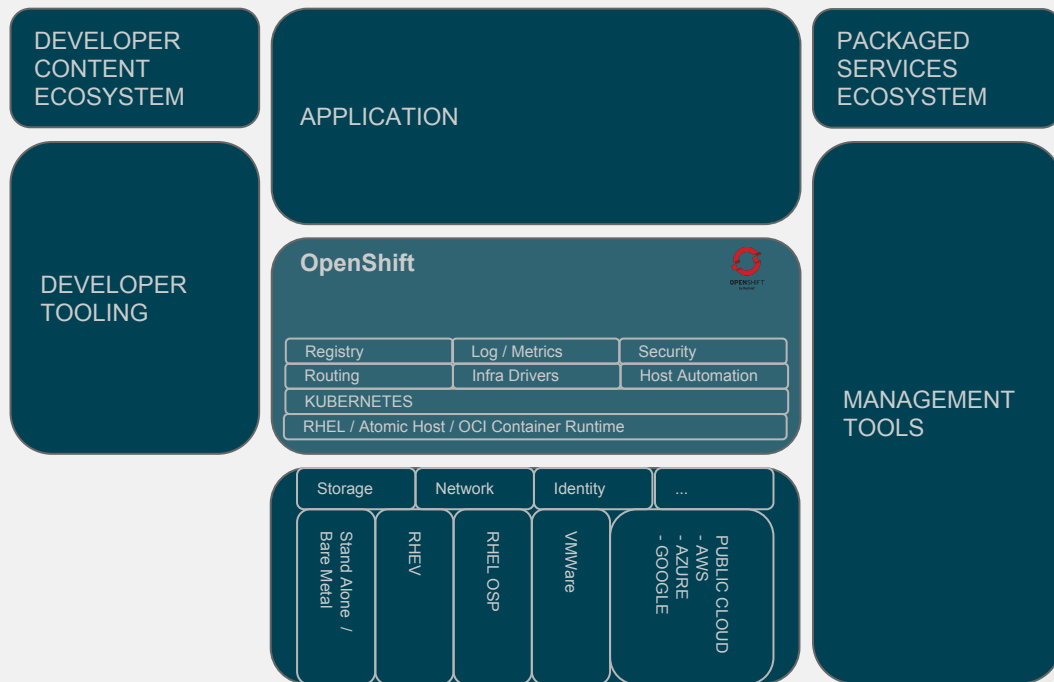
Abstraction Across Hybrid Cloud

OpenShift provides a scale-out, “the cluster is the computer” platform to deploy fully-orchestrated multi-container applications.

Built on RHEL Atomic Host in the immutable infrastructure paradigm, OCI containers, etcd, kubernetes, systemd.

Application is defined in abstraction from Infrastructure provider details, works across different cloud providers, integrates with infrastructure services.

Fully Open Source, Standards-based, pluggable.



Fully Portable Application

Applications consist of multiple containers, built as OCI Container images. Kubernetes defines application entity.

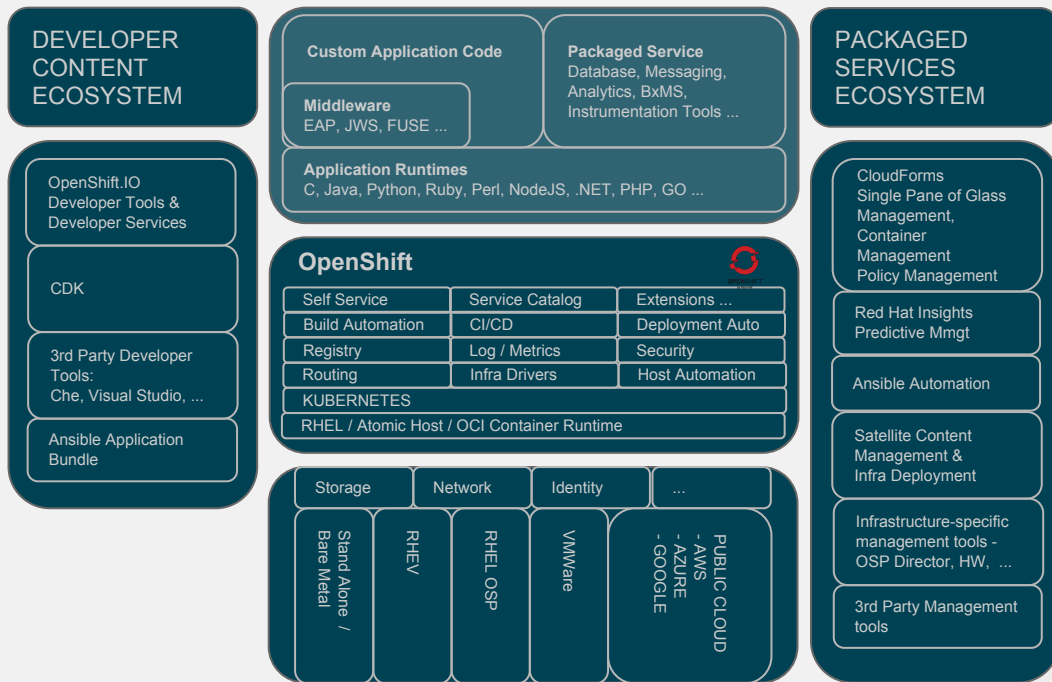
Ansible Application Bundels and the OpenServiceBroker API provide a transport model for full-application portability across the ecosystem.

Red Hat provides RHEL base images to build layered apps on. Today RHEL 6 and 7 are supported in parallel.

Pre-packaged services can be easily aggregated without need to build images or application templates. Middleware is integrated experience provided through

xPaaS.

#redhat #rhsummit



Trusted Ecosystems

OPEN ECOSYSTEMS

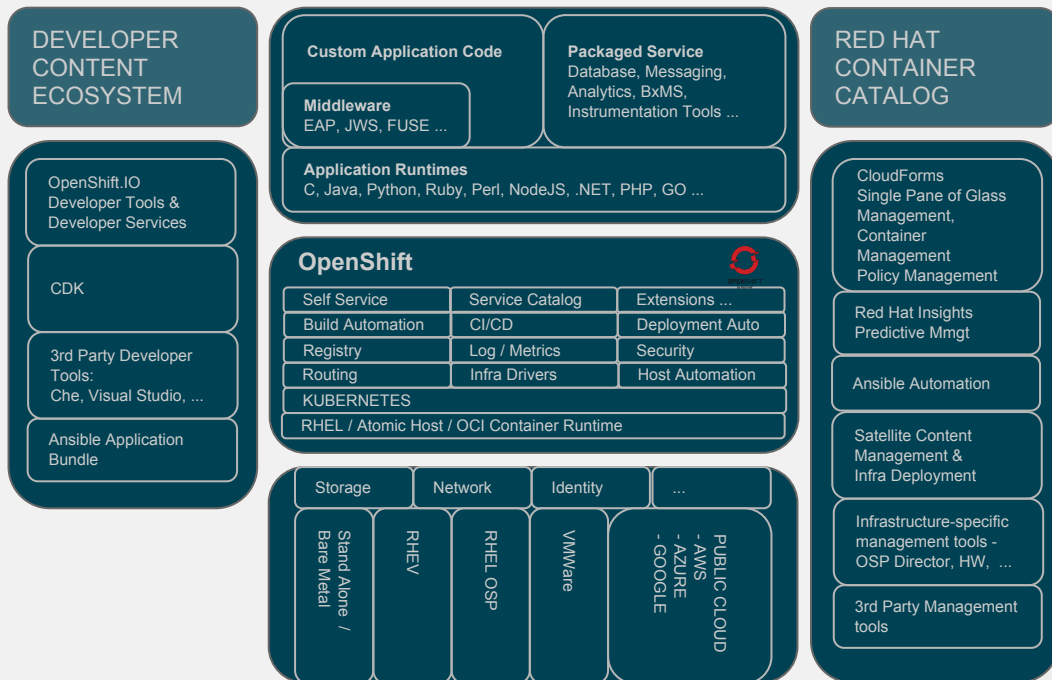
Content available from Red Hat, certified partners, community. Selection controlled by customer. Red Hat and partners provide guidance on content security: Red Hat Insights, Blackduck, etc.

COMPONENT-LEVEL

Downloaded to build
E.g. Java Library, RPM, zip, pip, npm module, ...
“As a developer, I want to develop my application using an existing UI framework library.”

PACKAGED SERVICES

Downloaded to install
OCI Container image
E.g. Database, messaging service, platform extensions, host drivers.
“As a sysadmin or developer, I want to aggregate a pre-packaged database service into my application.”



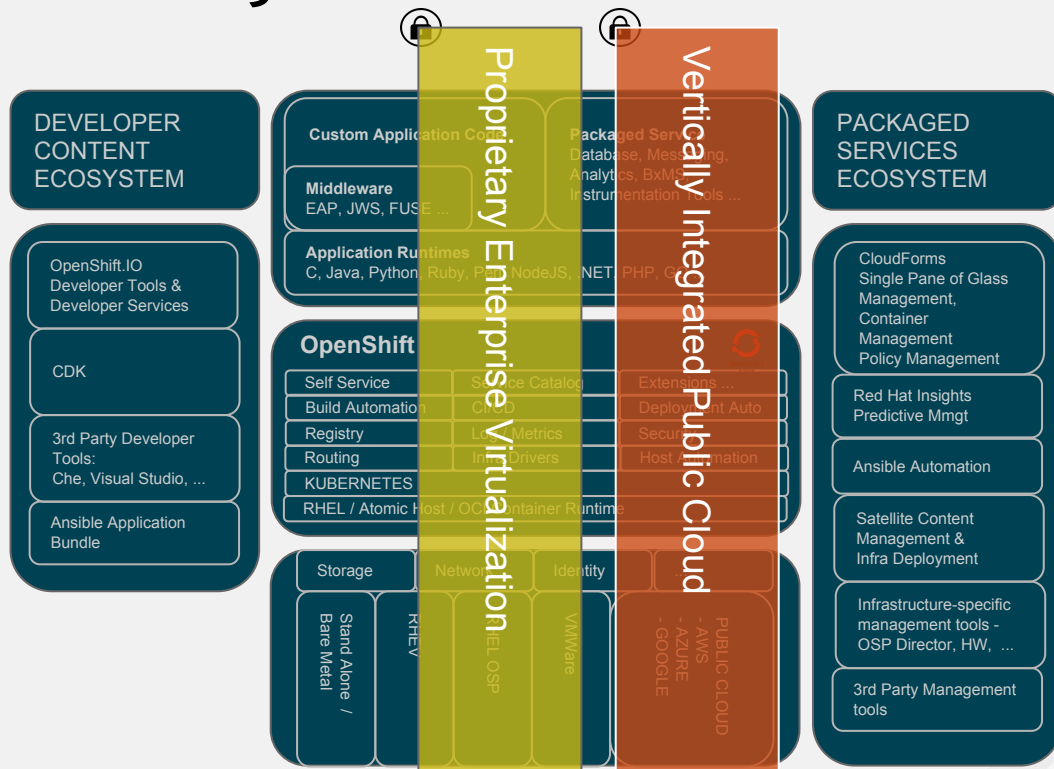
Breaking the Vertical Integration


Public cloud & proprietary private cloud are driving vertical integration and lock-in with pseudo-standards.

- Just like UNIX.

Red Hat is in the unique position to again become the neutral runtime for an open ecosystem on hybrid infrastructure, disrupting the vertical integration of proprietary vendors.

- Just like RHEL.





The future of the Linux OS is a scale-out cluster-as-computer platform for fully orchestrated multi-container apps, providing an abstraction layer across underlying infrastructure, and breaking the vertical integration of proprietary cloud.

RED HAT
SUMMIT

THANK YOU



plus.google.com/+RedHat



facebook.com/redhatinc



linkedin.com/company/red-hat



twitter.com/RedHatNews



youtube.com/user/RedHatVideos

The logo consists of a red speech bubble shape pointing downwards, containing the text "RED HAT" in a smaller font above "SUMMIT" in a larger, bold font.

RED HAT
SUMMIT

LEARN. NETWORK.
EXPERIENCE
OPEN SOURCE.