A Greybeard's Worst Nightmare

How Kubernetes and Containers are re-defining the Linux OS

Daniel Riek
August 2018
Introduction

- Name: Daniel Riek  Twitter: Ilunved
- Using Linux since 1994
- Co-founded Linux start-up ID-Pro in Europe in 1997
- Worked at Alcove, a french GNU/Linux company 2001-2003
- Red Hat, EMEA Sales Engineering 2003-2005
- Red Hat, ran RHEL Product Management 2005-2011
- CTO at trading startup Vincarex 2011-2012
- Product Management at back-up company Acronis 2012-2013
- Red Hat, Office of the CTO, Artificial Intelligence since 2017
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 its primary role to provide a common runtime for applications, abstracting from infrastructure.
Historic Role of GNU/Linux

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

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

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

**Linux - e.g. RHEL**
Completely Open HW and ISV ecosystem with the Linux OS as the neutral enterprise app platform
Early GNU/Linux 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.
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 like Ansible.
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.
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
- Shift from a broadcast-model to an on-demand model, SaaS

**PREFERENCES & BEHAVIOR**
- Aggregation of services replaces monolithic systems
- Preference to consume most current versions
- Open source is the default; driving rapid growth in content volume and stack complexity

**TECHNIQUES & TOOLS**
- Move towards Cloud Native behaviors
- DevOps enables developers to manage rapid pace of change
- Containers creates application-centric runtimes that allow maximum flexibility with minimal overhead
Growing Software Stack Complexity

Source: www.modulecounts.com
DevConf.US 2018
Traditional Distro vs App-Centricity

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 its 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’. Now implemented by native stack with CRI-O, Podman, and Buildah.
  - 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. CI/CD friendly.


The best of both worlds.

DevConf.US 2018
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. Hybrid model for existing apps.
- 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 artifacts move across dev/test/ops unmodified.
- Scale-out capabilities and HA are commoditized into the standard orchestration.
- Often built around immutable infrastructure models.
- OpenShift is Red Hat’s distribution of Kubernetes.
The New App-Centric Platform

Application View

(Red Hat Enterprise) Linux

Infrastructure View

Application

Developer Content Ecosystem

Developer Tooling

Application

(RED HAT OPENSIFT) Application Platform

Packaged Services Ecosystem

Management Tools

Phys

Private Cloud & Virt

Public Cloud
Cloud - Changing how do Software

- OPEX vs CAPEX
- Elasticity & Self Service
- Data Aggregation
- Integration
- Security
- Operational excellence made accessible
- Everything is a service
- Moving towards hybrid cloud with on-prem expansion
- Service abstraction & time to value
  - Pre-defined services, SaaS
  - Focus on the core app value
  - Function as a Service
The Cost of Cloud

The Cost:

- Lock-in
- Black-box-services
- Data Gravity
- Life cycle dependency
- High OPEX when scaled
- Reproducibility?
- Open Source?
- ISVs?

Black-box services on leased HW? - Mainframe anyone?
An Open Alternative?

The biggest challenges to an open alternative to the proprietary public cloud are:

- Service abstraction and time to value
- Sheer number of services and their integration
- Application portability
- Operational excellence
Abstraction Across Hybrid Cloud

Kubernetes (in the Red Hat case in the form of OpenShift) provides a scale-out, “the cluster is the computer” platform to deploy fully-orchestrated multi-container applications.

Built on the Linux OS (here RHEL and Red Hat CoreOS) 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.

Userspace portability across different versions of the Host. E.g. Red Hat provides RHEL images to build apps on. Today support for RHEL 6 and 7.

Pre-packaged services can be easily aggregated without need to build images or application templates.
Trusted Ecosystem

OPEN ECOSYSTEMS
Content available from vendors such as Red Hat, partners offering specific services, community. Selection controlled by customer.

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 images and metadata wrapped as APBs from Container Registries in early-binding model..
Examples: 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.

Kubernetes 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 Linux.
Standardized Operational Model

With concepts such as Operators and standardization of metrics integration across the whole stack, OpenShift provides a standardized operational model for applications across a hybrid environment.
Operational Excellence

The standardized operational model allows easier aggregation of knowledge and shared learning:

‘Herd immunity’ through aggregation of operational experience and predictive guidance:
Red Hat Insights
AIOps: Towards Self Driving Clusters

Integration of AI in the standardized operational model offers the promise of intelligent automation.
Conclusions

- Linux’ historic role was to break vertical integration and provide a common platform for an open ecosystem.
- The cloud has changed IT, driving efficiency through service abstraction and operational excellence.
- The downside of cloud is vertical integration and lock-in.
- Kubernetes offers the opportunity to create an open alternative platform.
- It can enable an ecosystem of services at the same level of service abstraction.
- Predictive guidance and AIOps can make it competitive to the cloud provider’s operational expertise.
Thank you!