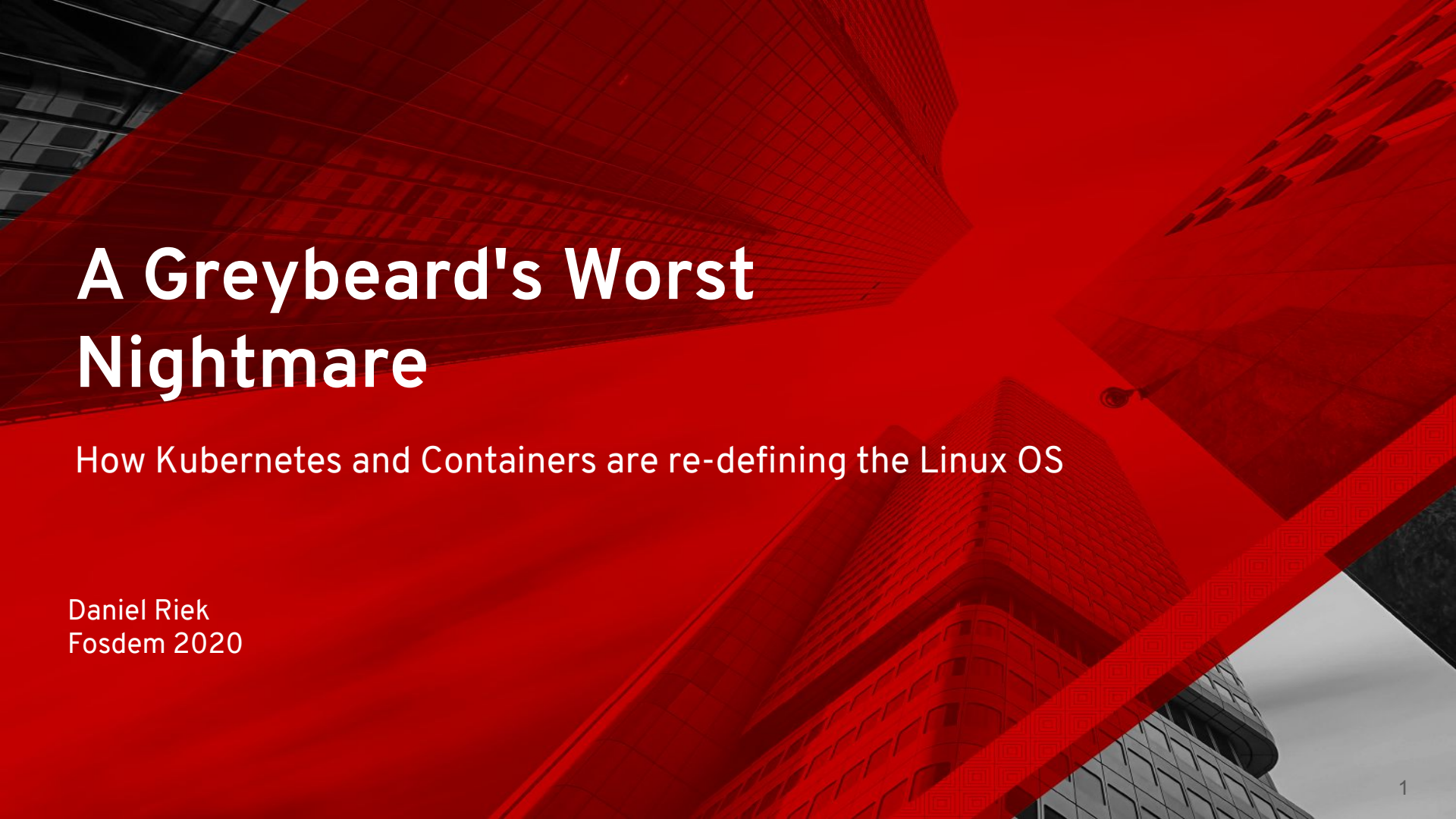


A Greybeard's Worst Nightmare

How Kubernetes and Containers are re-defining the Linux OS

Daniel Riek
Fosdem 2020

Introduction

- Name: Daniel Riek Twitter: llunved
- Using GNU/Linux since 1994
- Co-founded Free Software 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 Vincorex 2011-2012
- Product Management at back-up company Acronis 2012-2013
- Red Hat, managing cross-product integration, container initiative 2013-2017
- Red Hat, Office of the CTO, Artificial Intelligence since 2017
    - Working on FOSS AI incl. projects such as https://opendatahub.io and https://github.com/thoth-station

DISCLAIMER

So yes, I work at Red Hat, which is a subsidiary of IBM.

Red Hat is a Free and Open Source Cloud Software Company.

However, while I may use the Red Hat stack as an example, nothing I say here can be misconstrued into an official position of any of my former, current, or future employers. It’s just me.

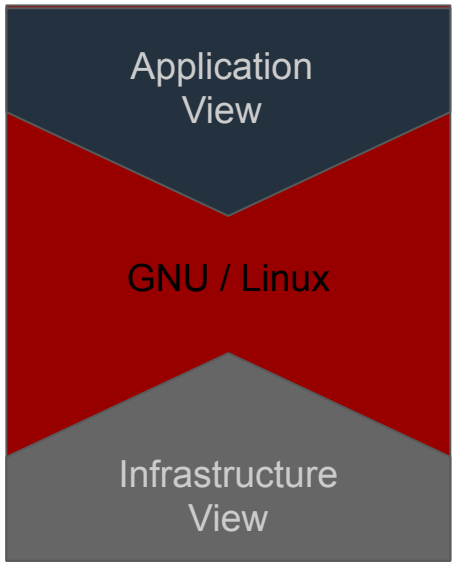
Greybeard

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

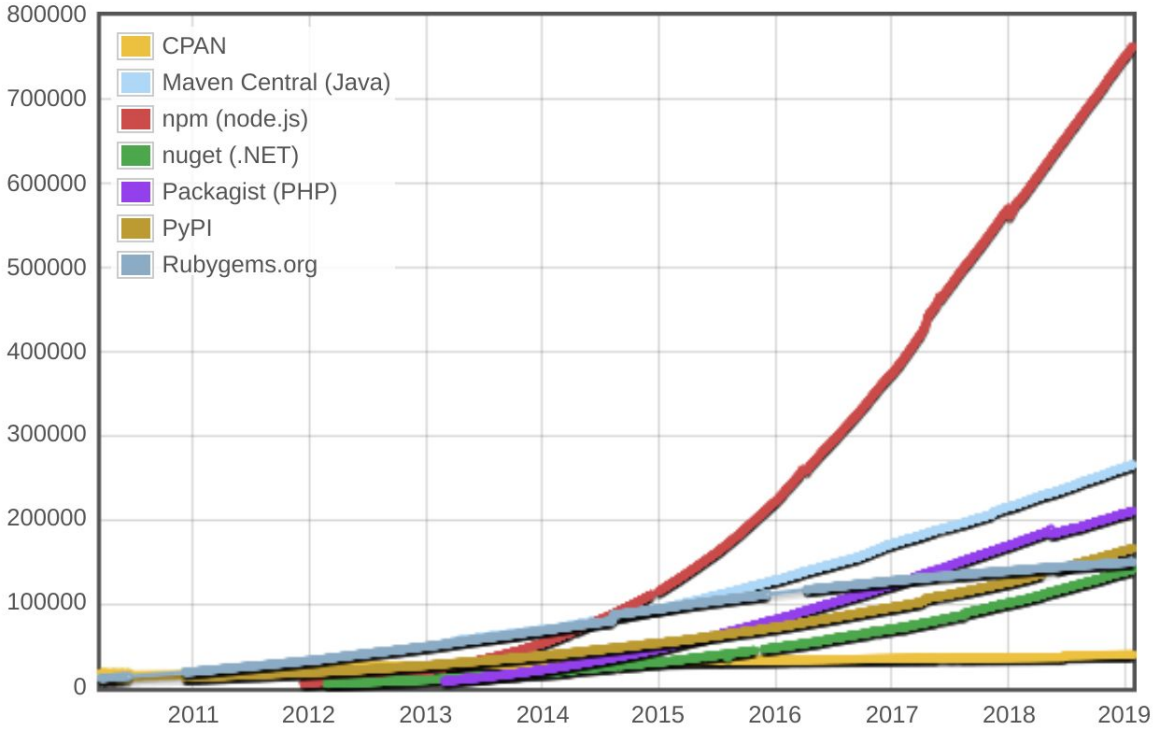
The Role of the Linux OS

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.



Meanwhile: Growing Software Stack Complexity



Source: www.modulecounts.com

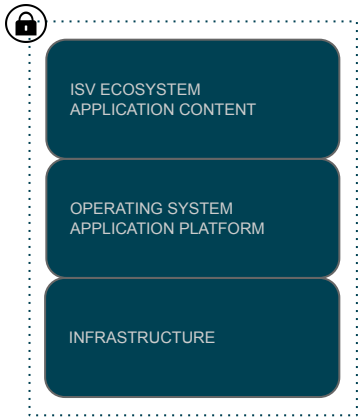
Historic Role of GNU/Linux

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

MAINFRAME

Complete vertical integration

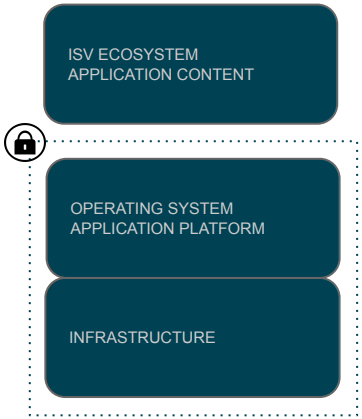
Vendor-controlled HW/OS/Ecosystem.



UNIX

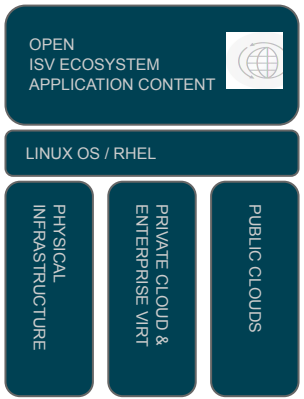
Vertical integration of infrastructure & app platform

Semi-open ecosystem.



GNU/Linux - e.g. RHEL

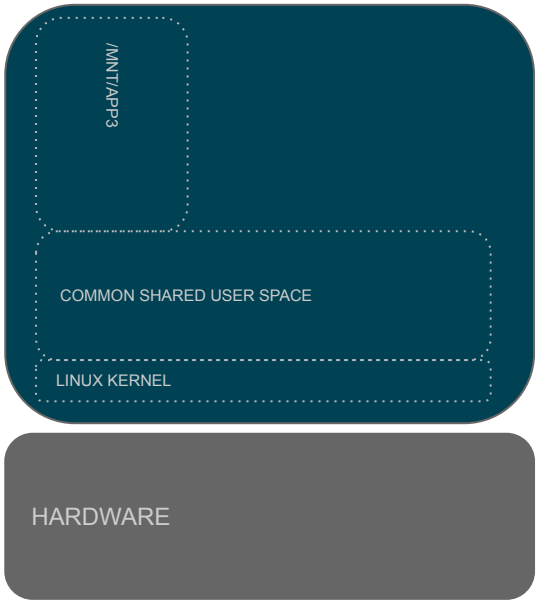
Completely Open HW and ISV ecosystem with the GNU/Linux OS as the neutral enterprise app platform



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.



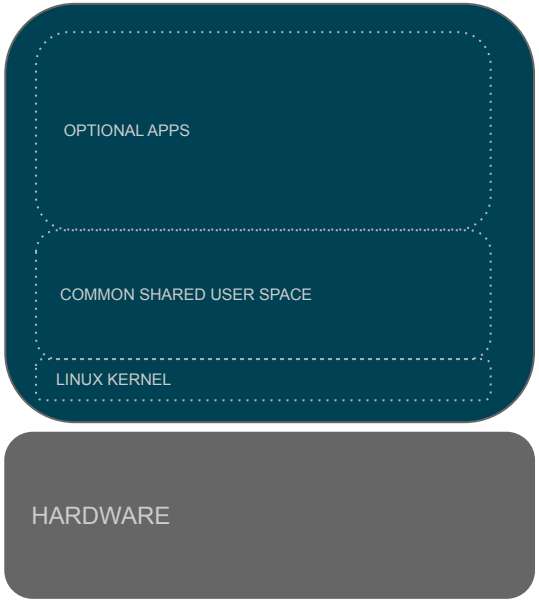
Doesn't scale in distributed environments (aka PCs).

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.

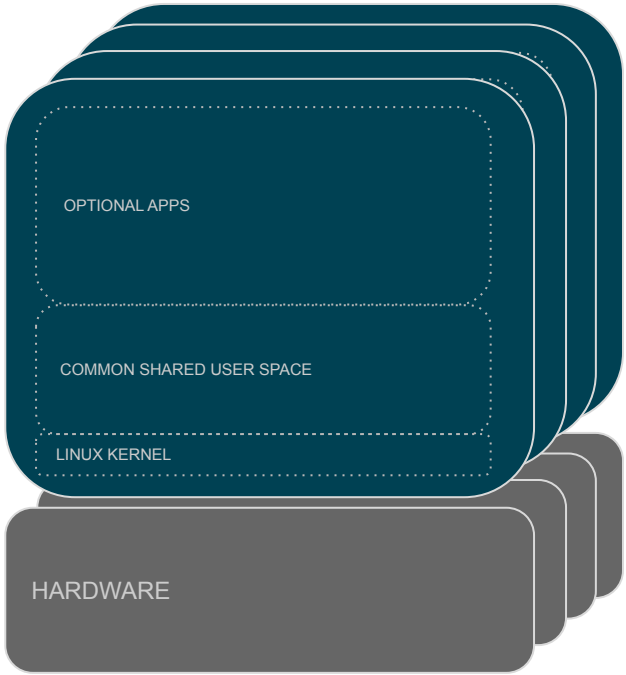


Efficiency Through Central Control

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 like Ansible.

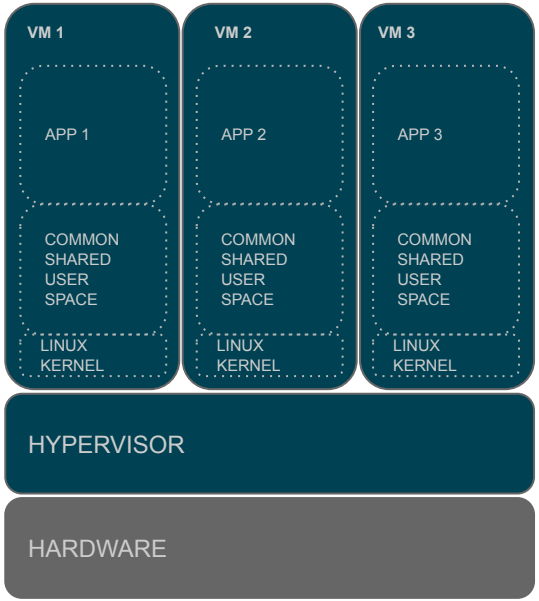


A Whiff Of Freedom

Virtualization, Appliances - Everything is a VM

- Common model: Deploy as pre-built images, operate as pet
- Predictable initial stack behaviour
- Abstraction from underlying HW
- 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.

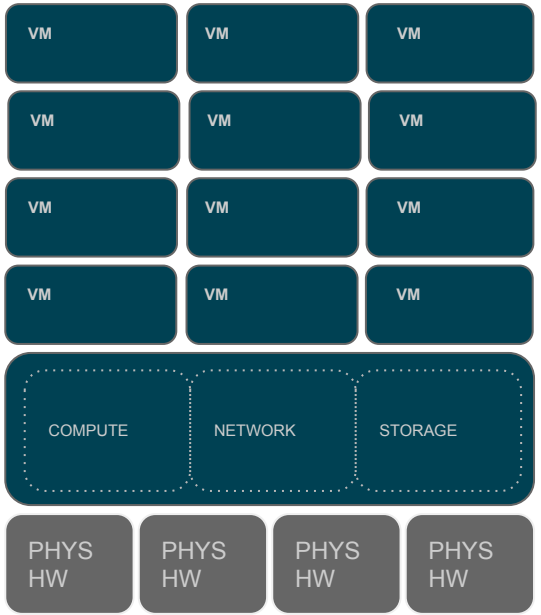


Enterprise Virtualization

Infrastructure Abstraction & Density

- Efficient sharing of physical HW due to sharing infrastructure.
- 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.
- In theory clean delegation. - In practice: shared root access and a lot of virtual pets.

Liberates your app from the HW lifecycle. Predominant operational paradigm for data centers in the earlier 2010s.



Infrastructure as a Service Cloud

Elastic Infrastructure

- Compute / Storage / Networking on demand
- Opex instead of CAPEX
- Elastically scale up and down as you need it
- Efficiency through scale
- Progressive reduction in cost passed through to customers

Shifting Paradigms

- "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

- 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

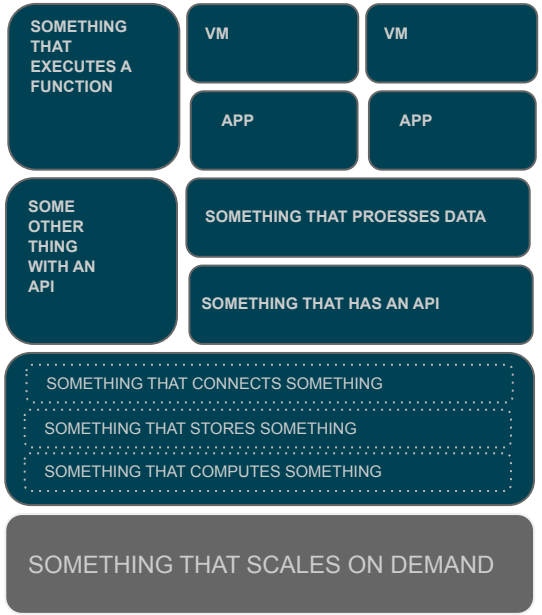
- Move towards Cloud Native behaviors
- DevOps enables developers to manage rapid pace of change
- Automation, automation, automation....

The (Modern) Cloud

Operational paradigm that maximizes time-to-value:

- Elasticity
- Developer Velocity through service abstraction, integration, and availability
- Encapsulated Operational Excellence

- Dominated by proprieatry public cloud offerings
- ‘GNU / Linux Distribution as a Service’ - Without the contributions back.
- ‘Strip-mining’ FOSS and SW innovation in general.
- Move towards service aggregation, vertical integration.



Cloud Changed How People See Software

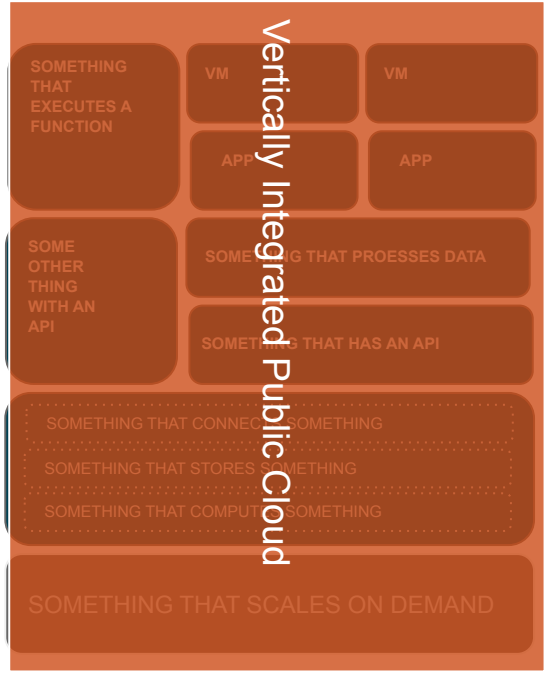
- Centralization of Operational Excellence

- 1990 / 2000s:
  - Access to enterprise HW and Software is exclusive.
- 2005 - 2015:
  - Free Software democratized access.
  - Commercial offerings (e.g. Red Hat) enable Enterprise use.
  - You can get integration, stability, maintenance… But you have to figure out how to deploy and operate.
- 2020:
  - The cloud operates your infrastructure and services.
  - You focus on the components that differentiate your business.

Predominante operational paradigm for IT in the late 2010s

The Cost of Cloud

- Dominated by proprietary public cloud offerings
- Lock-in with black-box-services
- Data Gravity
- Growing life cycle dependency
- High OPEX when scaled
- Reproducibility?
- ‘GNU / Linux Distribution as a Service’ - Without the contributions back.
- ‘Strip-mining’ FOSS and SW innovation in general.
- Move towards service aggregation, vertical integration.

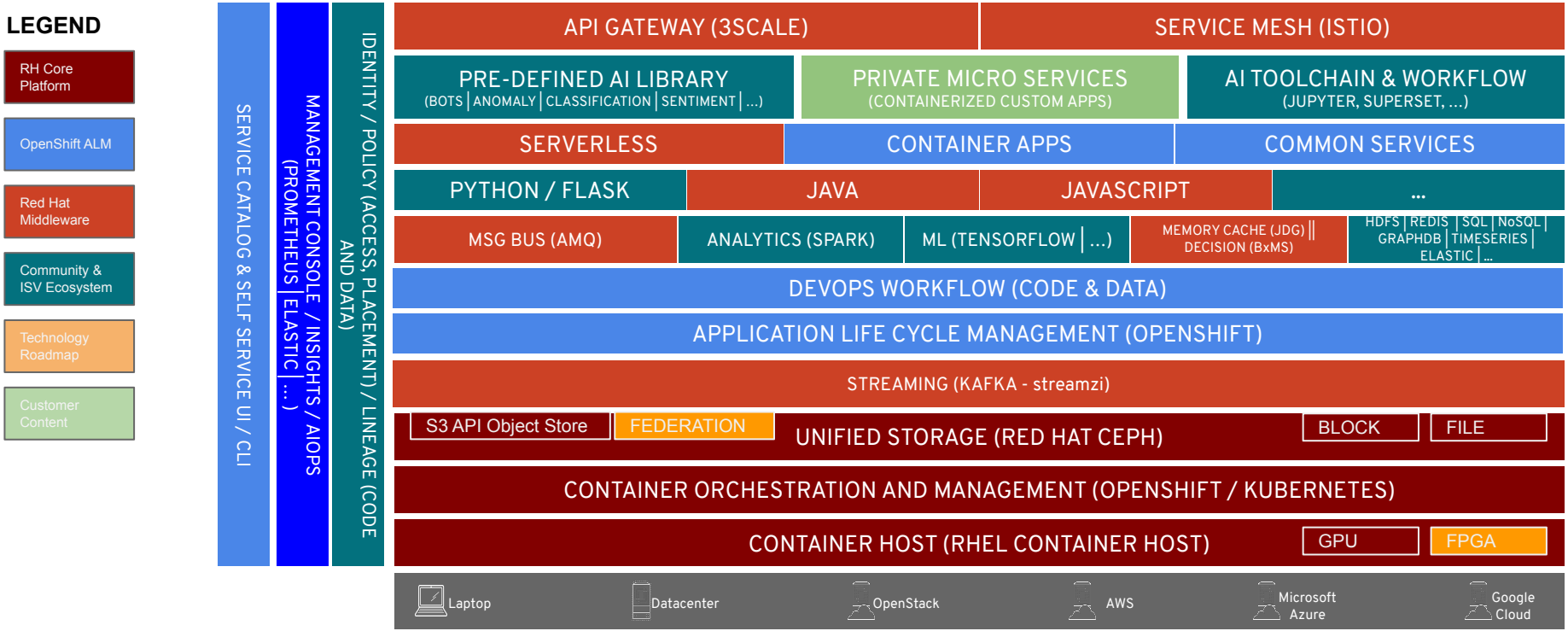


An Open Alternative?

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

AI STACK COMPLEXITY - RH-INTERNAL EXAMPLE



Traditional Distro vs App-Centricity

Diminishing Returns at Growing Complexity

Traditional binary software distribution great for foundational platform components…

But:

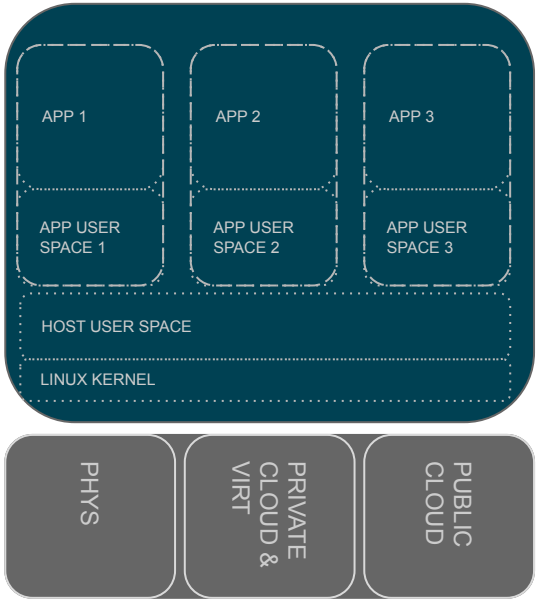
- Modern software stacks have become too complex to be mapped into a common, monolithic namespace.
- As a developer, I have to go to native packaging (e.g. npm) anyways because the distribution does only provide a small part of what I need to build my application.
- Slow delivering new versions to app 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.
- Updating of services in production at the component level is too volatile.

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



If you are running Fedora, try:

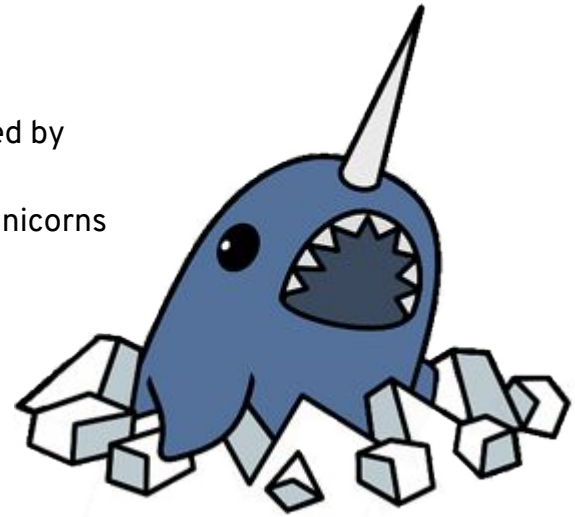
sudo dnf -y install toolbox

toolbox enter

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.



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

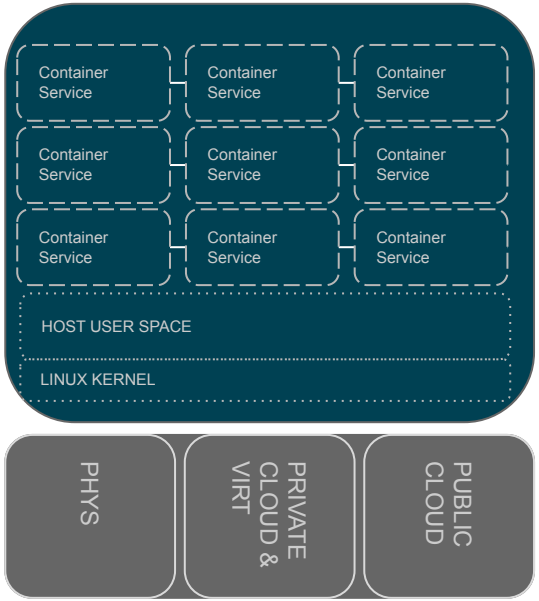
The best of both worlds. Encapsulates tack complexity and provides a relatively stable interface.

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.

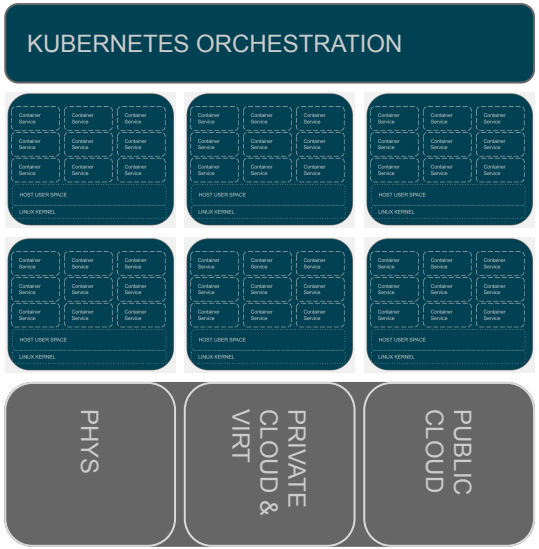
Great to solve dependency hell, but how to make sure my frontend knows which database to talk to?



Kubernetes: 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.



Operators: Standardized Operational Model

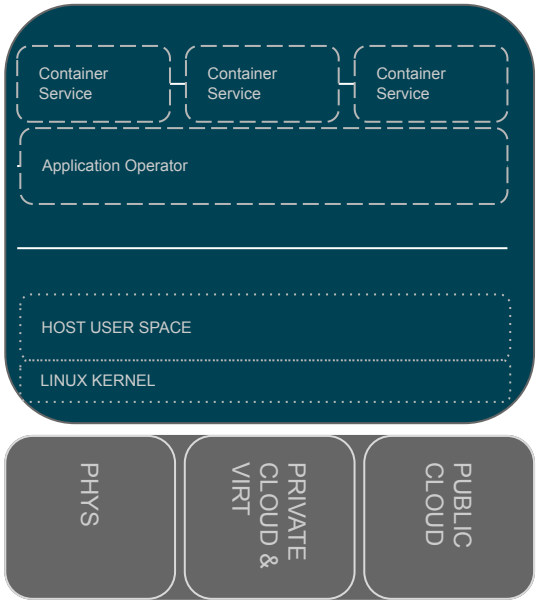
PROBLEM: How to manage the operation of complex, multi-container apps?

SOLUTION: Kubernetes Operators

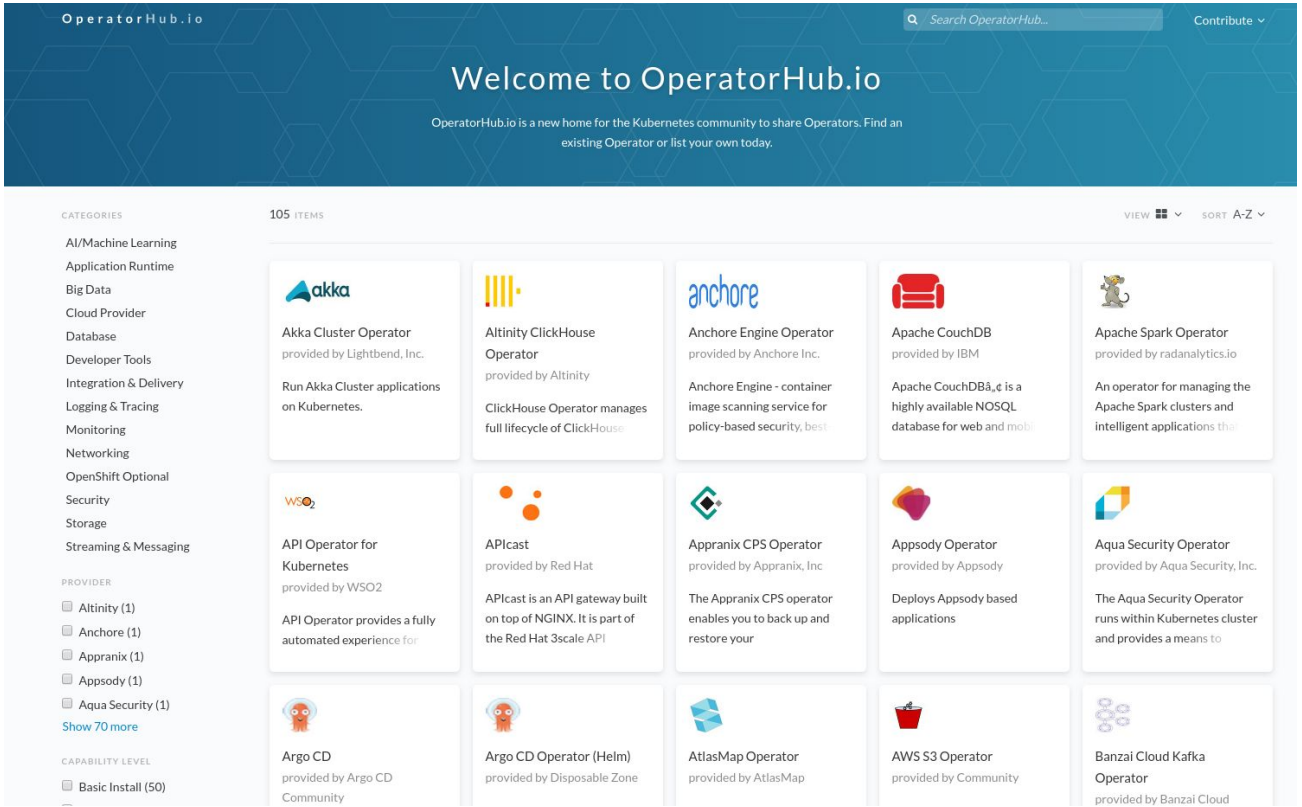
- Active component implementing the operational logic for a specific application.
- Capability Levels
  1) Basic Install
  2) Seamless Upgrades
  3) Full Lifecycle
  4) Deep Insights
  5) Auto Pilot

Federate the encapsulation of operational Excellence.

Future: AI Ops

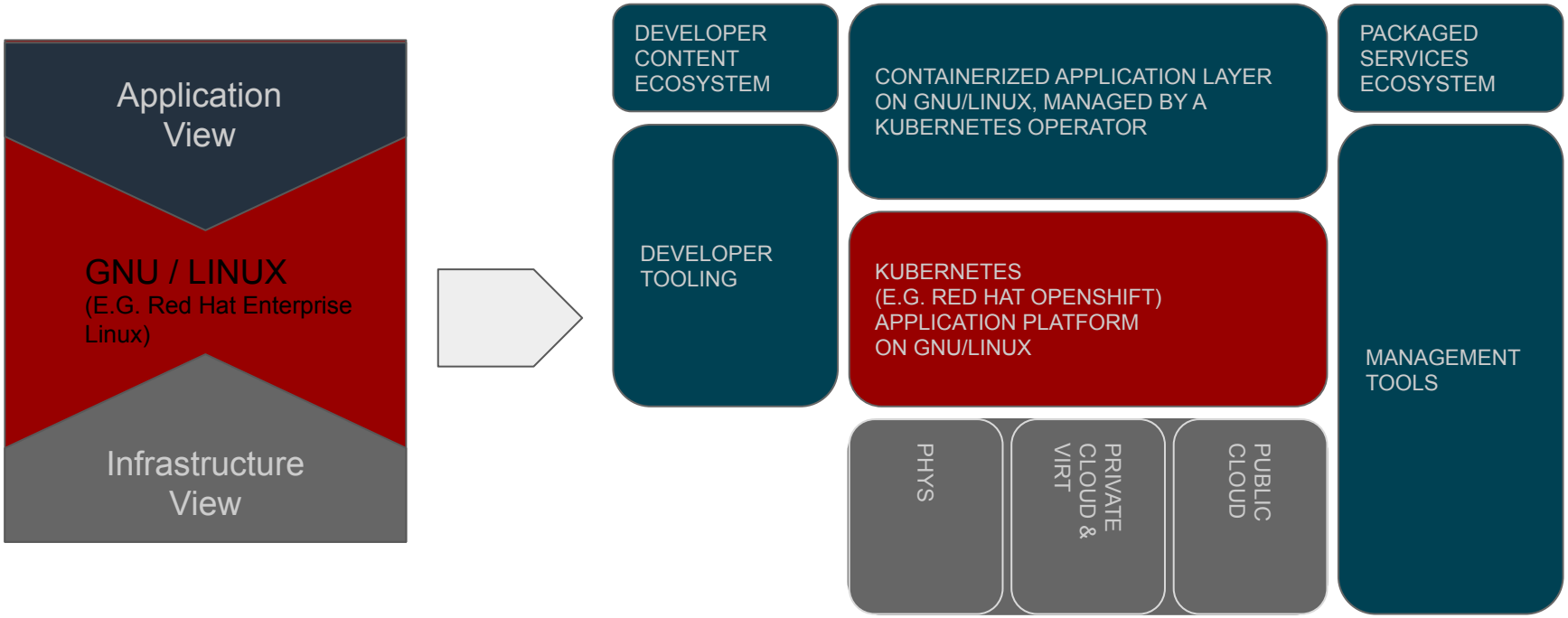


Operators: Standardized Operational Model



https://operatorhub.io/

The New App-Centric Platform



Conclusions

- GNU / Linux' historic role was to break vertical integration and provide a common platform for an open ecosystem.
- The cloud has changed IT, driving efficiency across elasticity, developer velocity and encapsulated operational excellence.
- The downside of cloud is concentration, vertical integration and lock-in.
- Containers and Kubernetes offer the opportunity to create an open alternative platform.
- Kubernetes and operators provide the base for a standardized operational model that can competitive to the cloud providers' operational expertise.
- It can enable a heterogeneous ecosystem of services at the same level of service abstraction as the public Clouds.
- FOSS needs to go beyond software access and democratize operations.

Talk Recommendations

- Thoth - a recommendation engine for Python applications
  Fridolín Pokorný
  - Room: UB2.252A (Lameere)
  - Time: Saturday - 18:00
  - https://fosdem.org/2020/schedule/event/python2020_thot/

- Do Linux Distributions Still Matter with Containers?
  Scott McCarty
  - Room: K.3.201
  - Time: Sunday - 09:00
  - https://fosdem.org/2020/schedule/event/dldsmwc/


http://www.etsy.com/shop/sharpwriter
VEUSER 2012


Thank you!