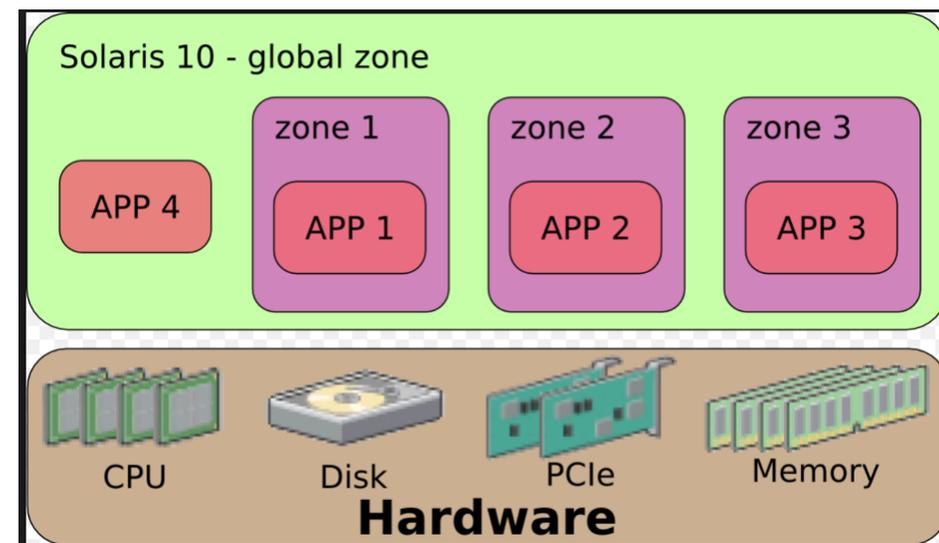
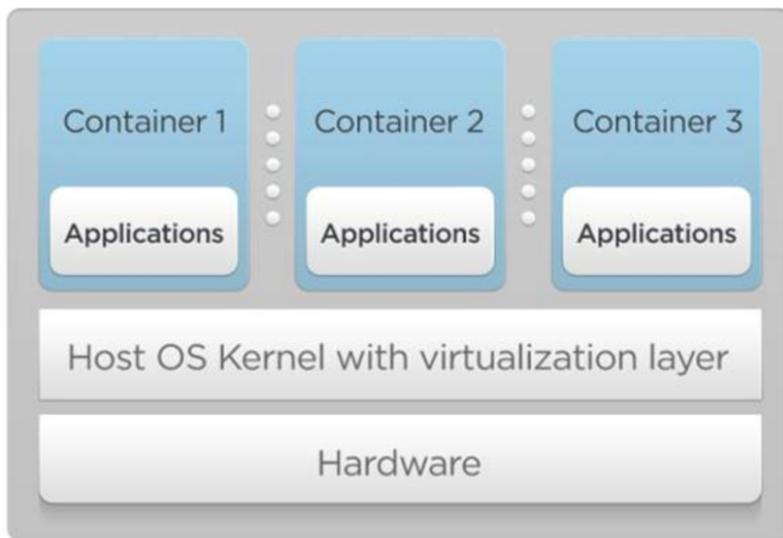


OS Virtualization

- Emulate OS-level interface with native interface
- “Lightweight” virtual machines
 - No hypervisor, OS provides necessary support

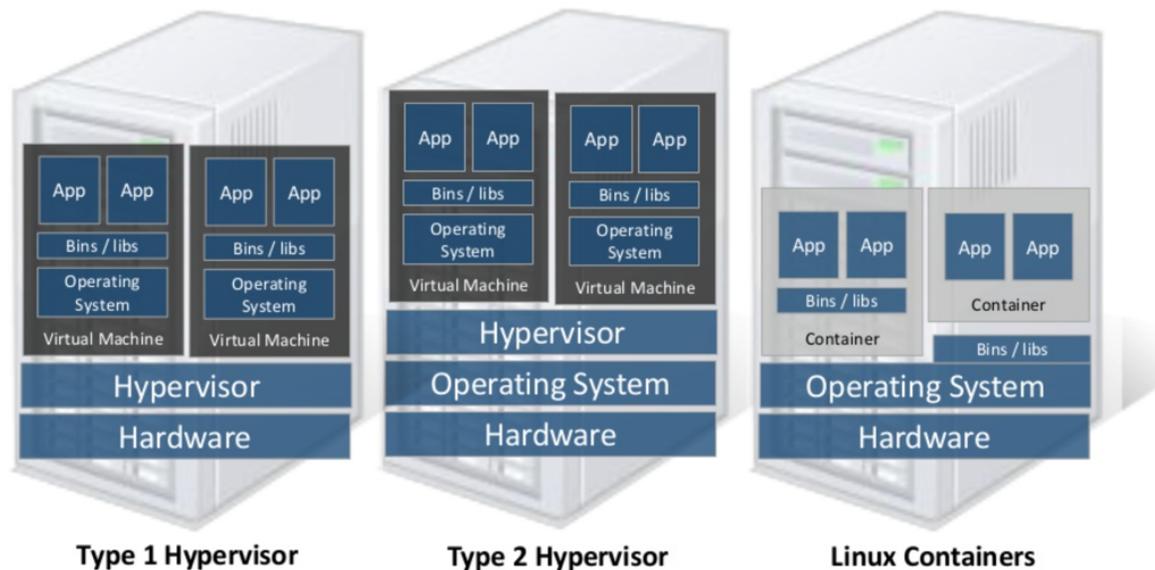


- Solaris containers, BSD jails, Linux containers



Linux Containers (LXC)

- Containers share OS kernel of the host
 - OS provides resource isolation
- Benefits
 - Fast provisioning, bare-metal like performance, lightweight



Material courtesy of
"Realizing Linux Containers"
by Boden Russell, IBM



OS Mechanisms for LXC

- OS mechanisms for resource isolation and management
- namespaces: process-based resource isolation
- Cgroups: limits, prioritization, accounting, control
- chroot: apparent root directory
- Linux security module, access control
- Tools (e.g., docker) for easy management



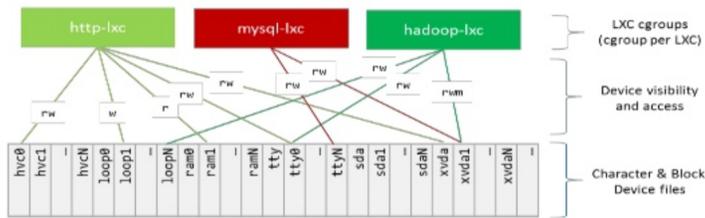
Linux Namespaces

- Namespace: restrict what can a container see?
 - Provide process level isolation of global resources
- Processes have illusion they are the only processes in the system
- MNT: mount points, file systems (what files, dir are visible)?
- PID: what other processes are visible?
- NET: NICs, routing
- Users: what uid, gid are visible?
- chroot: change root directory

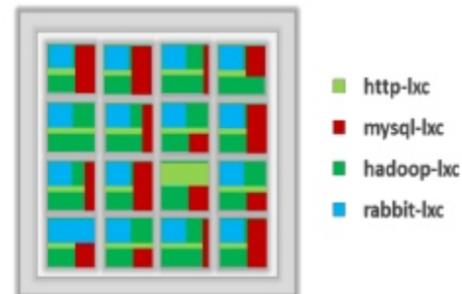


Linux cgroups

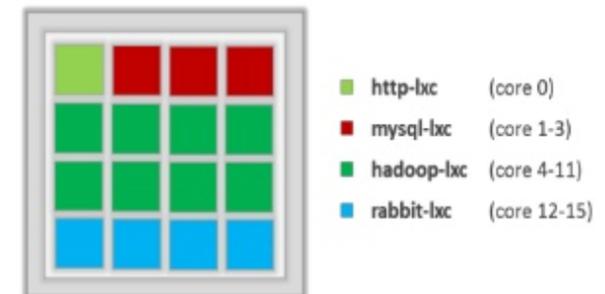
- Resource isolation
 - what and how much can a container use?
 - Set upper bounds (limits) on resources that can be used
 - Fair sharing of certain resources
- Examples:
 - cpu: weighted proportional share of CPU for a group
 - cpuset: cores that a group can access
 - block io: weighted proportional block IO access
 - memory: max memory limit for a group



Without CPU Core Pinning



With CPU Core Pinning



Proportional Share Scheduling

- Uses a variant of *proportional-share scheduling*
- *Share-based* scheduling:
 - Assign each process a weight w_i (a “share”)
 - Allocation is in proportional to share
 - fairness: reused unused cycles to others in proportion to weight
 - Examples: fair queuing, start time fair queuing
- *Hard limits*: assign upper bounds (e.g., 30%), no reallocation
- Credit-based: allocate credits every time T , can accumulate credits, and can burst up-to credit limit
 - can a process starve other processes?



Share-based Schedulers

```
From    paolo <>  
Subject [PATCH RFC RESEND 00/14] New version of the BFQ I/O Scheduler  
Date    Tue, 27 May 2014 14:42:24 +0200
```

From: Paolo Valente <paolo.valente@unimore.it>

[Re-posting, previous attempt seems to have partially failed]

Hi,
this patchset introduces the last version of BFQ, a proportional-share storage-I/O scheduler. BFQ also supports hierarchical scheduling with a cgroups interface. The first version of BFQ was submitted a few years ago [1]. It is denoted as v0 in the patches to distinguish it

[PATCH RFC 00/22] Replace the CFQ I/O Scheduler with BFQ

From: Paolo Valente
Date: Mon Feb 01 2016 - 17:50:39 EST

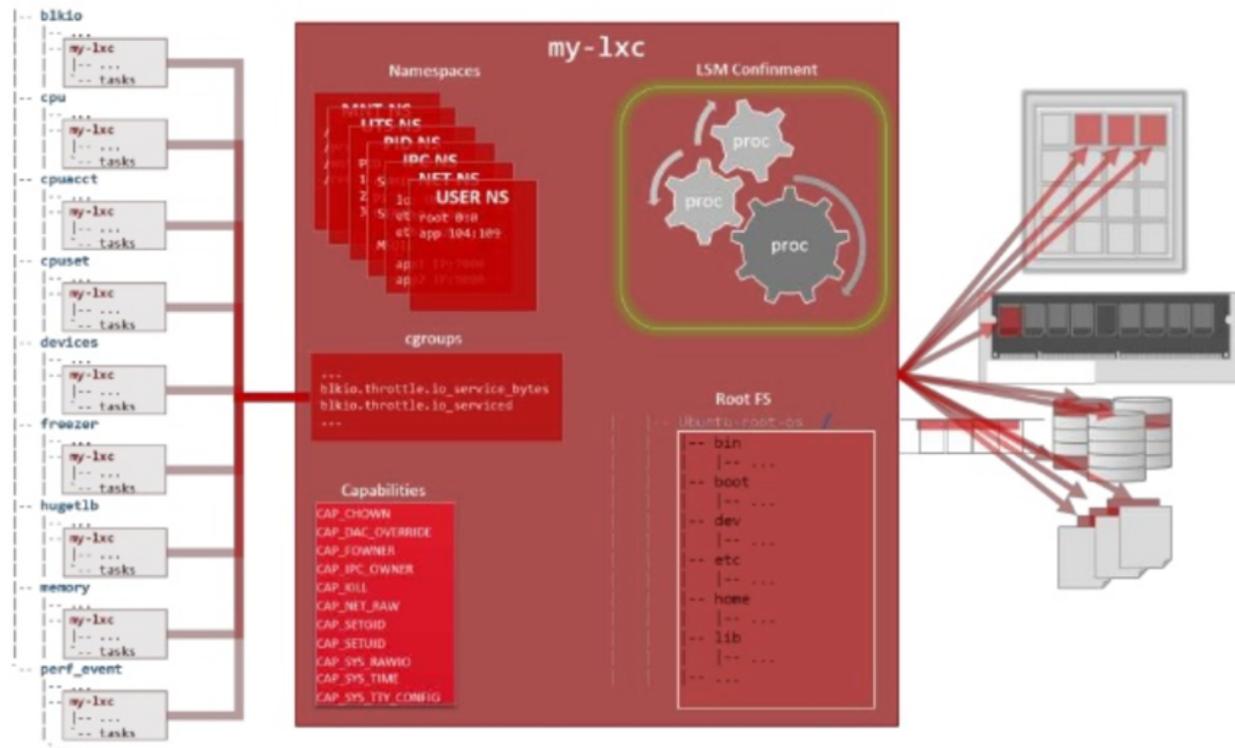
- **Next message:** [Paolo Valente: "\[PATCH RFC 03/22\] block: cfq: remove deen seek queues logic"](#)

T2 instances' baseline performance and ability to burst are governed by CPU Credits. Each T2 instance receives CPU Credits continuously, the rate of which depends on the instance size. T2 instances accrue CPU Credits when they are idle, and use CPU credits when they are active. A CPU Credit provides the performance of a full CPU core for one minute.



Putting it all together

- Images: files/data for a container
 - can run different distributions/apps on a host
- Linux security modules and access control
- Linux capabilities: per process privileges



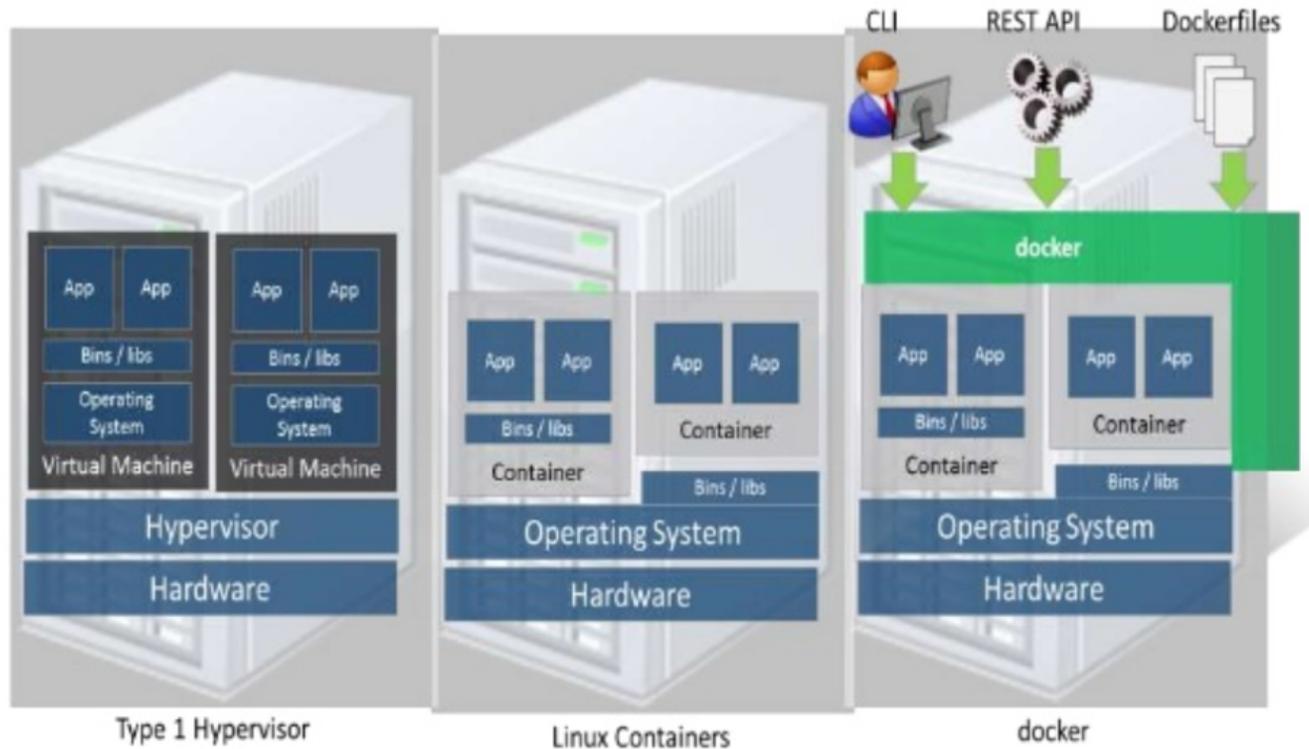
Docker and Linux Containers

- Linux containers are a set of kernel features
 - Need user space tools to manage containers
 - Virtuozzo, OpenVZm, VServer, Lxc-tools, Docker
- What does Docker add to Linux containers?
 - Portable container deployment across machines
 - Application-centric: geared for app deployment
 - Automatic builds: create containers from build files
 - Component re-use
- Docker containers are self-contained: no dependencies



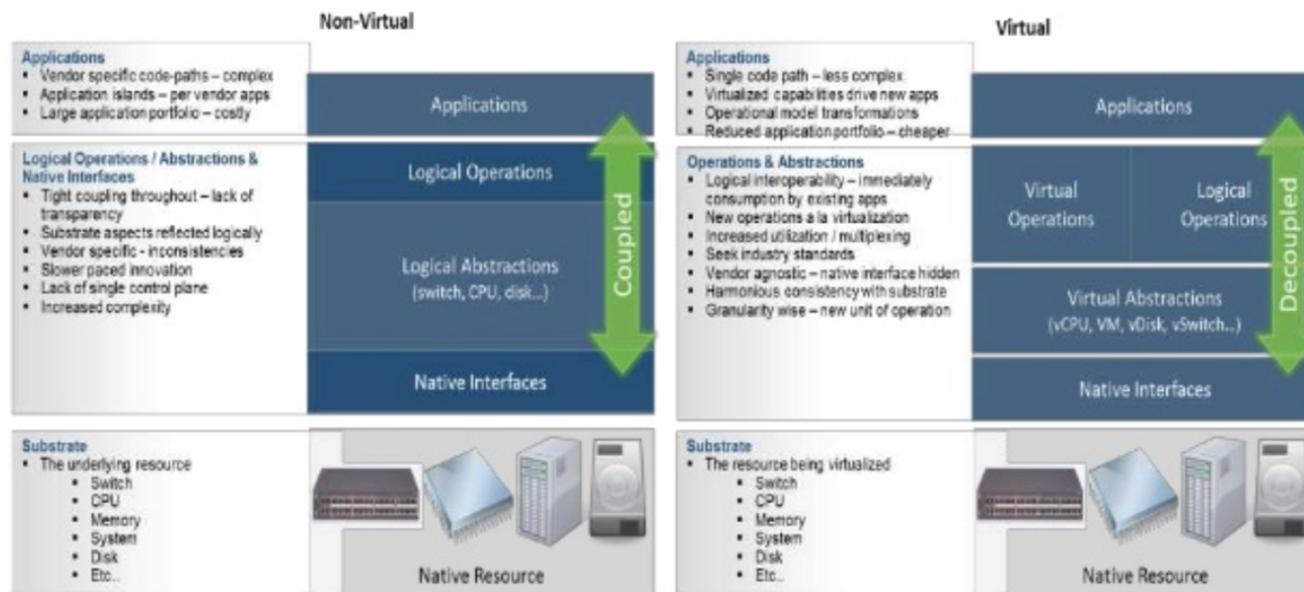
Docker

- Docker uses Linux containers



LXC Virtualization Using Docker

- Portable: docker images run anywhere docker runs
- Docker decouples LXC provider from operations
 - uses virtual resources (LXC virtualization)
 - fair share of physical NIC vs use virtual NICs that are fair-shared



Docker Images and Use

- Docker uses a union file system (AuFS)
 - allows containers to use host FS safely
- Essentially a copy-on-write file system
 - read-only files shared (e.g., share glibc)
 - make a copy upon write
- Allows for small efficient container images
- Docker Use Cases
 - “Run once, deploy anywhere”
 - Images can be pulled/pushed to repository
 - Containers can be a single process (useful for microservices) or a full OS

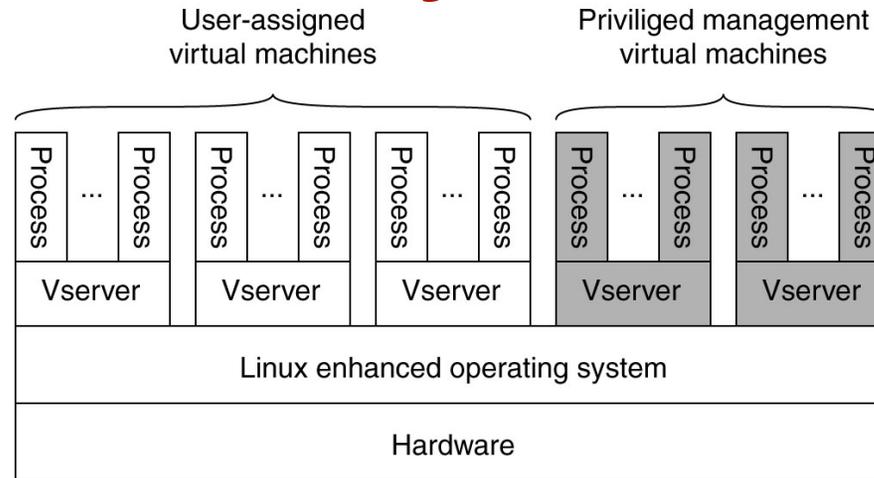


Use of Virtualization Today

- Data centers:
 - server consolidation: pack multiple virtual servers onto a smaller number of physical server
 - saves hardware costs, power and cooling costs
- Cloud computing: rent virtual servers
 - cloud provider controls physical machines and mapping of virtual servers to physical hosts
 - User gets root access on virtual server
- Desktop computing:
 - Multi-platform software development
 - Testing machines
 - Run apps from another platform



Case Study: PlanetLab



- Distributed cluster across universities
 - Used for experimental research by students and faculty in networking and distributed systems
- Uses a virtualized architecture
 - Linux Vservers
 - Node manager per machine
 - Obtain a “slice” for an experiment: slice creation service

