Containers: Design, Application & Hands-on

Prashanth
IIT Bombay
www.cspp.in

Getting Your Attention !

- Today’s talk will be applicable to many domains in CS
  - Cloud providers – IAAS, PAAS
  - HPC and Big Data
  - Support for heavy compute in ML
  - Application development
  - Resource accounting

- **Hot topic** in virtualization and app development
- Wide area to explore for your CS695 projects

Introduction

- IAAS – Provides resources as service
- Virtual machines (VM) helps resource
  - Partitioning
  - Scaling

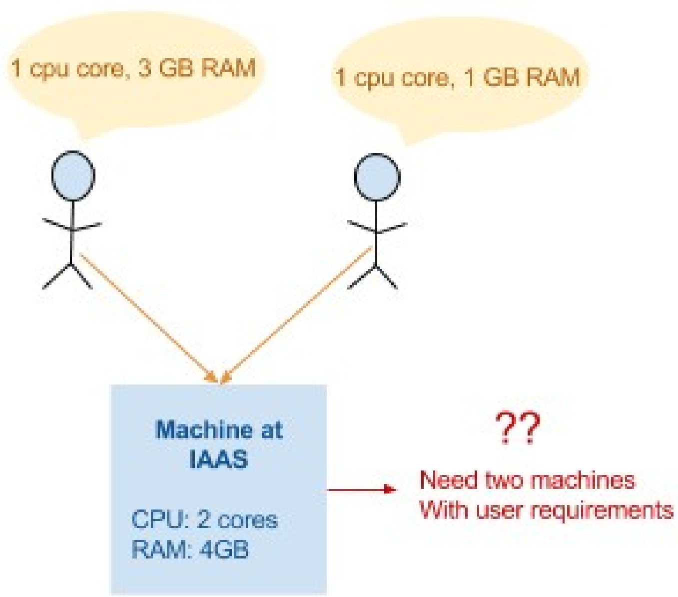


Fig: Without virtualization

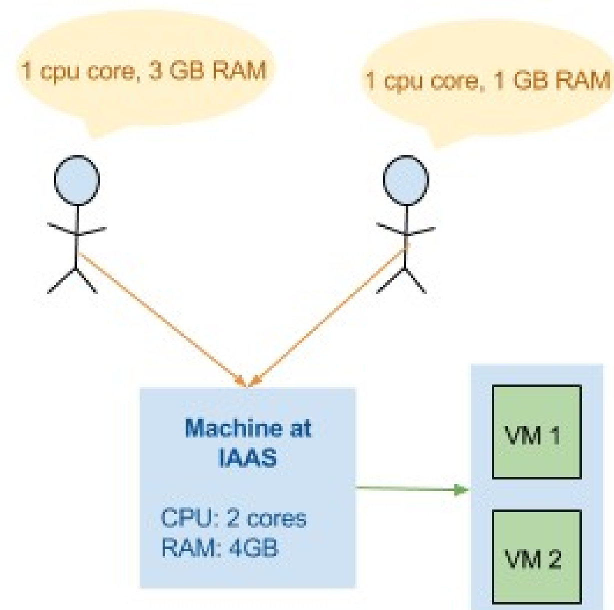


Fig: With virtualization

Issues with VM-based IAAS

Memory for each VM’s OS

- VM allocates memory for an OS leading to additional use of memory if host OS is same
- Start up latency
- Booting the OS from power off causes delays
- Dual control loop
- Scheduling for each resource happens at guest and host, leading to delays
- Complete hardware stack emulation
- Full virtualization requires emulation of hardware which utilizes compute resources

The issues mentioned above leads to overheads which in turn leads to bad cost-benefit ratios which adversely affects customers by overpricing services offer by IAAS

Requirements of IAAS provider

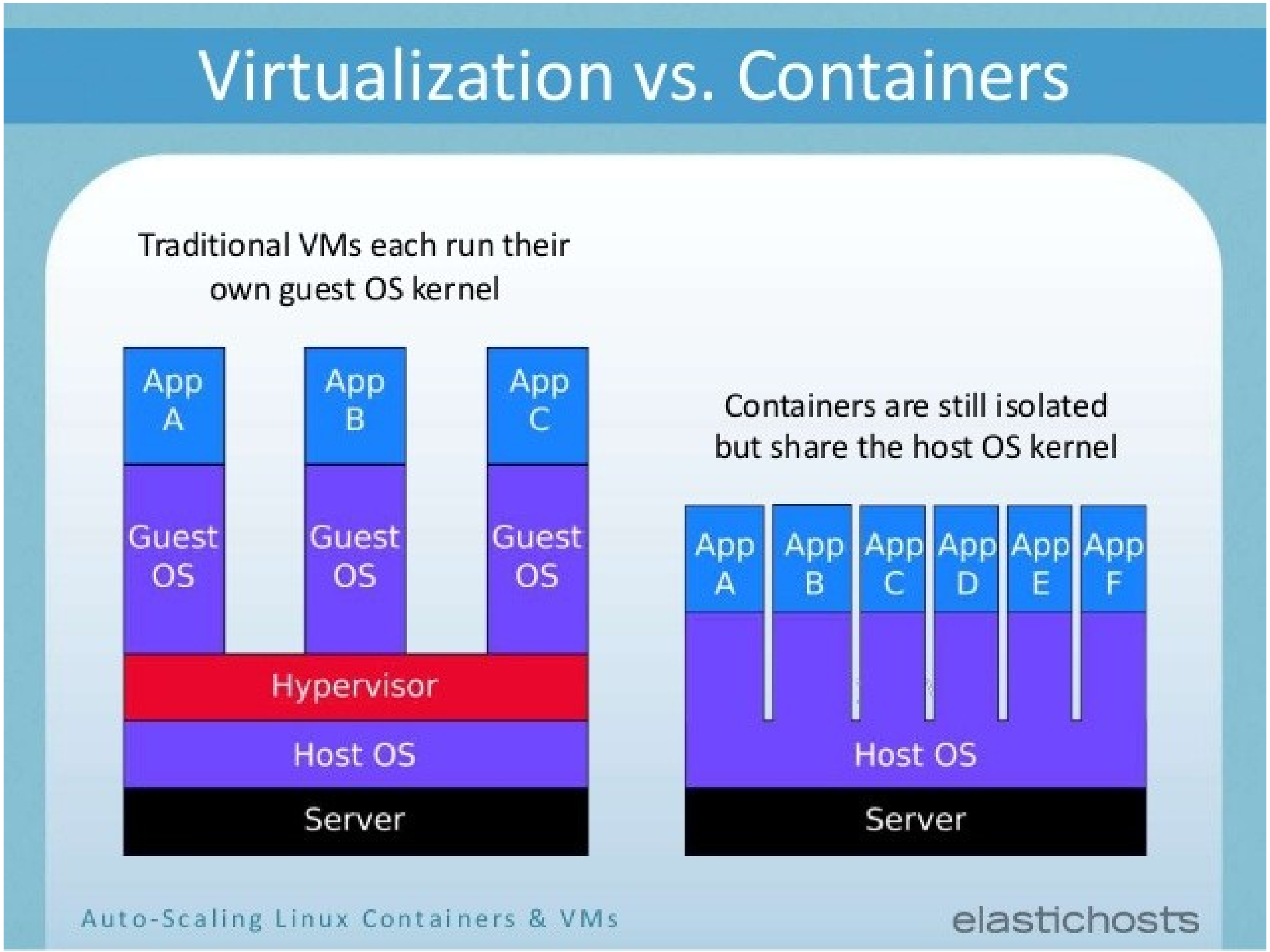
Desired features for a Virtual Environment (VE)

Resource control
1. Limit the amount of resource being utilized
2. Isolation
3. Running of application in one VE shouldn't be affect by the other VEs executing
4. Accounting of resource
5. Each resource utilized by an VE must be accountable
6. Resource provisioning
   - Deterministic – Maintain desired behavior
   - Elastic – Change resources provisioned (if desired)
   - Reuse of host OS functionality
7. Reusing host features whenever possible to avoid overheads when enforcing above

Container

“ Container is a virtual environment that contains a set of processes grouped along with its dependent resources into a single logical OS entity. “

- Also known as **OS Virtualization** (Reason: Next



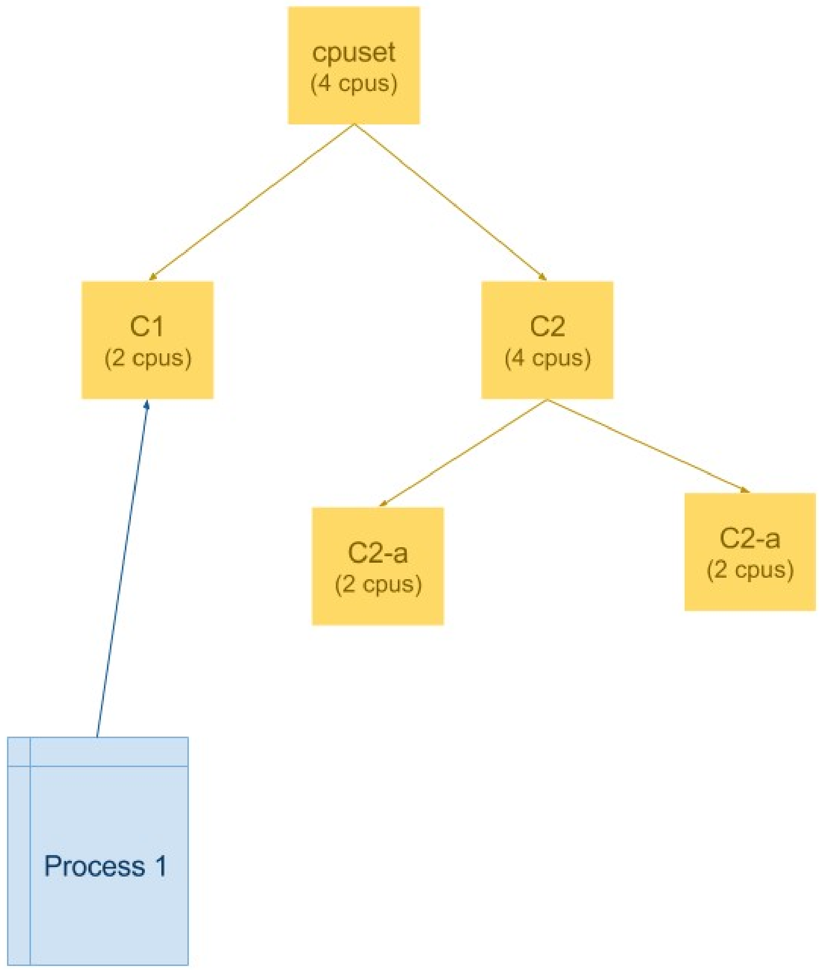
Reference: [16]

Building blocks of Containers

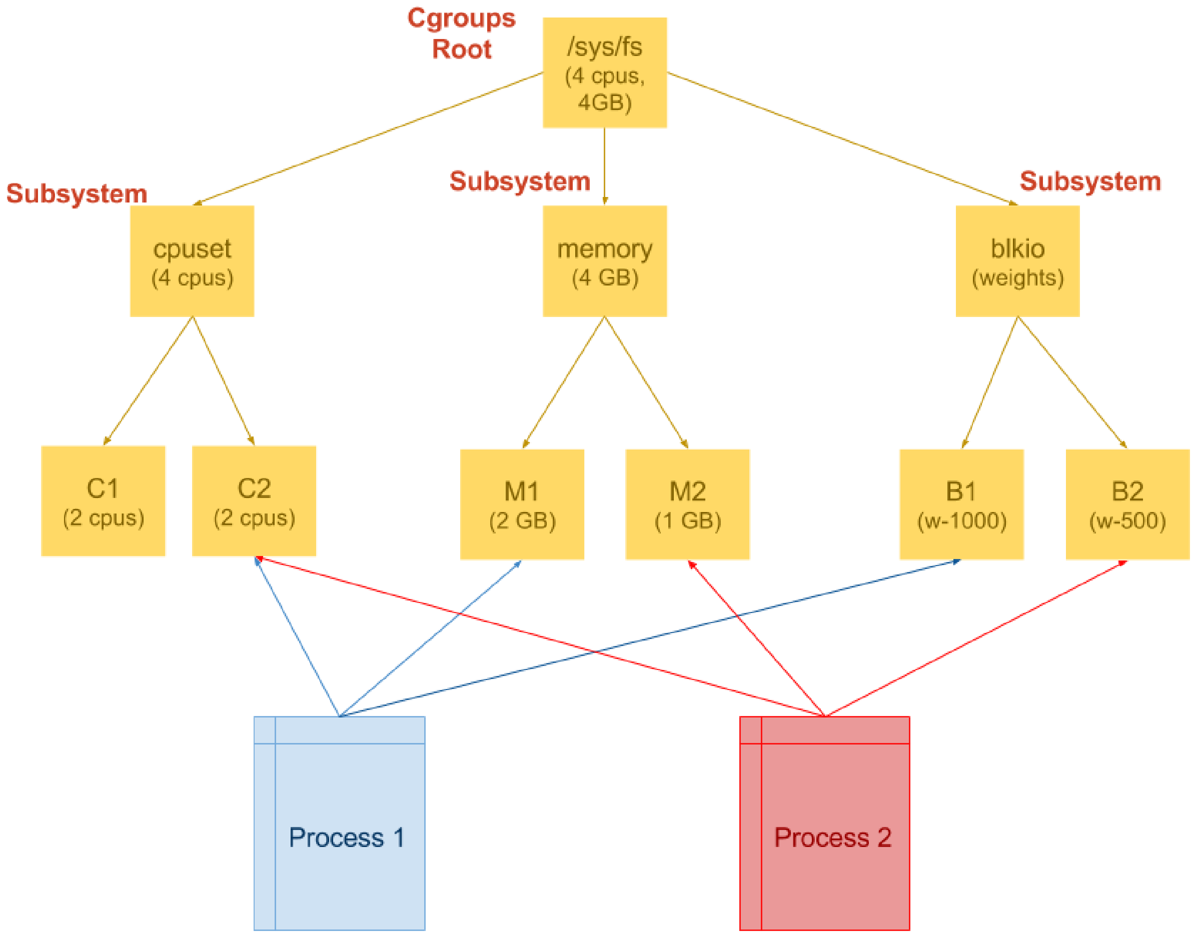
Control Groups (cgroups)

- **Resource controller** for each resource
- 12 different subsystems – CPU, memory etc.
- Perform **Accounting**
- Enforcing resource **Restriction**
- Follows hierarchy
- User space API – pseudo file-system

Reference: [1]



CPUSET Subsystem: Allows pinning of processes to CPU cores



Multiple Subsystems: **Attach a process to one node in each subsystem**

Situation

- You have 5 processes (PIDs 1-5) and you wish to divide them into two groups of processes with following constrains

  ❑Group 1
    - PIDs: 1,2
    - 4 CPUs, 4GB RAM, 2x Disk access rate

  ❑Group 2
    - PIDs: 3, 4, 5
    - 1 CPU, 4GB RAM, 1x Disk access rate

- Also you must be able to track their resource usage for each group

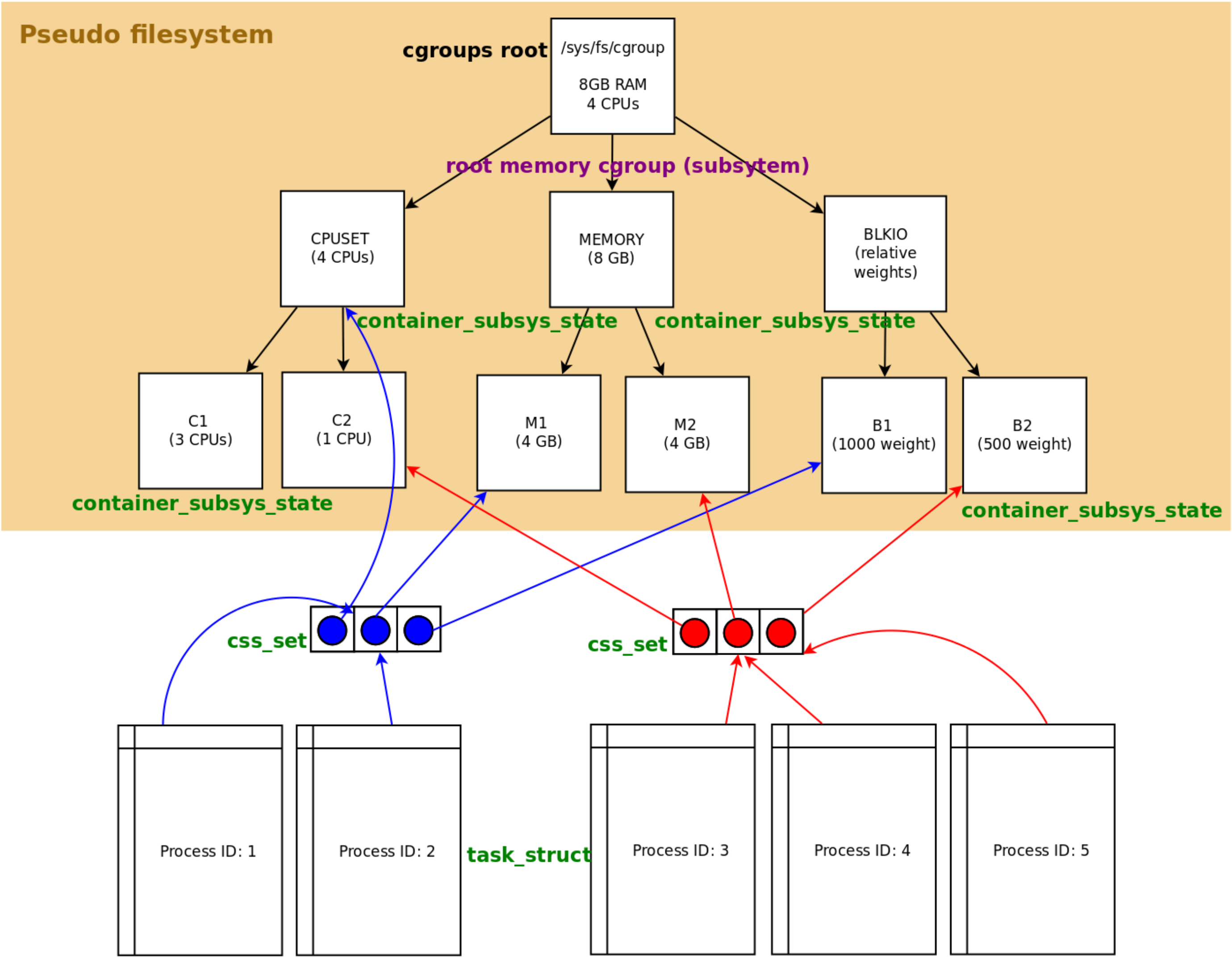


Fig: Control groups illustration using 3 controllers

LABELS

Violet:
Resource controller

Green:
Kernel Data structures

Blue:
Pointers for group 1

Blue:
Pointers for group 2

Black Boxes:
Directories used to manage cgroup nodes

Cgroups Demo

- Demo with memory (and cpu depending on time) cgroup
- Creating process attaching to cgroup, accounting, and setting limit

Namespaces

- **Isolated system views**, 6 namespaces, Each namespaces has multiple isolated environments.

- Each container is attached to 1 isolated namespace in all 6 types (similar to cgroups)
    1. Mount – Each container its own view of system files
    2. PID – Container processes are isolated from other container processes
    3. Network – Only aware of its network resources
    4. IPC – IPC communication local to container
    5. UTS – Host names and domain names can be different
    6. User – Users in each container are local

- API – passing flags to clone()

Reference: [2], [3]

Situation

A situation where you have N processes, and you wish to isolate them from other processes in the system in such a way that,

- Our processes must not be able to see/interact with other processes in the system
- We have our own range of PIDs for our processes

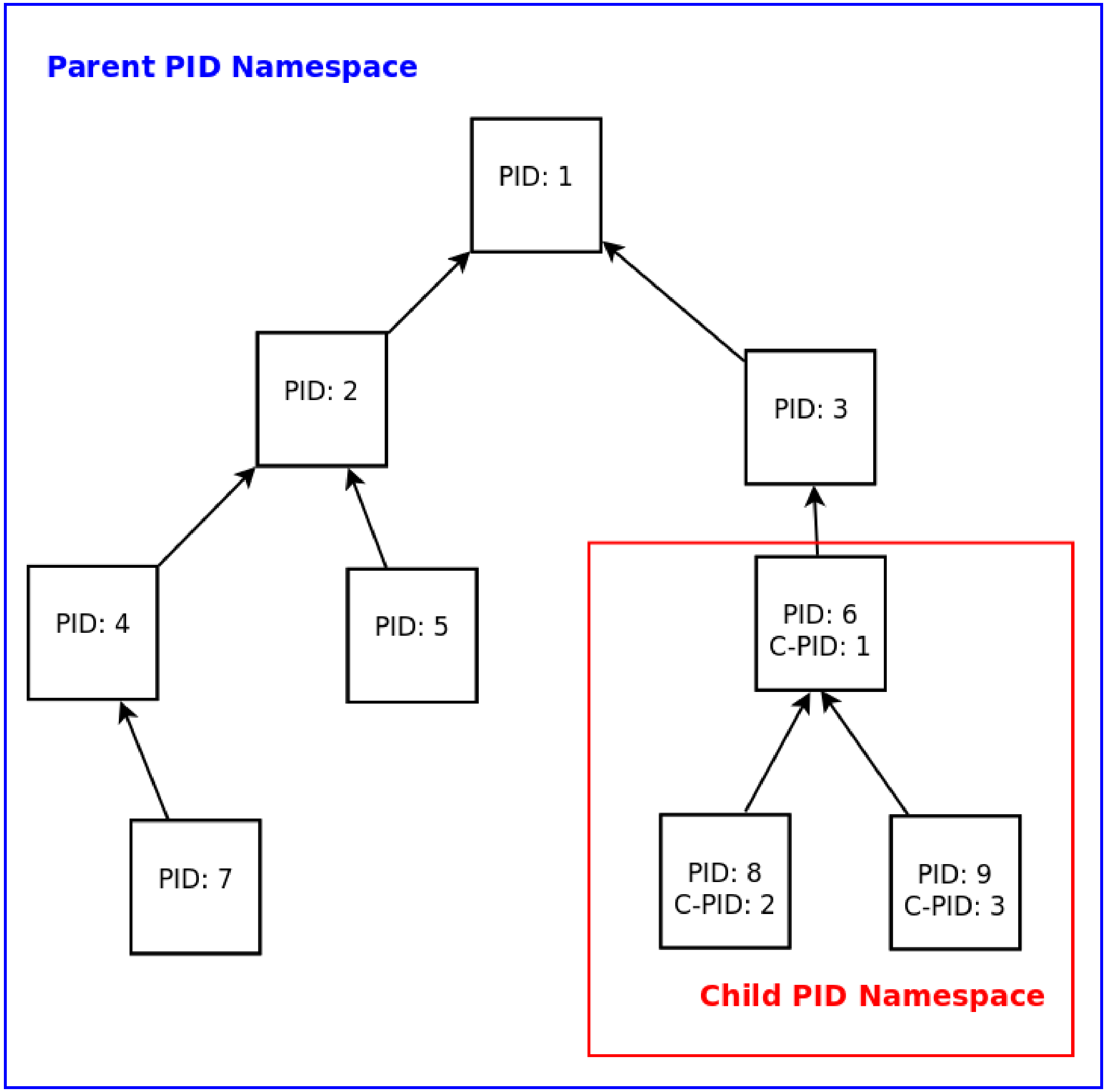
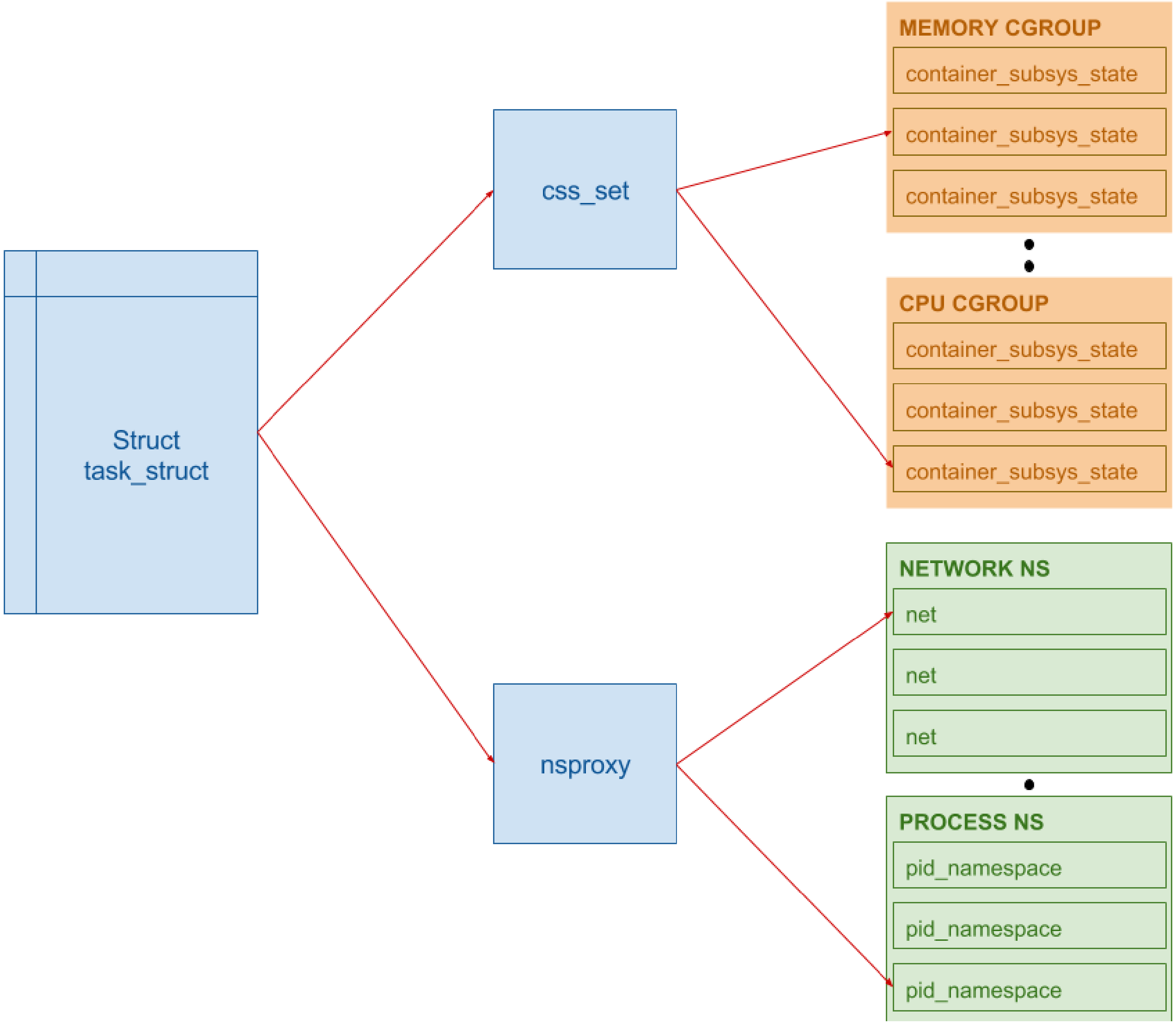


Fig: Example of PID Namespace in which pids 6,8,9 in parent map to 1,2,3 in child



LABELS

Orange Cgroup/subsystem

Green Namespaces

Fig: Kernel Data structure modifications to account for cgroups and namespaces

Container Disk Images

- Provides new mount point – avoid changing data of host
- New ROOTFS – mount namespace
- Smaller than the normal OS-disk image – No kernel
- Disk image could also contain only application

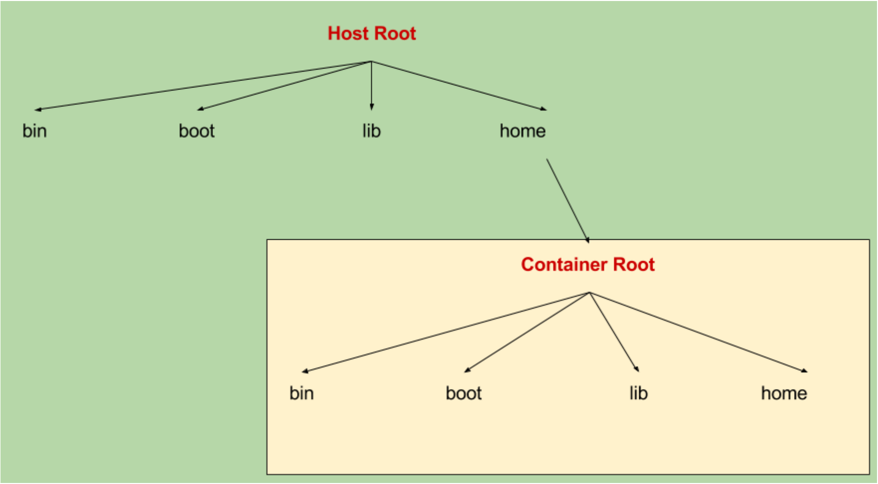
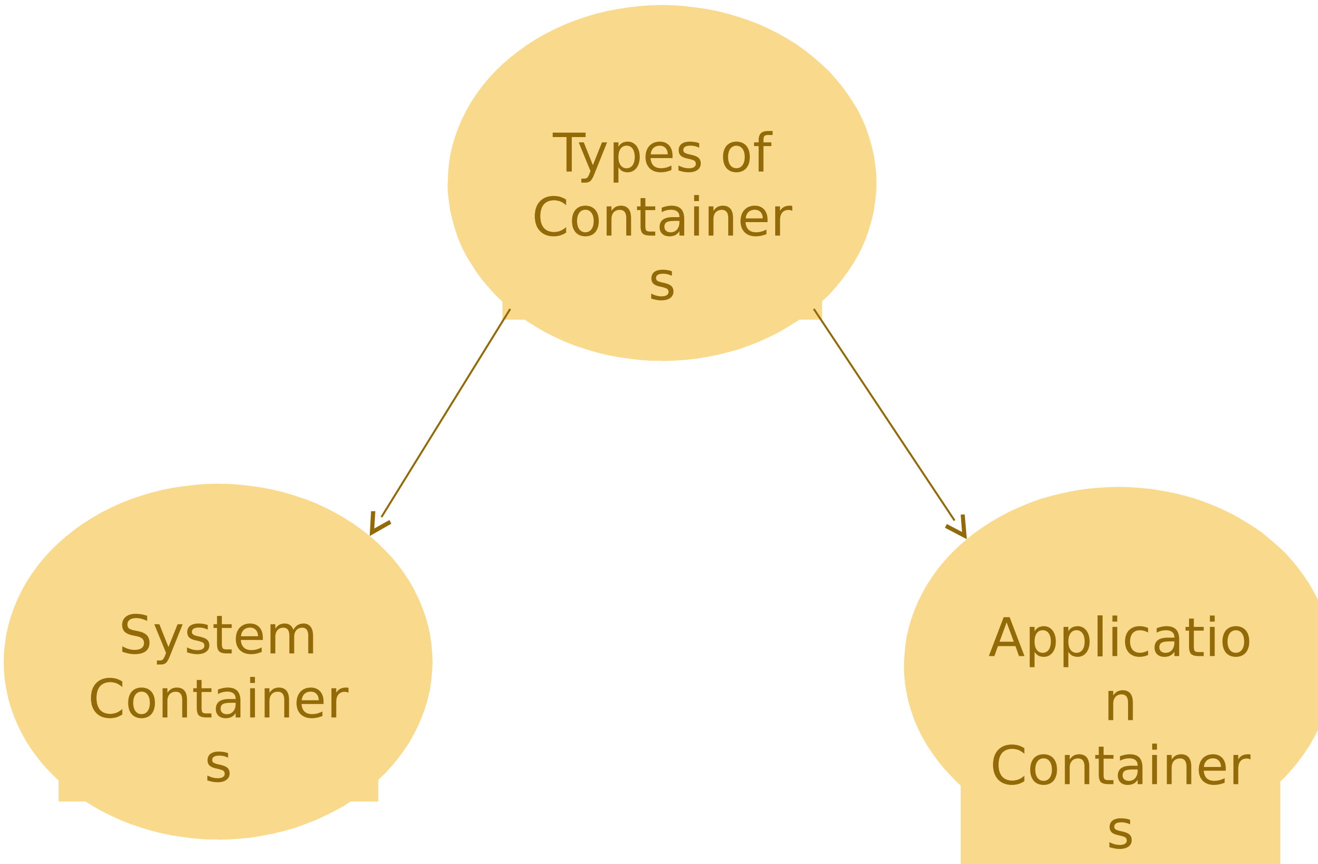


Fig: mount namespace used to mount a new container root


Types of Container s
System Container s
Applicatio n Container s

System Containers

- Environment similar to native machine
- Install, configure, run – apps, libraries, demons
- Used by cloud providers
- Have been used for a while
- Examples
    1. Linux Containers (LXC)
    2. Parallels virtuizzo
    3. Solaris zones
    4. Google lmctfy

Reference: [7], [8]

Linux Containers (LXC)

- API to deploy system containers
- Configured via CLI
- Image fetched from online repository – first time
- There after – local cache
- New container – image copied

Application containers

- Develop, build, test, ship and even run apps
- Recent – 2013
- Multiple apps – 1 container for each
- Cloud-native apps
- Examples
    1. Docker
    2. Rocket

Reference: [6]

Docker Architecture

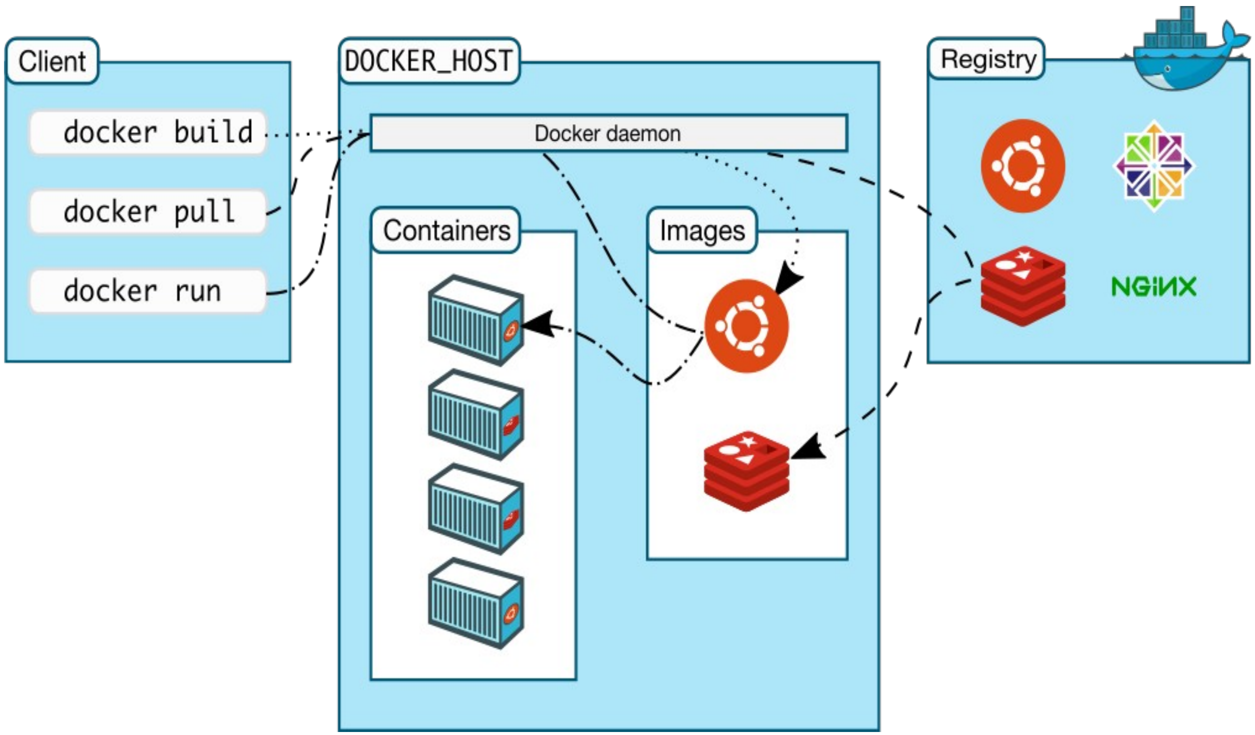


Fig: Docker Architecture, source: [6]

COMPONENTS

1. **Client:** UI to manage containers
2. **Host:** Build & Run containers
3. **Registry:** Image store

Images:

4. Read-only template

Containers:

5. Created from image

Docker Image layers

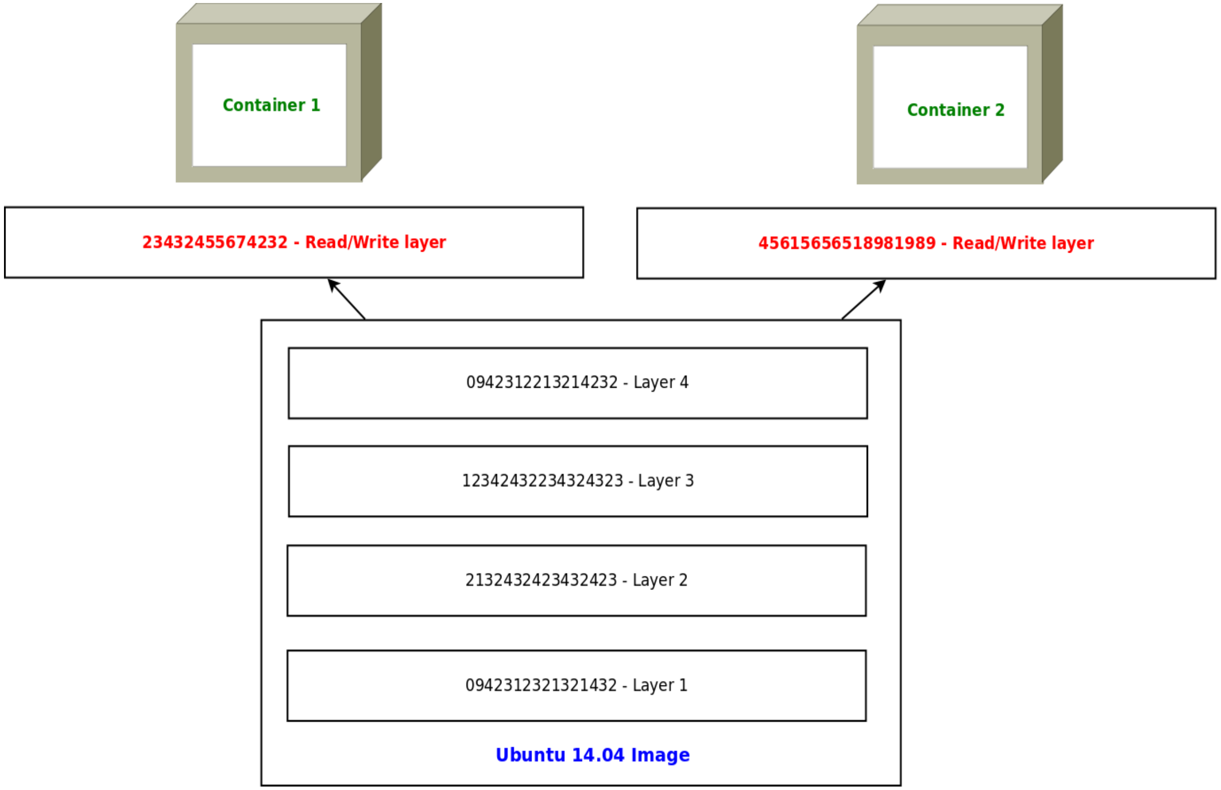


Fig: Docker image layers

POINTS

- Stackable image layers
- Reuse layers
- Copy-On-Write (CoW)
- Container adds Read-Write layer on image
- Commit makes layer read only

Containers Demo

- Short demo
- Starting a container with Lxc/Docker and how they differ

Application of containers

- System containers
  1. Cloud providers (IAAS/PAAS)
  2. Data centers
  3. Potentially anywhere instead of VM

- Application containers
  1. HPC clusters
  2. Application development
     - Sandboxing applications with dependencies
     - Micro services & Scalability
     - Version Control – Github alternative

Reference: [5], [10]

Kubernetes

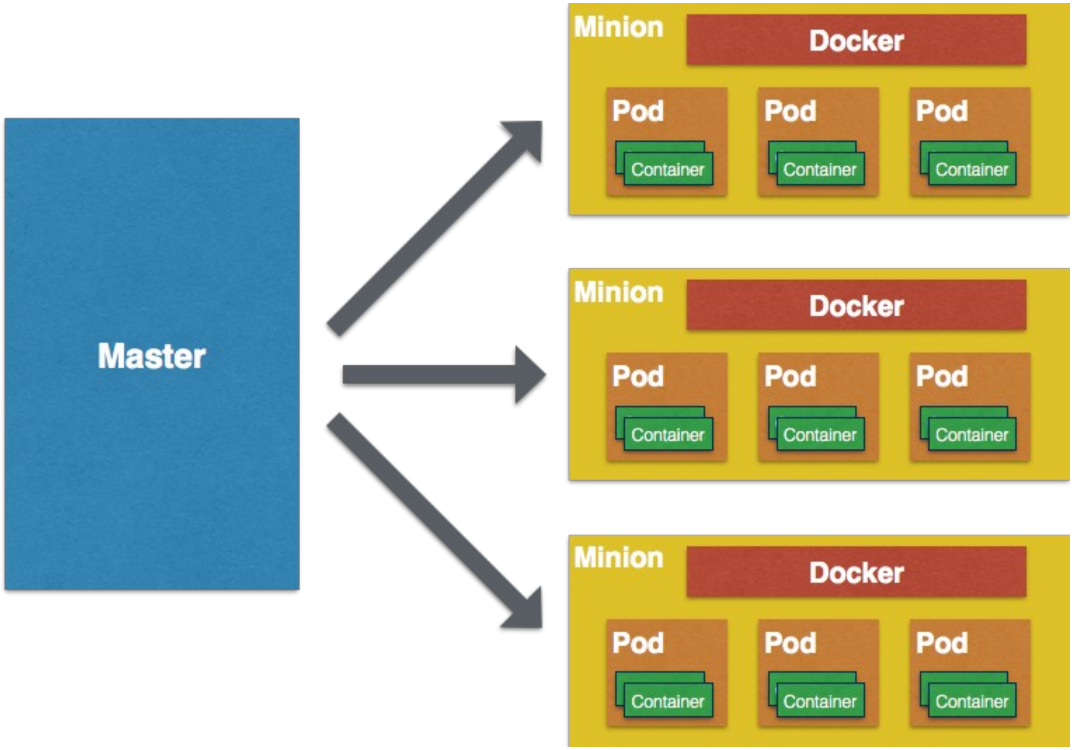


Fig: Container orchestration using Kubernetes, source [5]

- Container Orchestration Tool, originally designed by Google
- Automated Deployment, Management and Scaling
- Groups application into logical units – pods
- Minion is PM
- Manages services and also batch processes

Merits and Demerit of containers

Merits

- Startup latency minimal
- No hardware emulation
- No multiple OS copies
- Overheads - close to native

Demerits

- Only base kernel type containers
- Security

Comparing Containers to VMs

Container is better at

- Memory Usage – VM takes 11-60x container’s usage
- Disk I/O – VM takes 2x
- CPU utilization – Marginally better
- Startup Latency – VM typically takes about 50-100x

VM is better at

- Network – VM is 1.2x better here
- Live-Migration – Better in VMs
- Support for guest of OS of different kernel
- Security

Reference: [9], [10], [11], [12], [13

Related Works

- CoreOS – Linux distro for container management
- OSv - OS designed for the Cloud and is treated as a library operating system
- LXD - Next generation hypervisor for containers
- Disk Image Standardization

Reference: [17], [18], [19], [20]

Conclusion

- Performance overheads - Big win
- Tremendous potential
- Limitation of a container is the ability to only run OS of host kernel type

Possible Projects (Future Work)

Disk & Storage
- Comparative study of the different container imaging formats and providing use cases for each imaging format
- Extending BLKIO cgroup support to SSDs

Memory
- Design a per memory cgroup accounting enable/disable knob
- Shared pages accounting in containers charges the first cgroup that accesses it, design and implement solution to rectify this

Network
- Explore network cgroups, come up with drawbacks and propose new solutions to fix issues (will have to work with tc

Possible Projects (Future Work)

Application-level

- Deploy multi tier applications using Kubernetes and come up different ways to achieve load balance.
- Comparative study of LXD versus Docker and provide use cases

Miscellaneous

- Study the feasibility for reusing of host OS packages inside containers by implementing the same
- Live migration of containers – Look into CRIU

References

Components of container
[1] P. B. Menage, “Adding generic process containers to the linux kernel," in Proceedings of the Linux Symposium , vol. 2, pp. 45{57, Citeseer, 2007.
[2] M. Kerrisk, “Lwn namespaces overview," 2013.
[3] Michael Kerrisk “namespaces in operation”, https://lwn.net/Articles/531114/, 2013

Container
[4] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in OSDI , vol. 99, pp. 45{58, 1999.
[5 http://blog.arungupta.me/wp-content/uploads/2015/01/kubernetes-key-concepts.png
[6] D. Inc., “Docker offical documentation," 2016.
[7] K. Kolyshkin, “Virtualization in linux,“ White paper, OpenVZ , vol. 3, p. 39, 2006.
[8] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in ACM SIGOPS Operating Systems Review, vol. 41, pp. 275{287, ACM, 2007.
[16] http://image.slidesharecdn.com/linuxcontainers-thefutureofiaas-140620073031-phpapp02/95/linux-containers-the-future-of-iaas-4-638.jpg?cb=1403249627

Comparison with VMs

[9] K. Agarwal, B. Jain, and D. E. Porter, “Containing the hype," in Proceedings of the 6th Asia-Pacific Workshop on Systems , p. 8, ACM, 2015.
[10] D. Beserra, E. D. Moreno, P. Takako Endo, J. Barreto, D. Sadok, and S. Fernandes, “Performance analysis of lxc for hpc environments," in Complex, Intelligent, and Software Intensive Systems(CISIS), 2015 Ninth International Conference on, pp. 358{363, IEEE, 2015.
[11] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers," in Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On, pp. 171{172, IEEE, 2015.
[12] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in Cloud Engineering (IC2E), 2015 IEEE International Conference on , pp. 386{393, IEEE, 2015.
[13] M. S. Rathore, M. Hidell, and P. Sj odin, "Kvm vs. lxc: comparing performance and isolation of hardware-assisted virtual routers,“ American Journal of Networks and Communications , vol. 2, no. 4, pp. 88{96, 2013

Disk I/O and storage driver optimizations

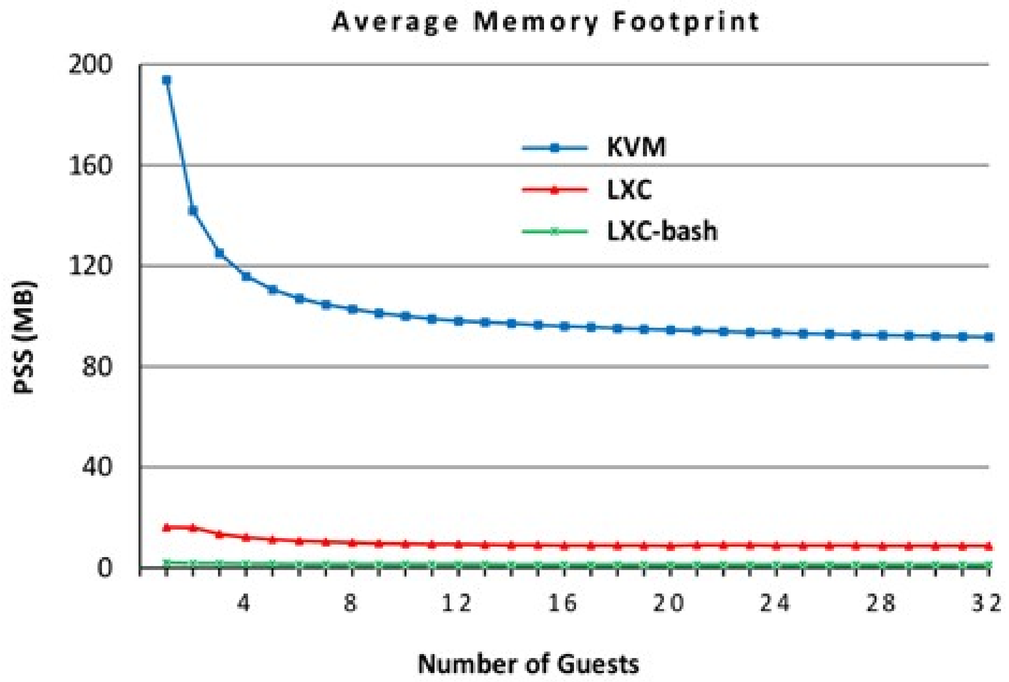
[14] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “Slacker: Fast distribution with lazy docker containers,"
[15] J. Kang, B. Zhang, T. Wo, C. Hu, and J. Huai, “Multilanes: providing virtualized storage for os-level virtualization on many cores," in Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14), pp. 317{329, 2014.

Related Works

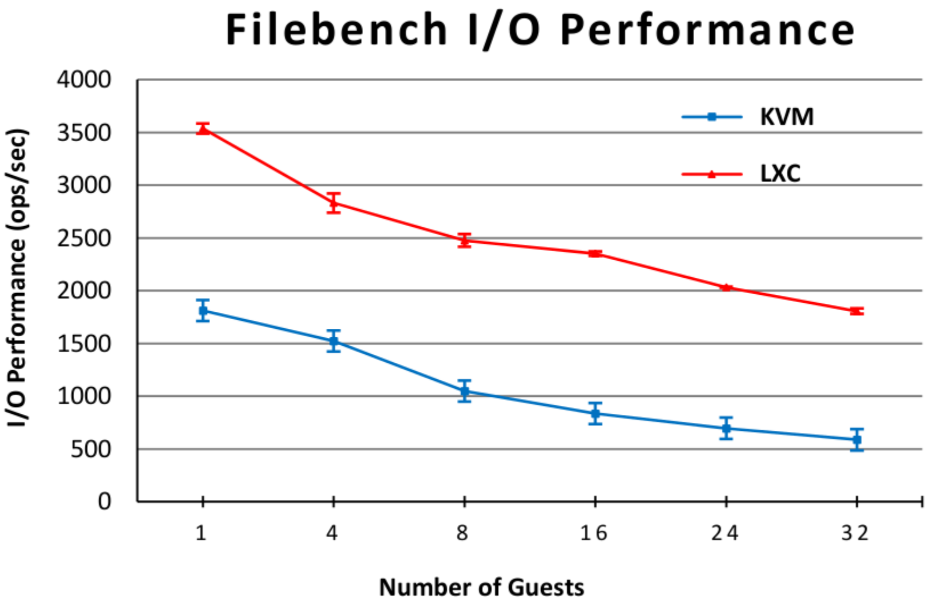
[17] CoreOS – https://coreos.com/
[18] Osv – https://osv.io/
[19] LXD – https://linuxcontainers.org/lxd/
[20] Disk Image Standarization - http://thenewstack.io/open-container-initiative-launches-container-image-format-spec/

Backup Slides

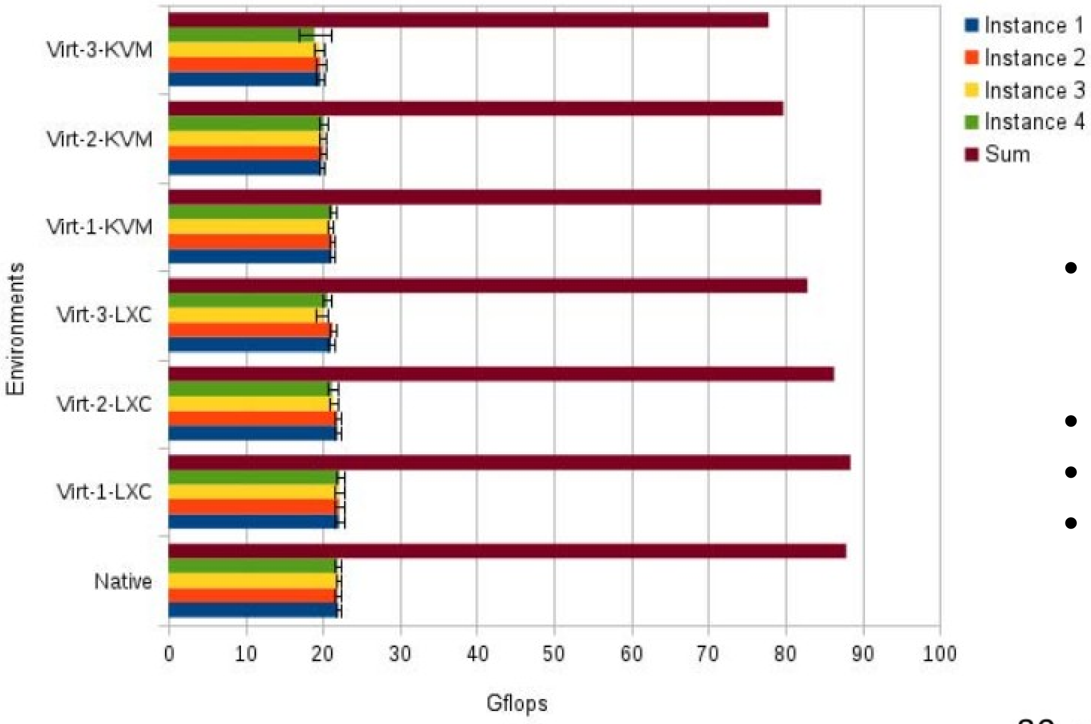
Not meant for presentation



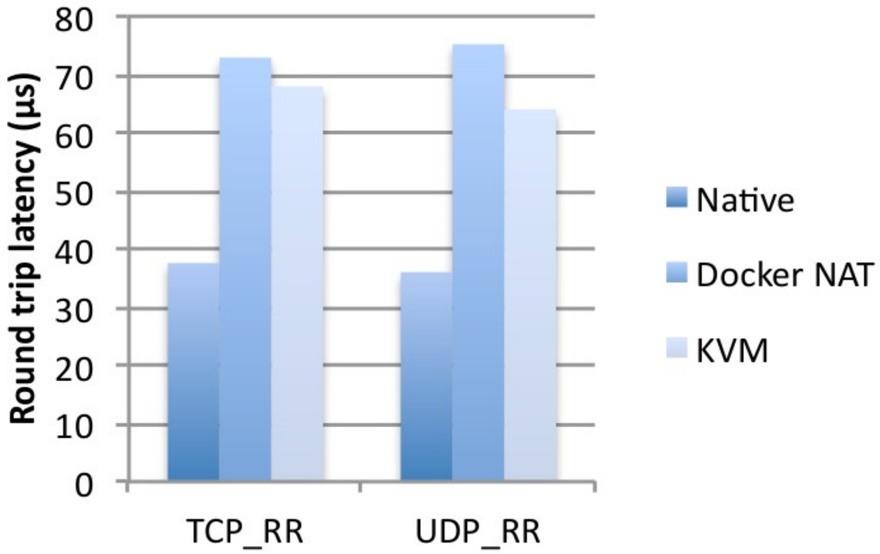
- Increasing number of guests and how it effects memory size
- lower the better
- 11-60x better in containers
- Source [9]



- Increasing number of guests and how it effects I/O throughput
- higher the better
- Optimization: direct map in VM
- source [9]



- Increasing number of guests in HPC environment and how it effects CPU throughput
- Higher the better
- 2-22% lesser in VM
- source [10]



- Effect on RTT – client-server
- lower the better
- VM (80%) > container (100%)
- source [11]

Memory Cgroups Commands

- cd /sys/fs/cgroup
- mkdir memory
- mount -t cgroup -o memory cgroup /sys/fs/cgroup/memory
- echo {{pid}} > cgroups.procs
- memory.stat
- echo 128M > memory.limit in bytes
- cat memory.usage in bytes

Container commands

- lxc-create -n test-container -t ubuntu
- lxc-ls –fancy
- lxc-start -n test-container –d
- lxc-console -n test-container
- /var/lib/lxc/test-container/config

- docker -m 512M -it ubuntu /bin/bash
- docker ps -a