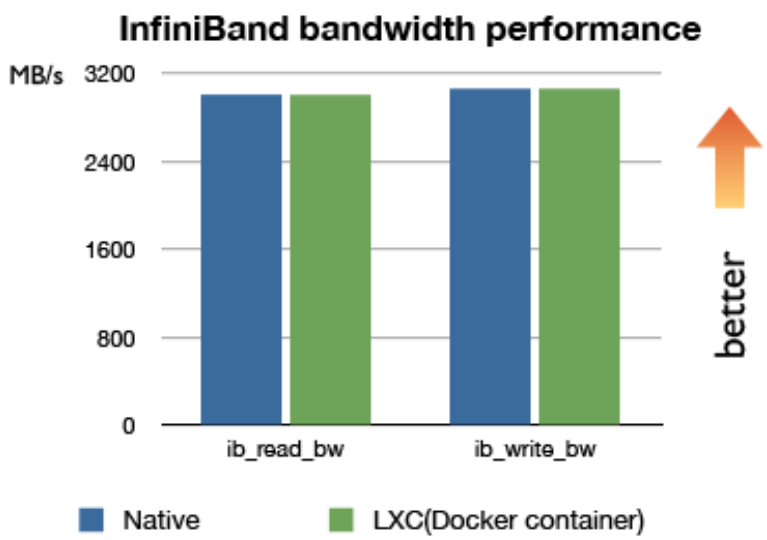
Lightweight virtual system mechanisms (containers)

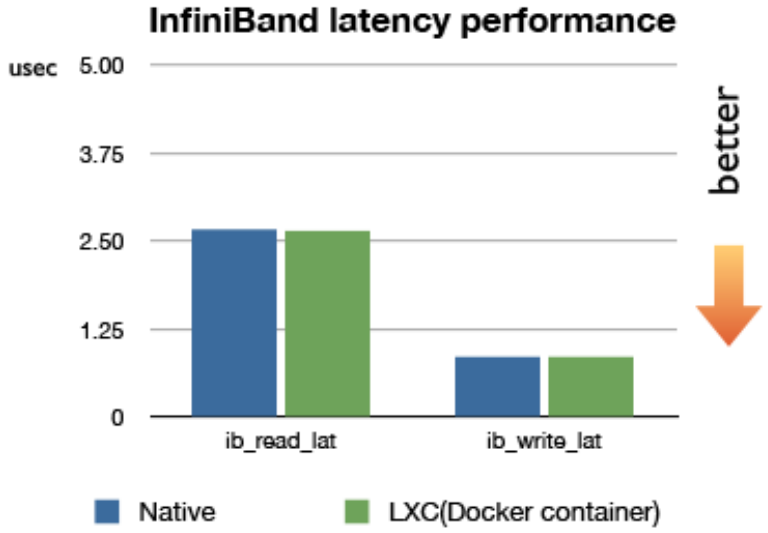
Cloud based networks

vitmma 02

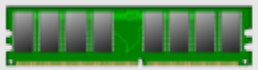
KONTÉNEREK

Performance: over 9k-times better





Konténer metafora: áruszállítás

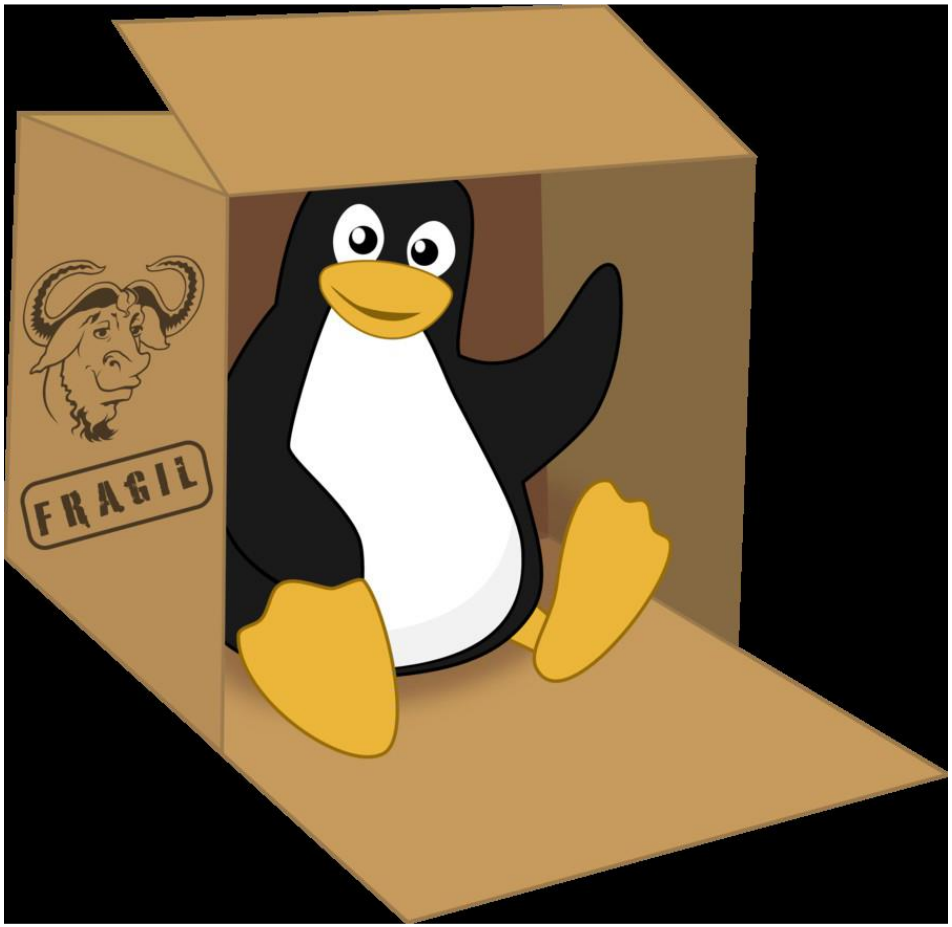
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
						

Konténer metafora: inter-modális konténer

Kódok „szállítása” virtualizációs megoldásokhoz

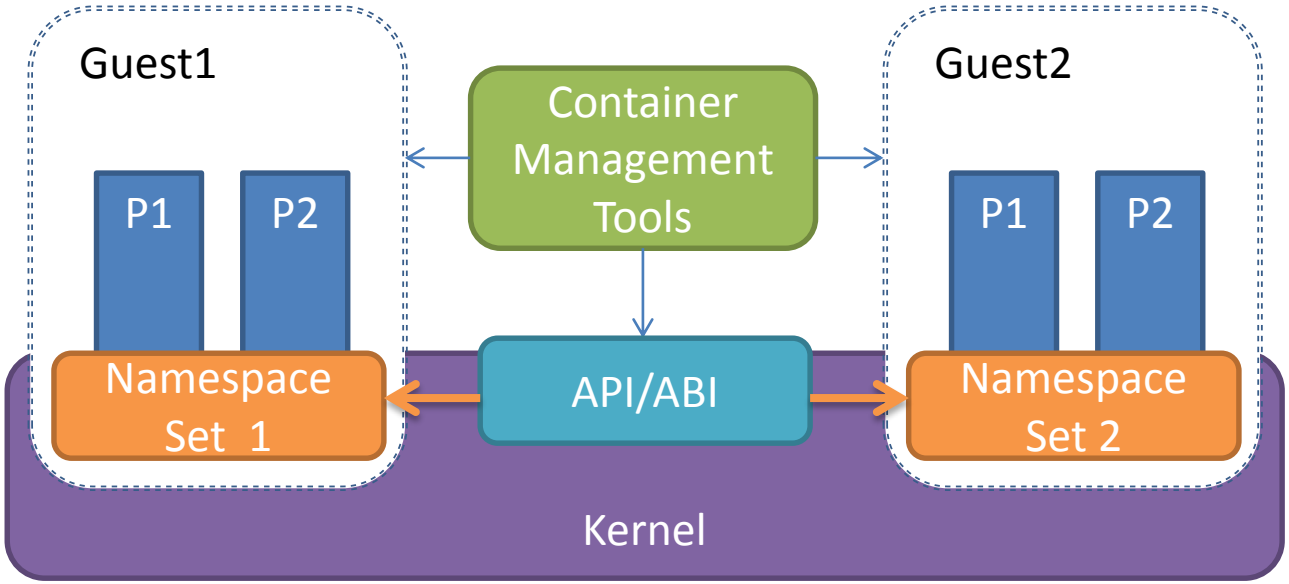
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
	?	?	?	?	?	?
						

A Linux konténerek mindent megoldanak (khm)



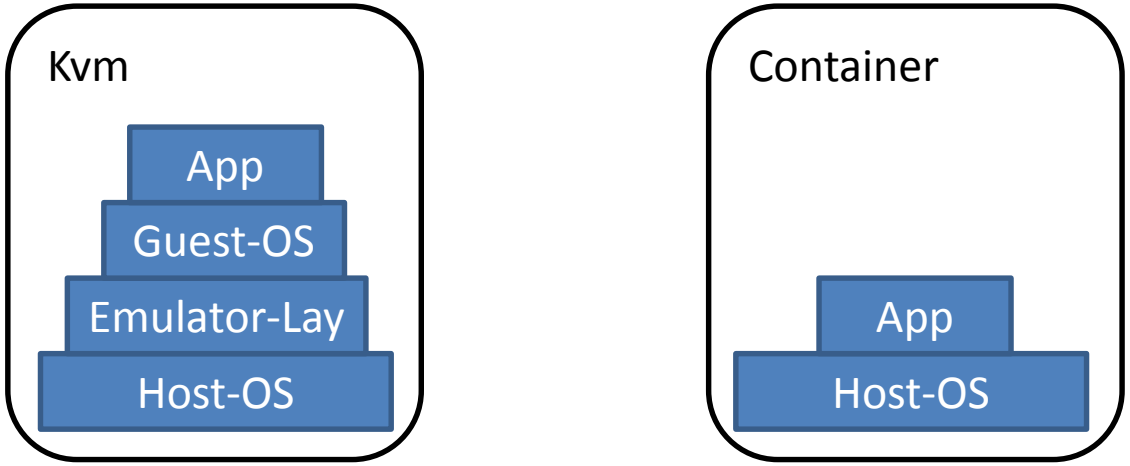
Introduction

- Container: Operation System Level virtualization method for Linux



Introduction

- Why Container
  - Better Performance



- Easy to set up Multi-Tenancy environment

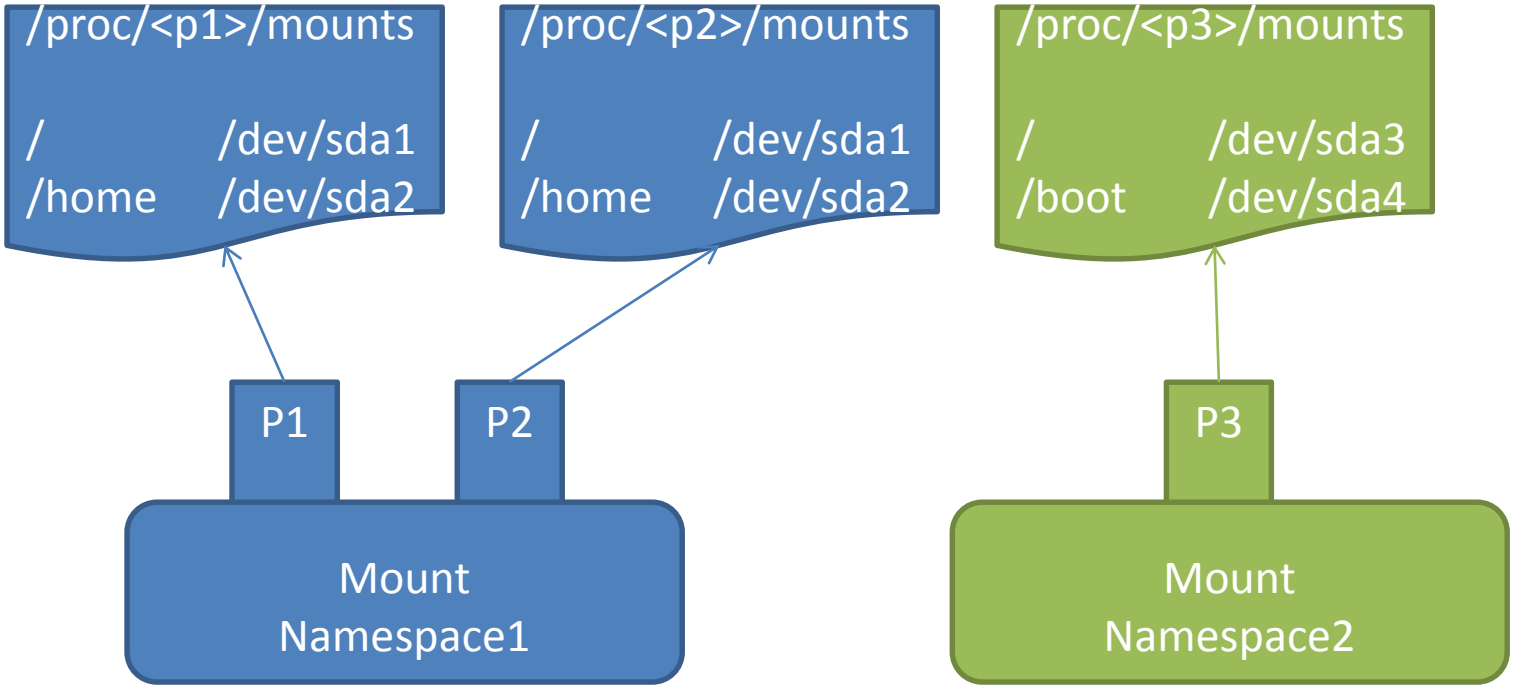
NAMESPACE

Namespace

- Namespace isolates the resources of system, currently there are 6 kinds of namespaces in linux kernel.
  - Mount namespace
  - UTS namespace
  - IPC namespace
  - Net namespace
  - Pid namespace
  - User namespace

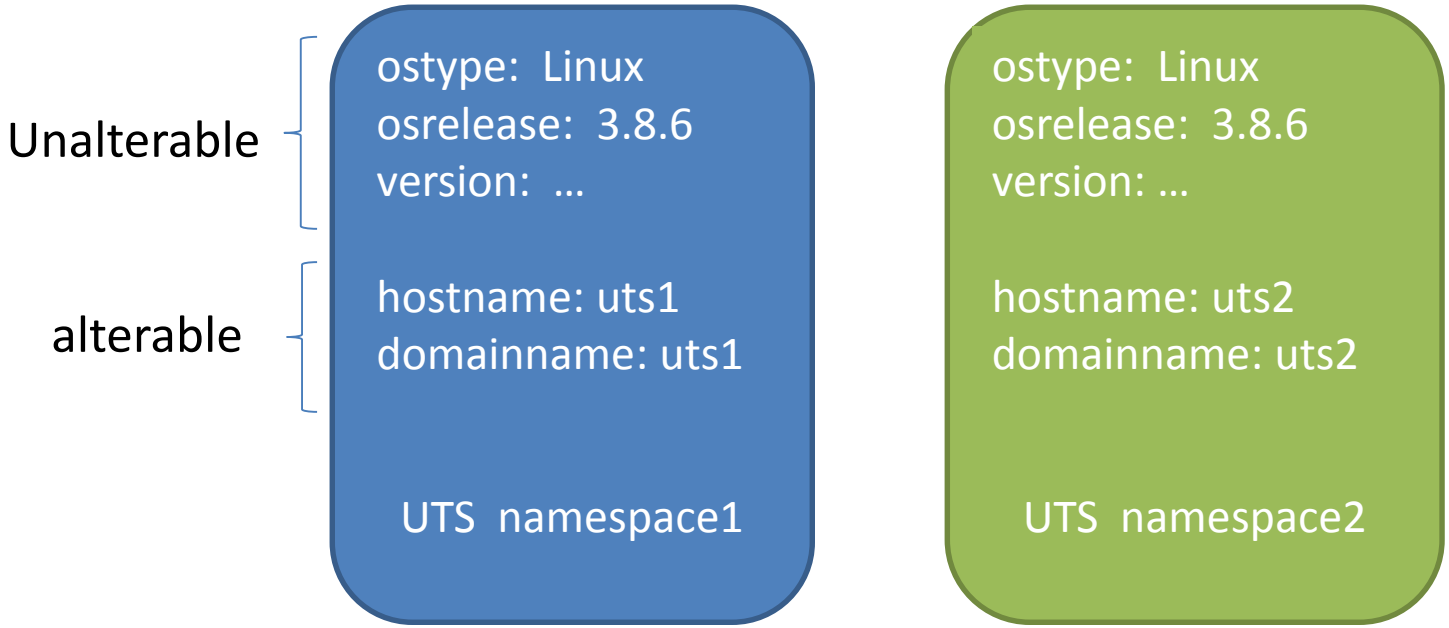
Mount Namespace

- Each mount namespace has its own filesystem layout.



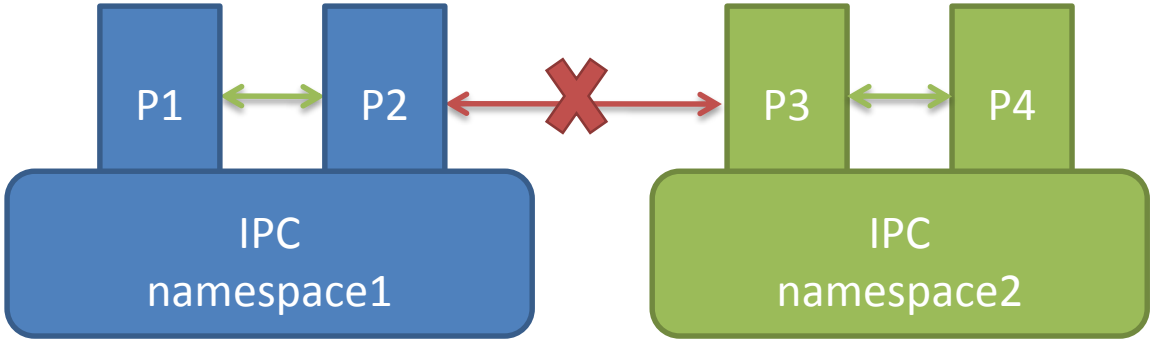
UTS Namespace

- Every uts namespace has its own uts related information.



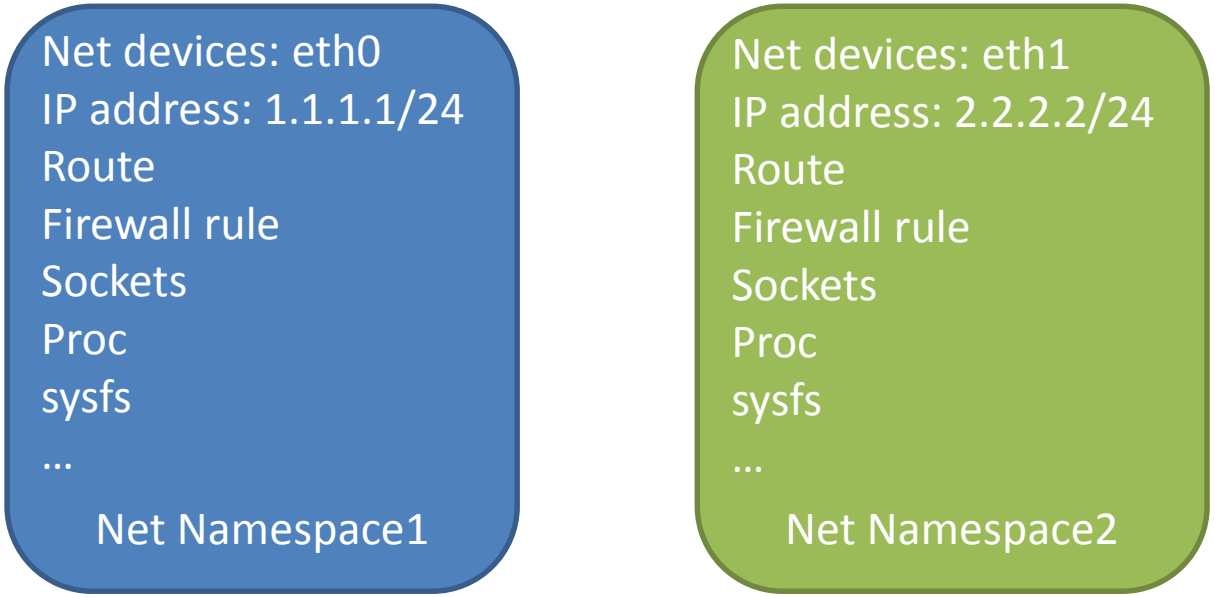
IPC Namespace

- IPC namespce isolates the interprocess communication resource(shared memory, semaphore, message queue)



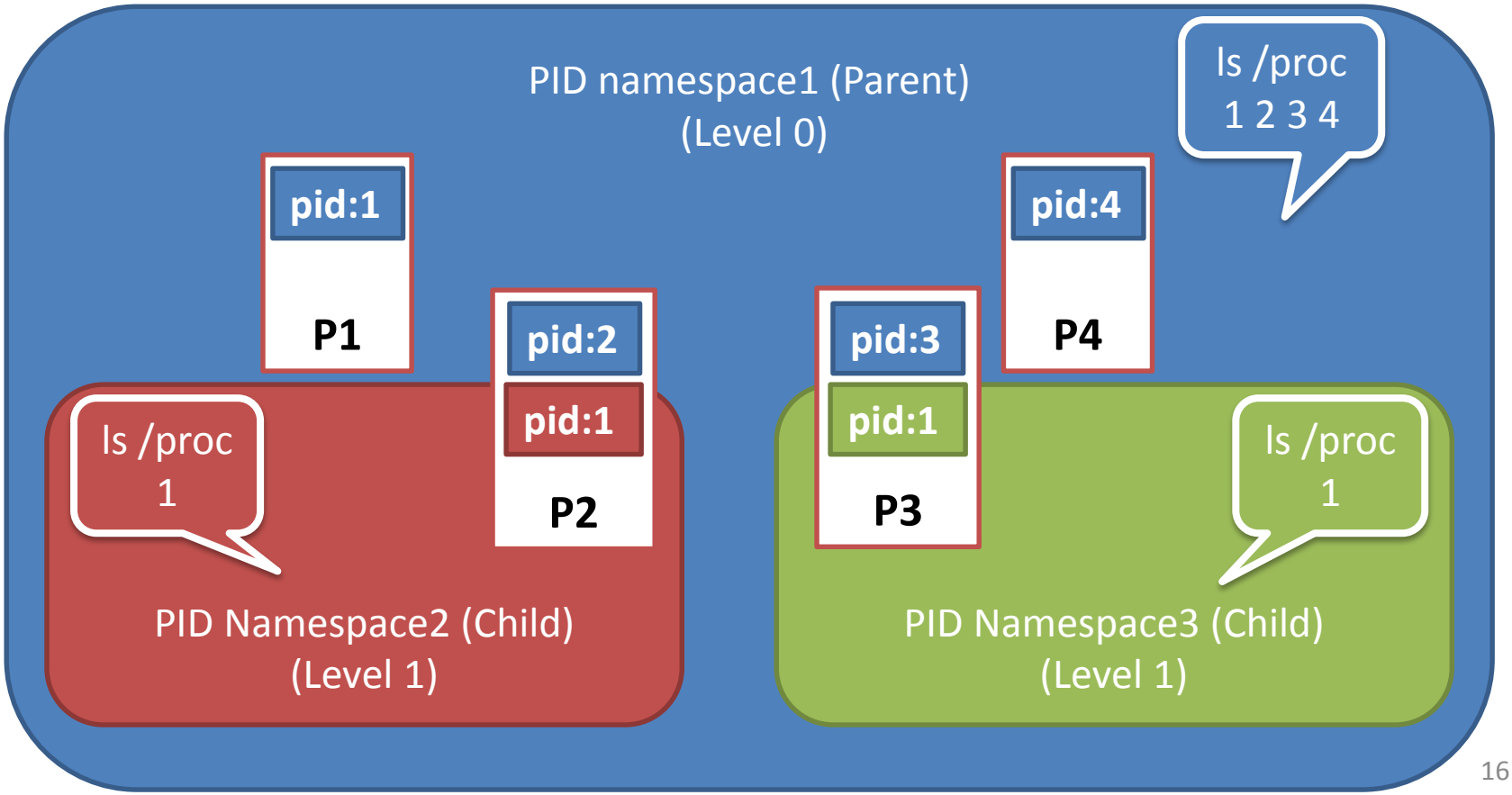
Net Namespace

- Net namespace isolates the networking related resources



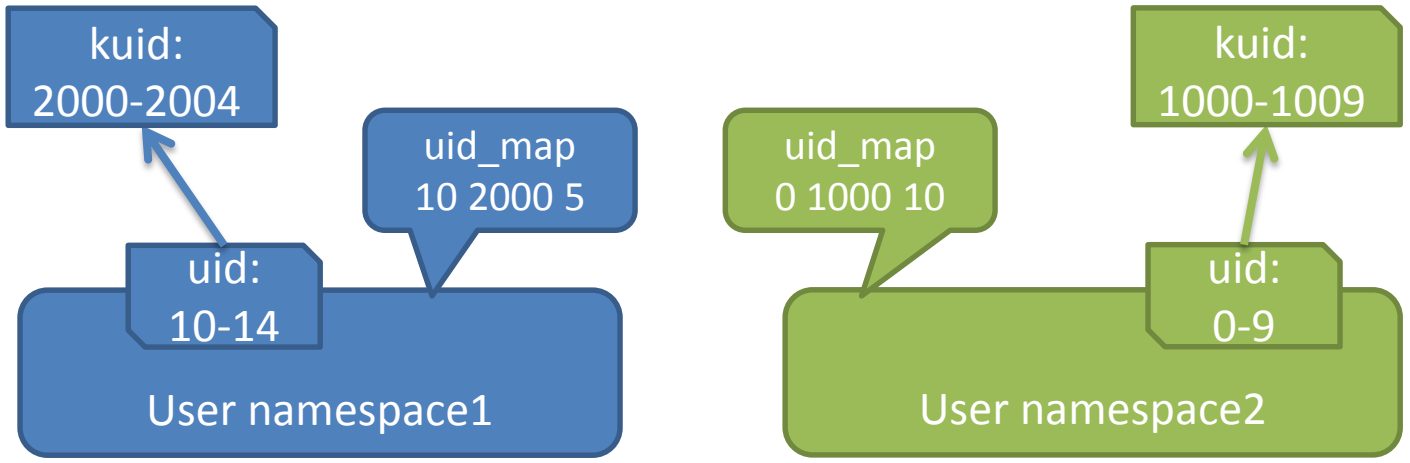
PID Namespace

- PID namespace isolates the Process ID, implemented as a hierarchy.



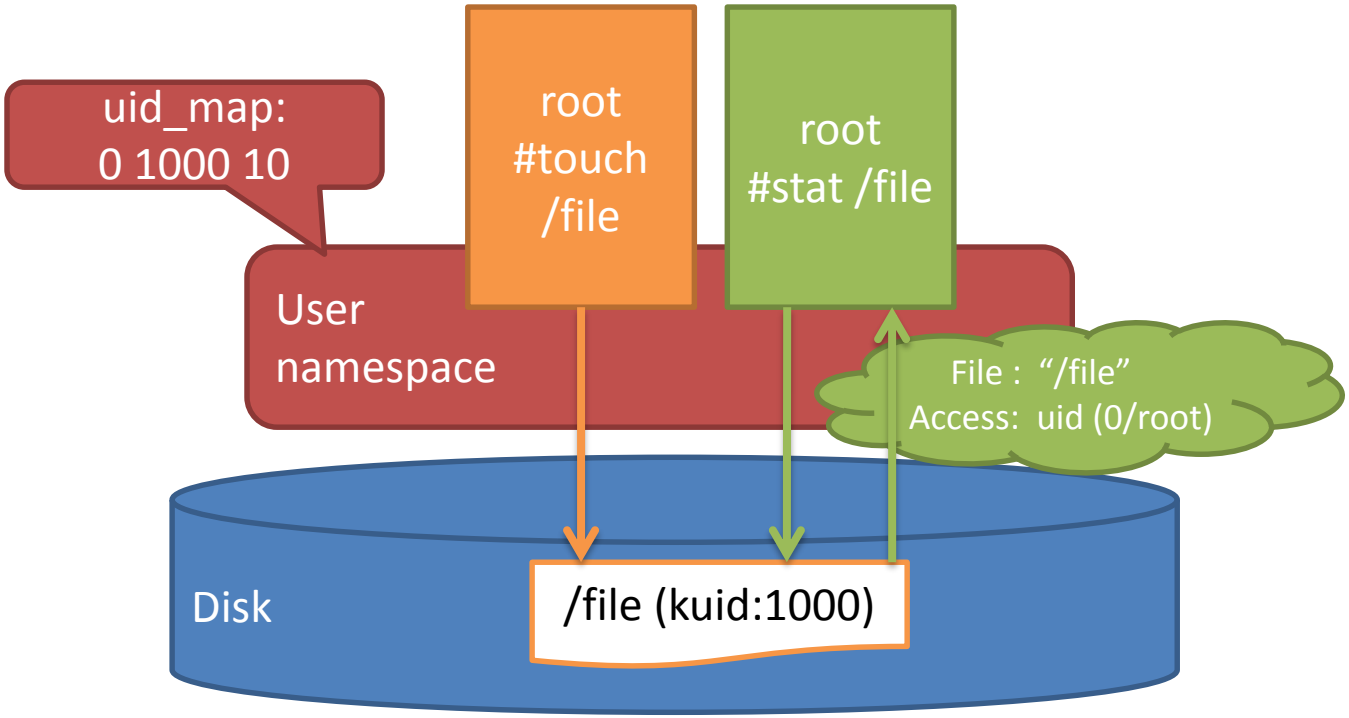
User Namespace

- kuid/kgid: Original uid/gid, Global
- uid/gid: user id in user namespace, will be translated to kuid/kgid finally
- Only parent User NS has rights to set map



User Namespace

- Create and stat file in User namesapce



LXC

System API/ABI

- Proc
  - /proc/<pid>/ns/
- System Call
  - clone
  - unshare
  - setns

Proc

- /proc/<pid>/ns/ipc: ipc namespace
- /proc/<pid>/ns/mnt: mount namespace
- /proc/<pid>/ns/net: net namespace
- /proc/<pid>/ns/pid: pid namespace
- /proc/<pid>/ns/uts: uts namespace
- /proc/<pid>/ns/user: user namespace

- If the proc file of two processes is the same, these two processes must be in the same namespace.

System Call

- clone

```
int clone(int (*fn)(void *), void *child_stack,
        int flags, void *arg, …);
```

6 new flags:

CLONE_NEWIPC,CLONE_NEWNET,

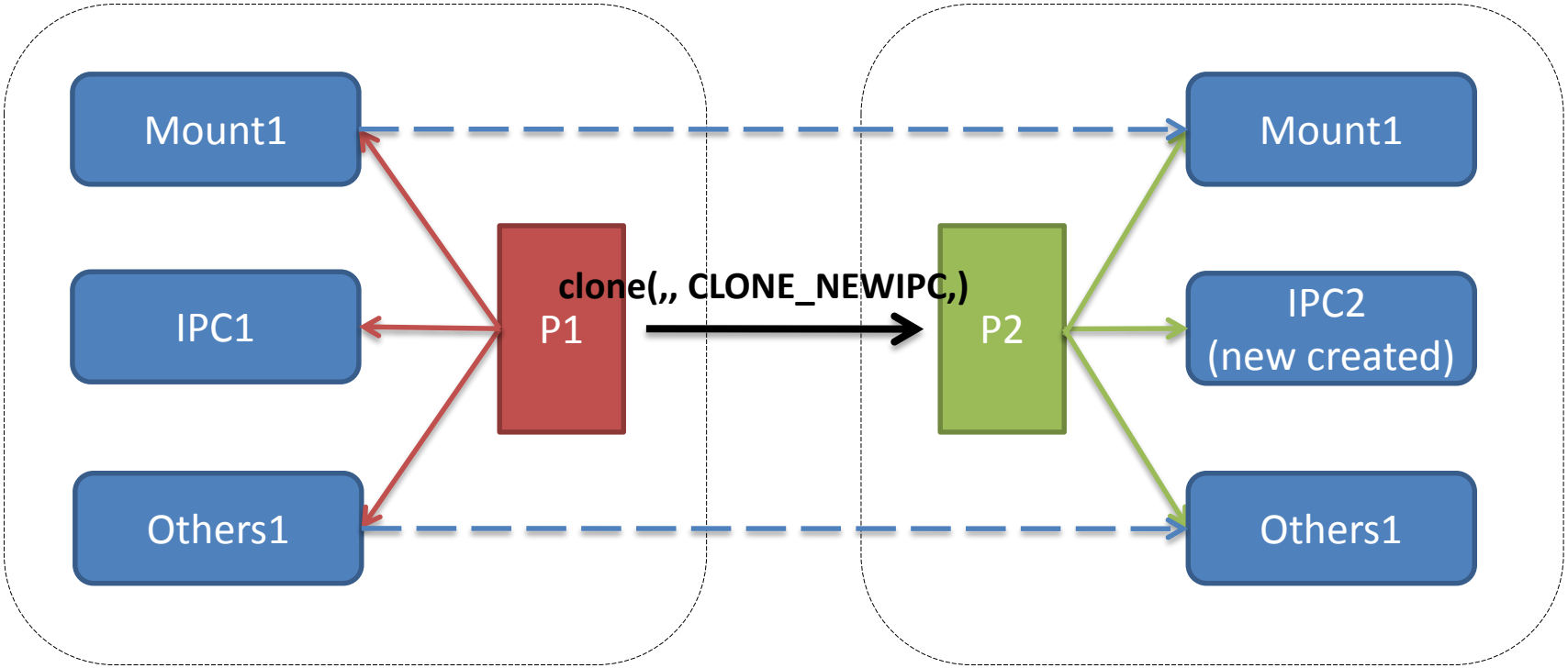
CLONE_NEWNS,CLONE_NEWPID,

CLONE_NEWUTS,CLONE_NEWUSER

System Call

- clone

  create process2 and IPC namespace2



System Call

- unshare

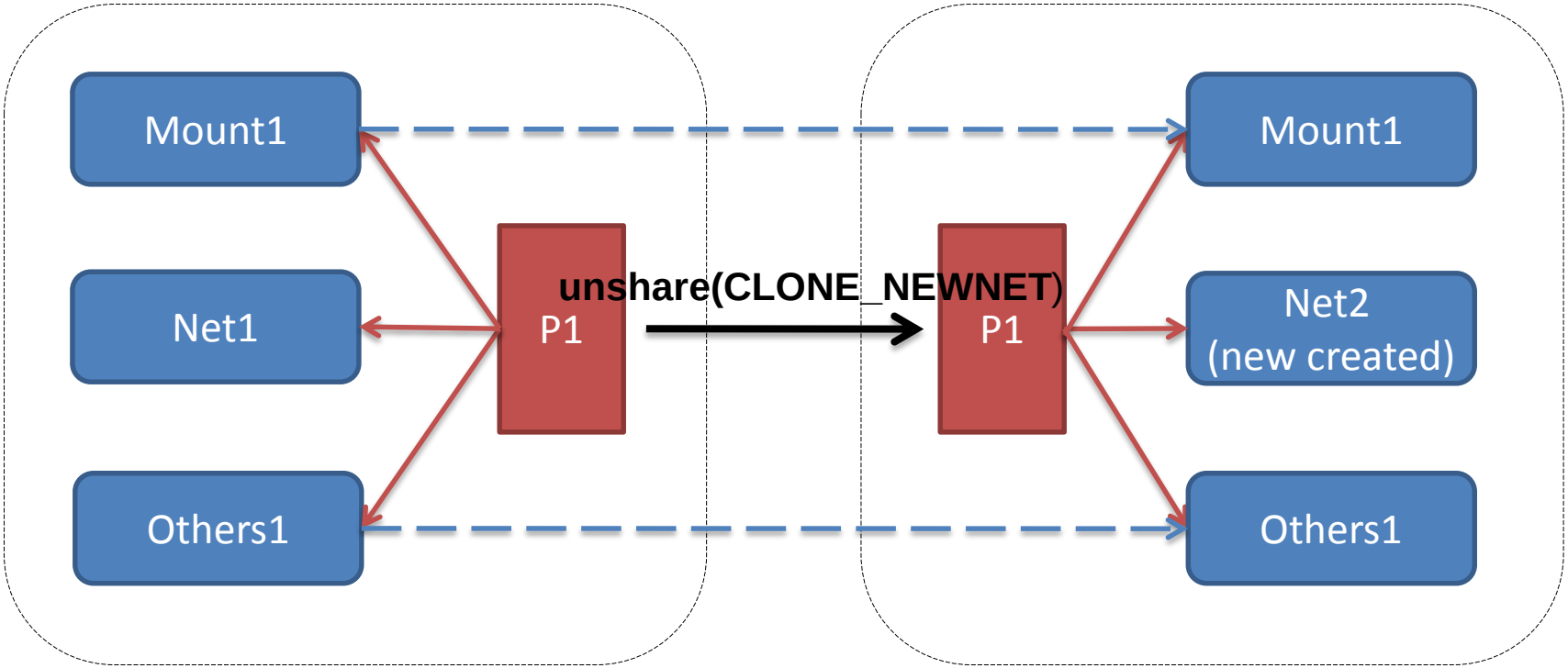
  int unshare(int flags);

  Namespace extends the system call unshare too. User space can use unshare to create new namespace and the caller will run in this new created namespace.

System Call

- unshare

  create net namespace2



System Call

- setns

  int setns(int fd, int nstype);

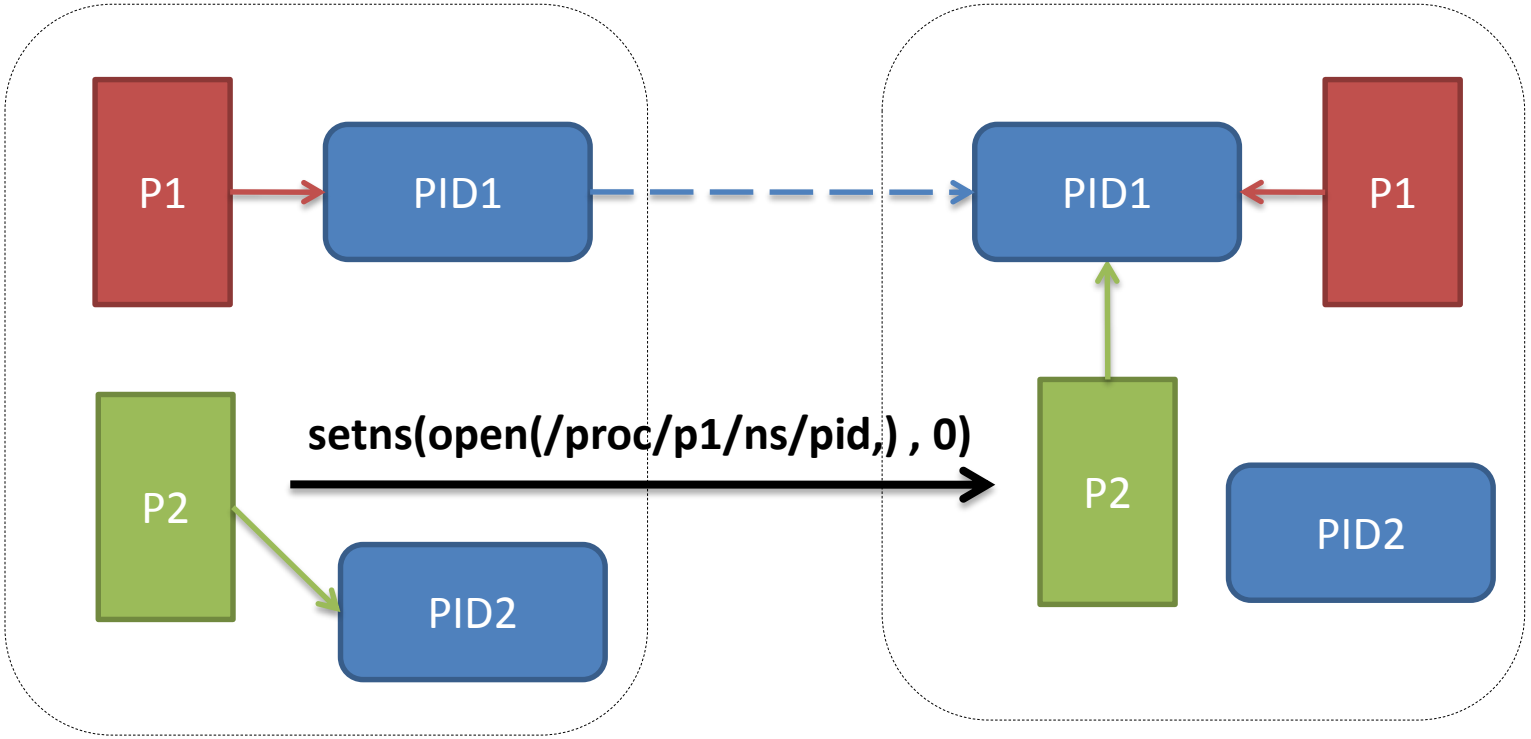
  setns is a new added system call for namespace. Process can use setns to set which namespace the process will belong to.

  @fd: file descriptor of namespace(/proc/<pid>/ns/*)
  @nstype: type of namespace.

System Call

- setns

  Change the PID namespace of P2



Libvirt LXC

- Libvirt LXC: userspace container management tool, Implemented as one type of libvirt driver.
  - Manage containers
  - Create namespace
  - Create private filesystem layout for container
  - Create devices for container
  - Resources controller by cgroup

Comparison

- The feature that host share the same kernel with guest makes container different from other virtualization method

	Container	KVM
performance	Great	Normal
OS support	Linux Only	No Limit
Security	Normal	Great
Completeness	Low	Great

Problems

- /proc/meminfo, cpuinfo…
  - Kernel space (relate to cgroup)
  - User space (poor efficiency)
- New namespace
  - Audit (assign to user namespace?)
  - Syslog (do we really need it?)

Problems

- Bandwidth control
  - TC Qdisc
    - On host (How to handle setting nic to container?)
    - On container (user can change it)
  - Netfilter
    - How to control Ingress bandwidth
- Disk quota
  - Uid/Gid Quota (Many users)
  - Project Quota (xfs only)

DOCKER

Docker workflow 1/2

- Work in dev environment (local machine or container)
- Other services (databases etc.) in containers
  - and behave just like the real thing!
- Whenever you want to test ≪ for real ≫:
  - Build in *seconds*
  - Run *instantly*

Docker workflow 2/2

- Satisfied with your local build?
  - Push it to a *registry* (public or private)
  - Run it (automatically!) in CI/CD
  - Run it in production
  - Happiness!
- Something goes wrong? Rollback painlessly!

Authoring img.s (w run/commit)

- 1) docker run ubuntu bash
- 2) apt-get install this and that
- 3) docker commit <containerid> <imagename>
- 4) docker run <imagename> bash
- 5) git clone git://.../mycode
- 6) pip install -r requirements.txt
- 7) docker commit <containerid> <imagename>
- 8) repeat steps 4-7 as necessary
- 9) docker tag <imagename> <user/image>
- 10) docker push <user/image>

Pros and Cons

- Pros
  - Convenient, nothing to learn
  - Can roll back/forward if needed
- Cons
  - Manual process
  - Iterative changes stack up
  - Full rebuilds are boring, error-prone

Authoring img.s (w Docker)

- RUN apt-get -y update
- RUN apt-get install -y g++
- RUN apt-get install -y erlang-dev erlang-manpages erlang-base-hipe ...
- RUN apt-get install -y libmozjs185-dev libicu-dev libtool ...
- RUN apt-get install -y make wget
- RUN wget http://.../apache-couchdb-1.3.1.tar.gz | tar -C /tmp -zxf-
- RUN cd /tmp/apache-couchdb-* && ./configure && make install
- RUN printf "[httpd]\nport = 8101\nbind_address = 0.0.0.0" >
- /usr/local/etc/couchdb/local.d/docker.ini

EXPOSE 8101

CMD ["/usr/local/bin/couchdb"]

docker build -t author_name/couchdb

Pros

- Minimal learning curve
- Rebuilds are easy
- Caching system makes rebuilds faster
- Single file to define the whole environment!

Docker

- Multi-arch, multi-OS
- Stable control API
- Stable plugin API
- Resiliency
- Signature
- Clustering

Docker advantages

- Docker:
- Is easy to install
- Will run anything, anywhere
- Gives you repeatable builds
- Enables better CI/CD workflows
- Is backed by a strong community
- Will change how we build and ship software

Summary

- Linux containers
  - LXC, Docker
- Lightweight virtualization
  - performance
- Easy software handling
- Needs orchestration