

OS-level virtualization (Containers)

Kartik Gopalan

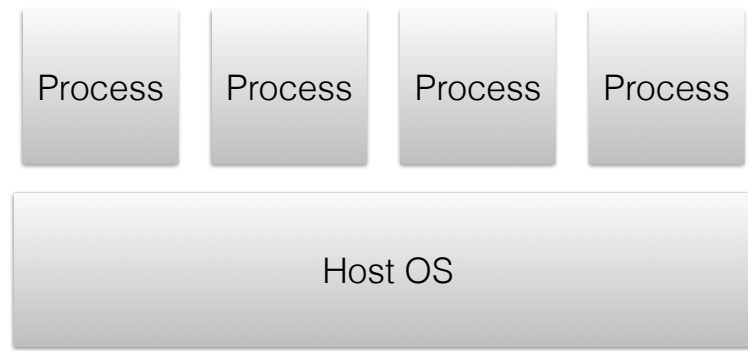
Containers

- A lightweight virtualization technology that packages software and its dependencies into an isolated environment
- Used to deploy and run applications more conveniently and efficiently than traditional virtual machines.
- There are many different container platforms available, such as Docker, Kubernetes, and OpenShift.

Isolation Models

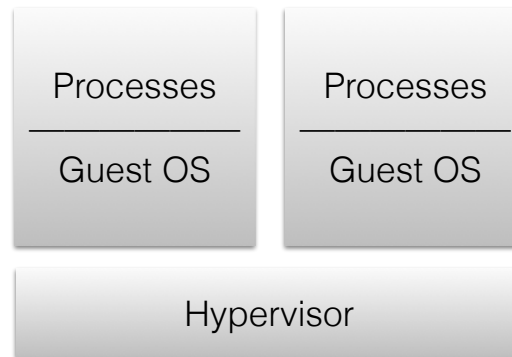
- Isolation means
 - Limiting what/who a process/application can see.
 - Limiting who can see a process/application
- Traditionally there were two extremes of isolation models
 - Traditional Process
 - System Virtual Machines

Traditional Processes



- Each process gets its own
 - Virtual memory
 - One or more virtual CPUs (threads)
 - Access to OS services via system calls
- All co-located processes can see/share a lot (in an OS-controlled manner)
 - File system, storage, network, and I/O devices
 - Other processes (for Inter-process communication)

System Virtual Machines

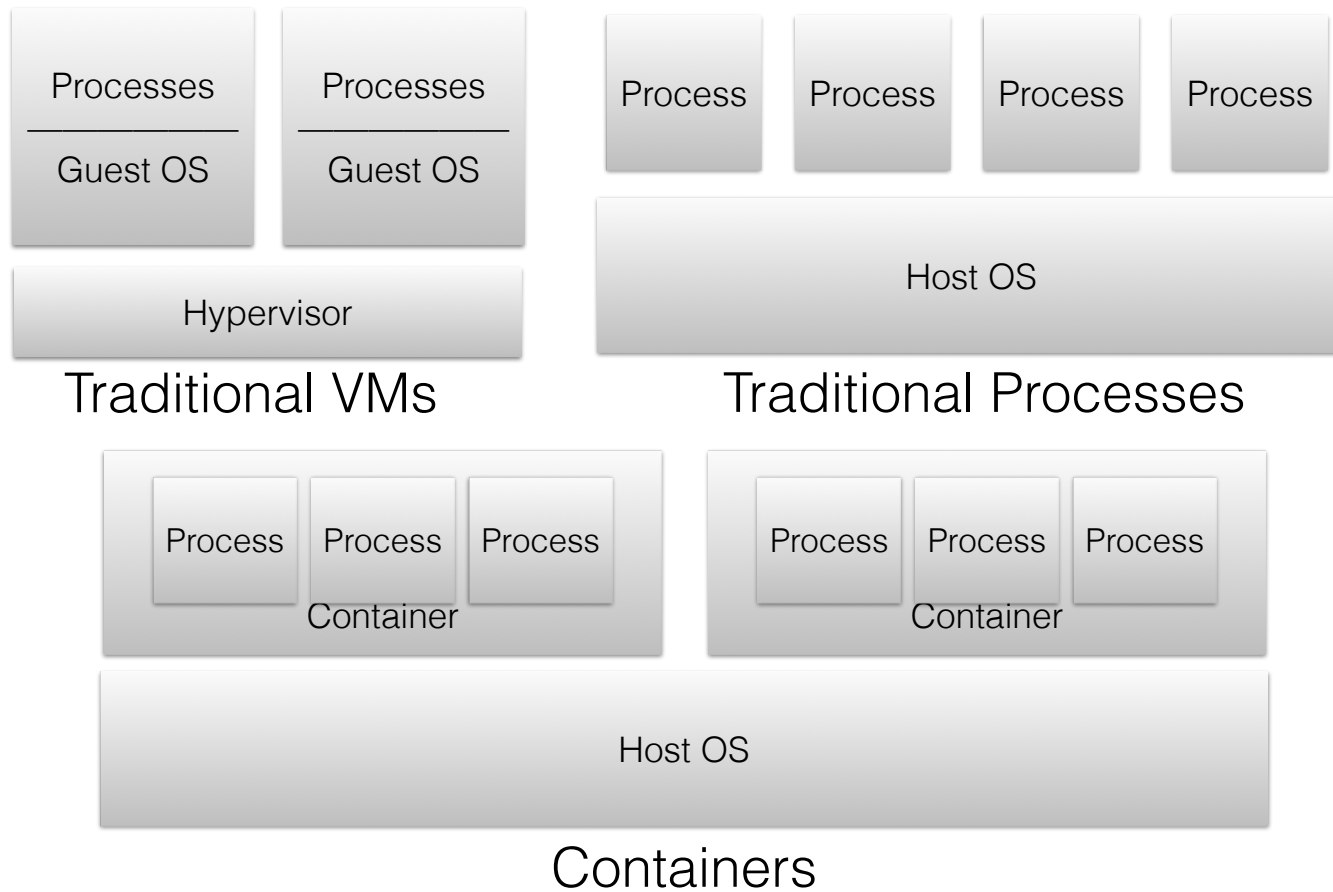


- Co-related processes grouped into VMs
- Each VM has its own
 - Guest OS
 - Guest physical memory (“virtualized” view of memory seen by guest OS)
 - One or more virtual CPUs
 - Virtual I/O devices: virtual disk, virtual network
- Ideally: Co-located VMs don’t see/share ANYTHING

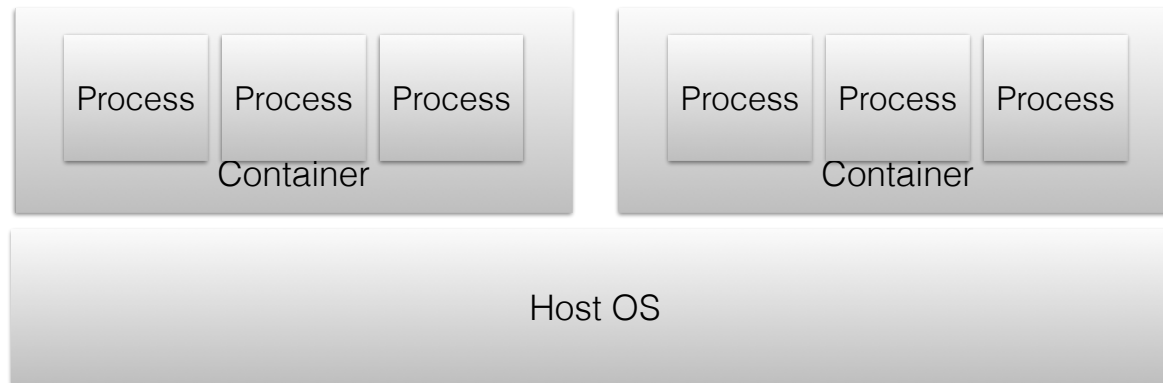
What level to isolate?

- Processes share too much
 - Great performance but not isolated enough
- System VMs are too heavy
 - Great Isolation but too heavy due to separate guest OS per VM
- **Alternative: Operating-system-level virtualization**
 - Multiple isolated user-spaces
 - Share one kernel.
 - Native performance

Process, VM, Container



Containers



- Containers
 - group traditional processes together and
 - restrict what resources they can see/access.
- In Linux, containers consist of
 - Namespaces
 - Control Groups (cgroups)

Chroot

- An early precursor to modern namespaces
- Change root directory for the calling process and its children to a given path
- `$ chroot NEWROOT`

OR

- `$ chroot(path)`
- “This call changes an ingredient in the pathname resolution process and does nothing else.” — `man chroot`
- Not secure. Lots of ways to escape chroot jail.

FreeBSD Jails

- Builds upon chroot to compartmentalize files and other resources
- Jails protects rest of the system from the jailed process
 - Not the other way around!
- Virtualized resources
 - file system,
 - the set of users, including own root account.
 - networking subsystem
- Again: Jail escapes were possible!

Linux Namespaces

- Linux namespaces are a feature of the Linux kernel that partitions kernel resources such as processes, networking, and file systems.
 - “A ***namespace*** wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource.” - from “`man namespaces`”
- This allows different processes to run in isolated environments, each with its own view of the system resources.
- Namespaces are used to implement lightweight virtualization technologies such as containers.

Types of Linux Namespaces

| Namespace | Flag | Page | Isolates |
|-----------|------------------------|------------------------------|--------------------------------------|
| Cgroup | CLONE_NEWCGROUP | cgroup_namespaces(7) | Cgroup root directory |
| IPC | CLONE_NEWIPC | ipc_namespaces(7) | System V IPC, POSIX message queues |
| Network | CLONE_NEWNET | network_namespaces(7) | Network devices, stacks, ports, etc. |
| Mount | CLONE_NEWNS | mount_namespaces(7) | Mount points |
| PID | CLONE_NEWPID | pid_namespaces(7) | Process IDs |
| Time | CLONE_NEWTIME | time_namespaces(7) | Boot and monotonic clocks |
| User | CLONE_NEWUSER | user_namespaces(7) | User and group IDs |
| UTS | CLONE_NEWUTS | uts_namespaces(7) | Hostname and NIS domain name |

PID Namespace

- Isolates the set of processes that can see each other.
- PID namespaces isolate the process ID number space, meaning that processes in different PID namespaces can have the same PID.
- PID namespaces allow containers to provide functionality such as
 - suspending/resuming the set of processes in the container
 - migrating the container to a new host while the processes inside the container maintain the same PIDs.
- PIDs in a new PID namespace start at 1, somewhat like a standalone system
- Calls to `fork(2)`, `vfork(2)`, or `clone(2)` will produce processes with PIDs that are unique within the namespace.

IPC Namespace

- Isolates which processes are allowed to communicate with each other
- IPC namespaces isolate three types of IPC objects
 - message queues, semaphores, and shared memory
- The common characteristic of these IPC mechanisms is that IPC objects are identified by mechanisms other than filesystem pathnames.
- Each IPC namespace has its own set of IPC identifiers.
- Objects created in an IPC namespace are visible to all other processes that are members of that namespace, but are not visible to processes in other IPC namespaces.

Mount Namespace

- Isolates which part of filesystem is seen by a process group
- Provide isolation of the list of mount points seen by the processes in each namespace.
- The processes in each of the mount namespace instances will see distinct single-directory hierarchies.
- A new mount namespace is created using either `clone(2)` or `unshare(2)` with the `CLONE_NEWNS` flag.
 - If the namespace is created using `clone(2)`, the mount list of the child's namespace is a copy of the mount list in the parent process's mount namespace.
 - If the namespace is created using `unshare(2)`, the mount list of the new namespace is a copy of the mount list in the caller's previous mount namespace.

Network Namespace

- Network namespaces provide isolation of the system resources associated with networking.
 - IP address
 - network devices
 - IPv4 and IPv6 protocol stacks
 - IP routing tables
 - firewall rules
 - associated configuration files.

User Namespace

- User namespaces isolate security-related identifiers and attributes
 - User IDs and Group IDs, the root directory, keys, and capabilities.
- A process's user and group IDs can be different inside and outside a user namespace.
- Process can have a normal unprivileged user ID outside a user namespace while at the same time having a user ID of 0 inside the namespace.
- In other words, the process has full privileges for operations inside the user namespace, but is unprivileged for operations outside the namespace.

UTS Namespace

- UTS namespaces provide isolation of two system identifiers
 - the hostname and the NIS domain name.
- These identifiers are set using `sethostname(2)` and `setdomainname(2)`, and can be retrieved using `uname(2)`, `gethostname(2)`, and `getdomainname(2)`.
- When a process creates a new UTS namespace using `clone(2)` or `unshare(2)` with the `CLONE_NEWUTS` flag, the hostname and domain name of the new UTS namespace are copied from the corresponding values in the caller's UTS namespace.

Time Namespace

- Time namespaces virtualize the values of system clocks
- The processes in a time namespace share per-namespace values for system clocks.
- This affects various APIs that measure against these clocks, including: `clock_gettime(2)`, `clock_nanosleep(2)`, `nanosleep(2)`, `timer_settime(2)`, `timerfd_settime(2)`, and `/proc/uptime`.

Cgroups (Control Groups)

- Linux kernel feature which allow processes to be organized into hierarchical groups whose usage of various types of resources can then be limited and monitored.
- Allows administrator to set soft/hard limits on cgroup-wide usage of
 - memory, network bandwidth, CPU etc.
- “Cgroup namespaces” virtualize the view of a process's cgroups as seen via `/proc/pid/cgroup` and `/proc/pid/mountinfo`.
- The kernel's cgroup interface is provided through a pseudo-filesystem called `cgroupfs`.

Single System Image

- Extend the notion of namespaces to multiple physical machines
- Multiple machines look like one (or more) namespace(s)
 - PID namespace
 - IPC Namespace
 - Filesystem namespace
- Process migration
 - Allows moving processes from one machine to another without changing its namespace.
- Examples: MOSIX, OpenSSI, Kerrighed