Container-Based Virtualization

Advanced Operating Systems

Luca Abeni

luca.abeni@santannapisa.it

Virtualized Resources

- Virtual Machine: efficient, isolated duplicate of a physical machine
  - Why focusing on *physical* machines?
  - What about abstract machines?
- Software stack: hierarchy of abstract machines
  - ...
  - Abstract machine: language runtime
  - Abstract machine: OS (hardware + system library calls)
  - Abstract machine: OS kernel (hardware + syscalls)
  - Physical machine (hardware)

Hardware Virtualization

- Can be full hardware virtualization or paravirtualization
  - Paravirtualization requires modifications to guest OS (kernel)
- Can be based on trap and emulate
- Can use special CPU features (hardware assisted virtualization)
- In any case, the hardware (whole machine) is virtualized!
  - Guests can provide their own OS kernel
  - Guests can execute at various privilege levels

OS-Level Virtualization

- The OS kernel (or the whole OS) is virtualized
  - Guests can provide the user-space part of the OS (system libraries + binaries, boot scripts, ...) or just an application...
  - ...But continue to use the host OS kernel!
- One single OS kernel (the host kernel) in the system
  - The kernel virtualizes all (or part) of its services
- OS kernel virtualization: container-based virtualization
- Example of OS virtualization: wine

Virtualization at Language Level

- The language runtime is virtualized
  - Often used to achieve independence from hardware architecture
- Example: Java Virtual Machine
- Often implemented by using emulation techniques
  - Interpreter or just-in-time compiler

OS-Level Virtual Machines

- Do not virtualise the whole hardware
  - Only OS services are virtualised
  - Host kernel: virtualise its services to provide isolation among guests
- Container: isolated execution environment to encapsulate one or more processes/tasks
  - Sort of “chroot on steroids”
- Two aspects: resource control (scheduling) and visibility
  - Control/monitor how much resources a VM is using
  - Make sure that virtual resources of a VM are not visible in other VMs

More on “Containers”

- Banga and others, 99: “Resource Containers: A New Facility for Resource Management in Server Systems”
  - Operating system abstraction containing all the resources used by an application to achieve a particular independent activity
- Today, “container” == execution environment
  - Used to run a whole OS $\rightarrow$ VM (with OS-level virtualization)
  - Used to run a single application / micro-service

Linux Containers

- The Linux kernel does not directly provide the “container” abstraction
- Containers can be built based on lower-level mechanisms: *control groups* (`cgroups`) and *namespaces*
  - namespaces: isolate and virtualise system resources
  - cgroups: limit, control, or monitor resources used by groups of tasks
- Namespaces are concerned with resources’ visibility, cgroups are concerned with scheduling

Linux Namespaces

- Used to isolate and virtualise system resources
  - Processes executing in a namespace have the illusion to use a dedicated copy of the namespace resources
  - Processes in a namespace cannot use (or even see) resources outside of the namespace
- Processes in a network namespace only see network interfaces that are assigned to the namespace
  - Same for routing table, etc...
- Processes in a PID namespace only see processes from the same namespace
  - PIDs can be“private to the namespace”

Linux Control Groups

- Used to restrict (limit, control) or monitor the amount of resources used by “groups of processes”
  - Processes can be organized in groups, to control their accesses to resources
- Example: CPU control groups for scheduling
  - Limit the amount of CPU time that processes can use, etc...
- Similar cgroups for other resources
  - memory, IO, pids, network, ...

Building a Container

- Namespaces and control group give fine-grained control on processes and resources
  - Per-resource control groups and/or namespaces
  - Lower level abstractions respect to other OSs (for example, FreeBSD jails)
- More powerful than other mechanisms, but more difficult to use
- To build a container, it is necessary to:
  - Setup all the needed namespaces and control groups
  - Create a “disk image” for the container (directory containing the container’s fs)

Running in a Container

- Chroot to the container fs
  - Must contain the whole OS, or the libraries/files needed to execute the program to containerize
- Start init, or the program to containerize
  - Thanks to the PID namespace, it will have PID 1 in the container!
- Note: init can mount procfs or other pseudo-filesystems
  - Namespaces allow to control the information exported in those pseudofilesystems!

Example: Networking in Containers

- Thanks to the network namespace, processes running in a container do not see the host's network interfaces
  - How to do networking, then?
- Create a *virtual ethernet pair*
  - Two virtual ethernet interfaces, connected point-to-point
  - Packets sent on one interface are received on the other, and vice-versa
- Associate one of the two virtual ethernet interfaces to the network namespace of the container
- Bind the other one to a software bridge

User-Space Tools

- Building and running a container can be difficult...
  - But users do not have to do it “by hand”!!!
- User-space tools for building containers and deploying OSs/applications in them
  - Simplest tool: `lxc` (`http://linuxcontainers.org`)
  - Server-based version of `lxc`: `lxd`
  - Docker: more advanced features
  - Kubernetes
  - ...
- Recent proliferation of tools, all with different interfaces/features

lxc / lxd

- `lxc`: set of tools and libraries that allow to easily use containers, namespaces and friends
  - Focus on installing and running Linux distributions in containers
- Need root privileges, at least partly
- `lxd`: daemon running with root privileges and using the `lxc` library
  - Clients can connect to it through a socket to request operations on containers
  - More secure, because user tools do not need to be privileged (the only privileged component is the daemon)

More Advanced Tools

- Docker, Kubernetes and similar allow to also containerize single applications
  - Container with application binary, libraries, needed files, etc...
  - Useful for distributing consistent execution environments
- More advanced tools respect to `lxc`/`lxd`
- Also provide “container images” distributed with custom image formats
- Lot of different solutions with different features, interfaces, etc...

Standardizing the Container Tools

- Open Container Initiative (OCI): `https://www.opencontainers.org/`
  - Tries to define standards for the user-space tools
  - Currently, two standards: runtime specification and image specification
- Runtime specification: standardizes the configuration, execution environment, and lifecycle of a container
  - A “filesystem bundle” described according to this specification can be started in a container by any compliant runtime
- Image specification: standardizes how the content of a container is represented in binary form

OCI's Goals

- Define containers in a "technology neutral" way
- Container: encapsulates a software component and all its dependencies
  - Using a format that is self-describing and portable
  - Any compliant "runtime" must be able to run it without extra dependencies
- This must work regardless of the implementation details
  - Underlying machine, containerization technology, contents of the container, ...

OCI Runtime Specification

- Standardizes important aspects of containers
  - Configuration: specified through a standardized `config.json`, describing all the details of the container
  - Execution environment: standardized so that applications running in containers see a consistent environment between runtimes
  - Standard operations possible during the containers' lifecycles
- If a "runtime" is compliant with these specifications, the implementation details do not matter

More than Containers

- Looking at the OCI definitions, there is not mention to OS-level virtualization anymore...
  - The terms “container” and “containerized application” are evolving...
- “container” is just a synonim for “lightweight virtual machine”, independently from the used technology
  - Kata containers: use kvm-based VMs (qemu/nemu) instead of namespaces and cgrouops
  - Compliant with the OCI runtime specification
- Thanks to OCI, it is possible to *almost* transparently replace the runtime/containerization mechanism without changing userspace tools!