Containers, Orchestration, Network Virtualization Intro

Lecture 6

Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S23

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Building Blocks of Containers

What goes into a container?

- Not a natively hardware-supported abstraction like privilege rings, which enable OSes (and virtual machines)
- Instead, use software mechanisms built into OS kernels
- Containers: a loose conglomeration of kernel-level mechanisms
  - Access isolation of global resources (namespaces)
  - Resource/Performance isolation of global resources (control groups)
  - Sharing data on filesystem for efficiency (union filesystems)
    - (need to add on isolation of unique data)
  - Security mechanisms: appArmor, capabilities
- Kludgy, but essential since hard to get it right from scratch

Namespaces

- Access isolation
- Show an instance of a global resource as available to all processes inside a namespace
- Changes visible to other processes within namespace
  - But invisible outside the namespace
- Show different “copies” of resources associated with the kind of namespace
  - Network, IPC, mount, PID, …
- Every process starts in init namespace, change with `setns`
- Network: (software/hardware) network device; routing rules; port numbers. `veth` pair connects two network namespaces

Control groups

- Resource/Performance isolation
- Subsystem: a specific kind of resource
  - CPU time, memory, network bandwidth, block device access, priority, CPU and memory (numa) node assignment. Many configurable parameters per subsystem
- Control group or cgroup: a set of processes
  - If fork(), inherit a bunch of attributes including parent's cgroup
- Hierarchy: a tree where each node is a cgroup
  - Many hierarchies can exist, unlike process hierarchy
- Each subsystem "mounted" onto one hierarchy
  - Possible to use a single hierarchy for multiple subsystems (resources)
- Every process has exactly one reservation per resource

Union FS: ~~“container images too big”~~

- Directory structures on disk are typically “mounted” at some point in the virtual filesystem (/, /home/users/name, etc.)
- Processes in containers want mostly the same files, with a small number of modifications per process or container
  - Think: common third-party packages and shared library images
  - (while supporting the need for distinct libraries/versions across containers)
  - Similar use cases in the past: data on a read-only medium which needed a small number of updates and refresh into new medium
- Union filesystem: maintain a stack of filesystems at each mount point. Only the latest one is writable. Lower layers are read-only.
- Write fresh to the top. Update by copy up. Deletion requires a special mechanism to record a file that isn’t there (whiteout). Cache heavily.
- Virtual Filesystem layer accomplishes this with minimal changes to underlying filesystem.

Orchestrating Containers

Components you need?

- The machines (nodes), pods (container-ish), images
- Controllers and mechanization (“choreography”)
  - Provisioning pods and nodes with desired resources (kube-scheduler + kubelet)
  - Replicating according to system metrics or demand (autoscaling controller)
  - Detecting and reacting to failures (replicaSet controller)
- Maintaining the cluster’s desired and observed state
  - Persistent data store (etcd; consensus protocol -- RAFT)
  - How should everyone see and access this? api server (versioning, etc.)
- Desired state: declaratively specified. Label selectors to group.
  - … even when we say kubectl do this and that
- Naming and connecting to remote entities
  - Pods shouldn’t have to know physical addresses; IP address management for applications connecting from within container network namespaces
  - Routing between nodes; within a node from/to a pod on the node
  - Container Network Interface

Network Virtualization

Virtualizing Networking on a Single Machine

How to virtualize I/O?

- How device I/O works in general:
  - Registers. Interrupts and polling. Shared memory. DMA.
- Full virtualization: trap and emulate any I/O data operation
  - e.g., moving each byte of guest data through VMM memory is too expensive (not a “zero copy” solution)
- Xen’s initial approach (SOSP’03)
  - Descriptor rings: async I/O over memory shared between hypervisor & guest
- Hypervisor responsibilities for virtualization:
  - Validate data pages pointed from guest-enqueued descriptors
  - Remap data pages (avoid time-of-check to time-of-use), even if not copy
  - (incoming) find which VM to signal? Send event notification to guest OS
- Hypervisor intervention to check every descriptor is bad for perf