OS Virtualization

- Emulate OS-level interface with native interface
- “Lightweight” virtual machines
  - No hypervisor, OS provides necessary support
  
  ![Diagram of virtual machines and containers]

- Solaris containers, BSD jails, Linux containers

Linux Containers (LXC)

- Containers share OS kernel of the host
  - OS provides resource isolation
- Benefits
  - Fast provisioning, bare-metal like performance, lightweight
OS Mechanisms for LXC

- OS mechanisms for resource isolation and management
- namespaces: process-based resource isolation
- Cgroups: limits, prioritization, accounting, control
- chroot: apparent root directory
- Linux security module, access control
- Tools (e.g., docker) for easy management

Linux Namespaces

- Namespace: restrict what can a container see?
  - Provide process level isolation of global resources
- Processes have illusion they are the only processes in the system
- MNT: mount points, file systems (what files, dir are visible)?
- PID: what other processes are visible?
- NET: NICs, routing
- Users: what uid, gid are visible?
- chroot: change root directory
Linux cgroups

• Resource isolation
  – what and how much can a container use?
    • Set upper bounds (limits) on resources that can be used
    • Fair sharing of certain resources

• Examples:
  – cpu: weighted proportional share of CPU for a group
  – cpuset: cores that a group can access
  – block io: weighted proportional block IO access
  – memory: max memory limit for a group

Proportional Share Scheduling

– Uses a variant of proportional-share scheduling

• Share-based scheduling:
  – Assign each process a weight \( w_i \) (a “share”)
  – Allocation is in proportional to share
  – fairness: reused unused cycles to others in proportion to weight
  – Examples: fair queuing, start time fair queuing

• Hard limits: assign upper bounds (e.g., 30%), no reallocation

• Credit-based: allocate credits every time \( T \), can accumulate credits, and can burst up-to credit limit
  – can a process starve other processes?
Putting it all together

- Images: files/data for a container
  - can run different distributions/apps on a host
- Linux security modules and access control
- Linux capabilities: per process privileges
Docker and Linux Containers

- Linux containers are a set of kernel features
  - Need user space tools to manage containers
  - Virtuozo, OpenVZm, VServer, Lxc-tools, Docker
- What does Docker add to Linux containers?
  - Portable container deployment across machines
  - Application-centric: geared for app deployment
  - Automatic builds: create containers from build files
  - Component re-use
- Docker containers are self-contained: no dependencies

Docker

- Docker uses Linux containers
LXC Virtualization Using Docker

• Portable: docker images run anywhere docker runs
• Docker decouples LXC provider from operations
  – uses virtual resources (LXC virtualization)
    • fair share of physical NIC vs use virtual NICs that are fair-shared

Docker Images and Use

• Docker uses a union file system (AuFS)
  – allows containers to use host FS safely
• Essentially a copy-on-write file system
  – read-only files shared (e.g., share glibc)
  – make a copy upon write
• Allows for small efficient container images
• Docker Use Cases
  – “Run once, deploy anywhere”
  – Images can be pulled/pushed to repository
  – Containers can be a single process (useful for microservices) or a full OS
Use of Virtualization Today

• Data centers:
  – server consolidation: pack multiple virtual servers onto a smaller number of physical server
    • saves hardware costs, power and cooling costs
• Cloud computing: rent virtual servers
  – cloud provider controls physical machines and mapping of virtual servers to physical hosts
  – User gets root access on virtual server
• Desktop computing:
  – Multi-platform software development
  – Testing machines
  – Run apps from another platform

Case Study: PlanetLab

• Distributed cluster across universities
  – Used for experimental research by students and faculty in networking and distributed systems
• Uses a virtualized architecture
  – Linux Vservers
  – Node manager per machine
  – Obtain a “slice” for an experiment: slice creation service
Server Design Issues

- **Server Design**
  - Iterative versus concurrent
- **How to locate an end-point (port #)?**
  - Well known port #
  - Directory service (port mapper in Unix)
  - Super server (inetd in Unix)

- **Stateful or Stateless?**
  - **Stateful server**
    - Maintain state of connected clients
    - Sessions in web servers
  - **Stateless server**
    - No state for clients
  - **Soft state**
    - Maintain state for a limited time; discarding state does not impact correctness
Server Clusters

- Web applications use tiered architecture
  - Each tier may be optionally replicated; uses a dispatcher
  - Use TCP splicing or handoffs

Server Architecture

- Sequential
  - Serve one request at a time
  - Can service multiple requests by employing events and asynchronous communication

- Concurrent
  - Server spawns a process or thread to service each request
  - Can also use a pre-spawned pool of threads/processes (apache)

- Thus servers could be
  - Pure-sequential, event-based, thread-based, process-based

- Discussion: which architecture is most efficient?
Scalability

- *Question:* How can you scale the server capacity?
- Buy bigger machine!
- Replicate
- Distribute data and/or algorithms
- Ship code instead of data
- Cache