TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

Cloud Enabling Technology, Containerization

Wes J. Lloyd
School of Engineering and Technology
University of Washington – Tacoma

MW 5:50-7:50 PM

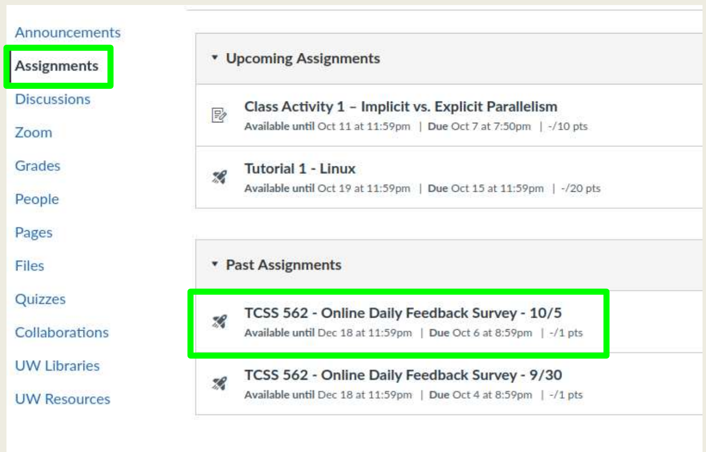
OBJECTIVES – 11/18

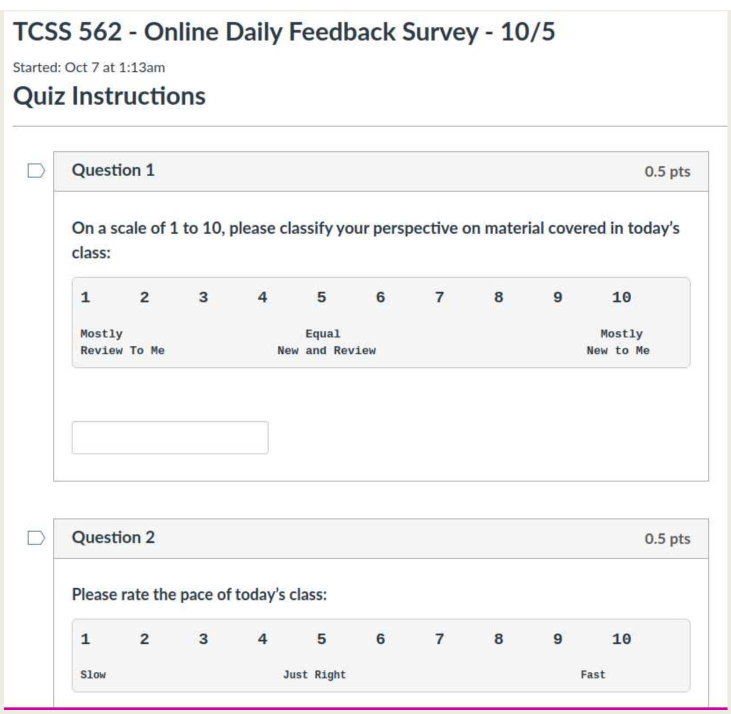
- Questions from 11/9
- Quiz 2– due Mon 11/23 @ noon (note: *no grace period*)
- Group Presentation Overview:
  Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Mon 11/30 @ 11:59p
- Introduction to Containerization

- 2nd hour:
- Tutorial 7 – to be posted
- Introduction to Containerization cont'd
- Tutorial questions
- Team planning

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Take After Each Class
- Extra Credit for completing





MATERIAL / PACE

- Please classify your perspective on material covered in today's class (22 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- Average – 6.09 (↓ - *previous 6.55*)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- Average – 5.55 (↑ - *previous 5.41*)

FEEDBACK FROM 11/16

- *Can you please explain about if there is any compatibility required for the Host OS and Guest OS in the hypervisors?*

- Type 1 Hypervisors generally require the Guest OS to support being virtualized
- Traditionally a special OS kernel was provided
- This kernel has special TRAPS where privileged instructions/operations are trapped as running them directly on the HW without emulation/simulation will cause corruption

TYPE 1 VS. TYPE 2 HYPERVISORS

- **Comparison of: Paravirtualization (type I) vs. Full (type II) hypervisors**
- **GOAL**: run all user mode instructions directly on the CPU
- x86 instruction set has **~17** privileged user mode instructions

- **MAJOR DIFFERENCE:**
- **Full virtualization**: scan the EXE, insert code around privileged instructions to divert control to the VMM
  - *THIS IS SOFTWARE EMULATION*
  - *Imagine how this might be slow…*
- **Paravirtualization**: special OS kernel eliminates side effects of privileged instructions
  - SPECIAL INSTRUCTIONS ARE TRAPPED BY A SPECIALIZED VERSION OF THE OPERATIONG SYSTEM KERNEL AND HANDLED

TUTORIAL QUESTIONS

- Tutorial 5: Thursday Nov 19th @ 11:59p
- Tutorial 6: Tuesday Nov 24th @ 11:59p

UPCOMING TUTORIALS

- Tutorial 7 – Introduction to Docker Containerization
- Going further - optional tutorials:
  - Ungraded or substitute
- Tutorial 8 – Introduction to FaaS IV: Step Functions and SQS
- Tutorial 9 – Asynchronous Function Profiling with SAAF
- Tutorial 10 – Automating Experiments with SAAF & FaaS Runner
- Tutorial 11 – Scaling beyond a single client – concurrent webservice benchmarking with multiple EC2 instances

OBJECTIVES – 11/18

- Questions from 11/9
- Quiz 2– due Mon 11/23 @ noon (note: *no grace period*)
- Group Presentation Overview:
  Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Mon 11/30 @ 11:59p
- Introduction to Containerization
- 2nd hour:
- Tutorial 7 – to be posted
- Introduction to Containerization cont'd
- Tutorial questions
- Team planning

OBJECTIVES – 11/18

- Questions from 11/9
- Quiz 2– due Mon 11/23 @ noon (note: *no grace period*)
- Group Pres
  Cloud Techr 30 – 12/9
- Term Projec 1:59p
- Introductior

Quiz 2 Coverage:

- Focus on lectures 7 – 12
- AWS
- Tutorials

- 2nd hour:
- Tutorial 7 –
- Introduction to Containerization cont'd
- Tutorial questions
- Team planning

OBJECTIVES – 11/18

- Questions from 11/9
- Quiz 2– due Mon 11/23 @ noon (note: *no grace period*)
- Group Presentation Overview:
  Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Mon 11/30 @ 11:59p
- Introduction to Containerization

- 2nd hour:
- Tutorial 7 – to be posted
- Introduction to Containerization cont'd
- Tutorial questions
- Team planning

GROUP PRESENTATION

- TWO OPTIONS:
- ***Cloud technology presentation***
- ***Cloud research paper presentation***
  - Recent & suggested papers will be posted at: http://faculty.washington.edu/wlloyd/courses/tcss562/papers/
- Submit presentation type and topics (paper or technology) with desired dates of presentation via Canvas by Monday November 23rd @ 11:59pm
- Presentation dates:
  - Monday November 30, Wednesday December 2
  - Monday December 7, Wednesday December 9

OBJECTIVES – 11/18

- Questions from 11/9
- Quiz 2– due Mon 11/23 @ noon (note: *no grace period*)
- Group Presentation Overview:
  Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Mon 11/30 @ 11:59p
- Introduction to Containerization
- 2nd hour:
- Tutorial 7 – to be posted
- Introduction to Containerization cont'd
- Tutorial questions
- Team planning

OBJECTIVES – 11/18

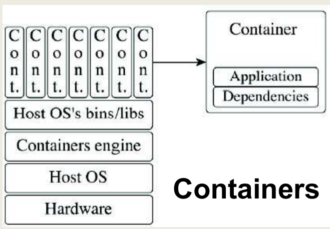
- Questions from 11/9
- Quiz 2– due Mon 11/23 @ noon (note: *no grace period*)
- Group Presentation Overview:
  Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Mon 11/30 @ 11:59p
- Introduction to Containerization

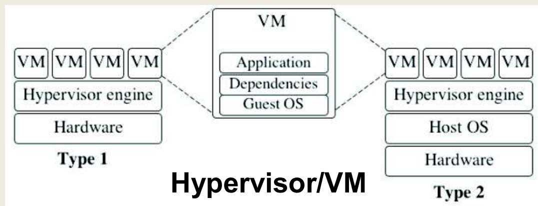
- 2nd hour:
- Tutorial 7 – to be posted
- Introduction to Containerization cont'd
- Tutorial questions
- Team planning

CONTAINERIZATION

MOTIVATION FOR CONTAINERIZATION

- Containers provide "light-weight" alternative to full OS virtualization provided by a hypervisor
- Containers do not provide a full "machine"
- Instead use operating system constructs to provide "sand boxes" for execution
  - Linux cgroups, namespaces, etc.
- Containers can run on bare metal, or atop of VMs





CONTAINER PERFORMANCE – LU FACTORIZATION PERFORMANCE

- Solve linear equations – matrix algebra

Performance data from IC2E 2015: Hypervisors vs. Lightweight Virtualization: A Performance Comparison

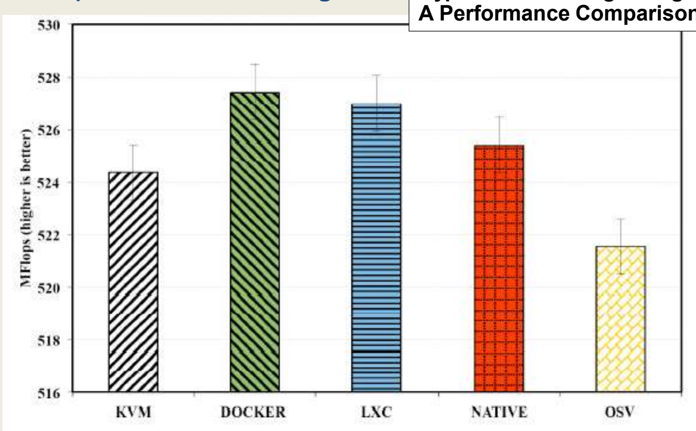
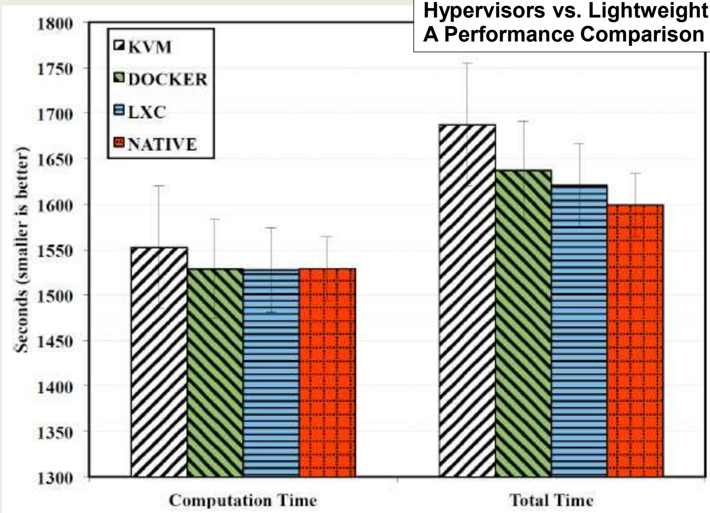


Fig. 4. The value of Linpack results on each platform over 15 runs. This is the particular case of N=1000.

CONTAINER PERFORMANCE – Y-CRUNCHER: PI CALCULATOR

Performance data from IC2E 2015:
Hypervisors vs. Lightweight Virtualization:
A Performance Comparison



November 18, 2020 | TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma | L14.19

CONTAINER PERFORMANCE – BONNIE++

Performance data from IC2E 2015:
Hypervisors vs. Lightweight Virtualization:
A Performance Comparison

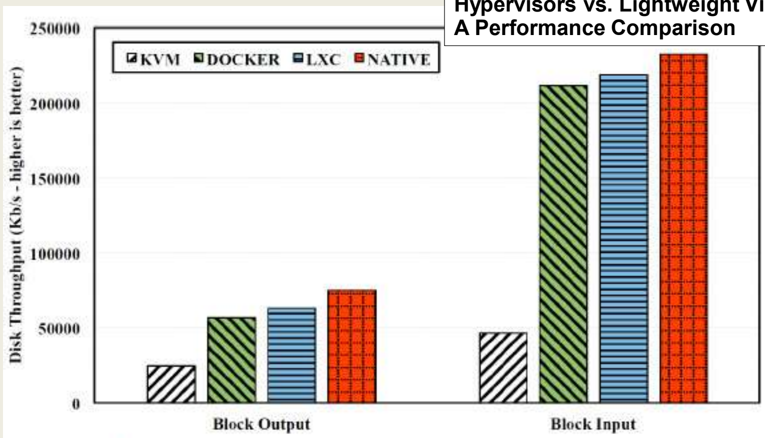


Fig. 6. Disk Throughput achieved by running Bonnie++ (test file of 25 GiB). Results for sequential writes and sequential read are shown.

November 18, 2020 | TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma | L14.20

WHAT IS A CONTAINER?

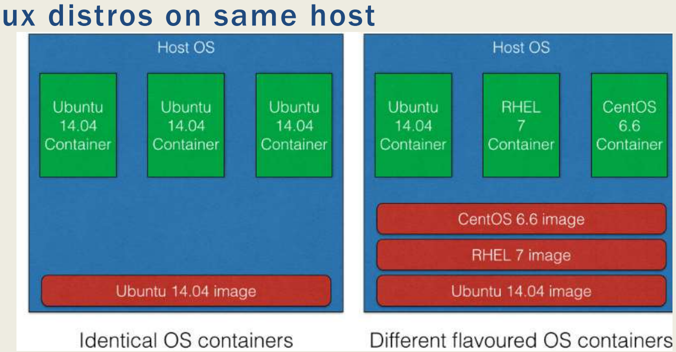
According to NIST (National Institute of Standards Technology)

- **Virtualization**: the simulation of the software and/or hardware upon which other software runs. (800-125)
- **System Virtual Machine**: A System Virtual Machine (VM) is a software implementation of a complete system platform that supports the execution of a complete operating system and corresponding applications in a cloud. (800-180 draft)
- **Operating System Virtualization** (aka OS Container): Provide multiple virtualized OSes above a single shared kernel (800-190). E.g., Solaris Zone, FreeBSD Jails, LXC
- **Application Virtualization** (aka Application Containers): Same shared kernel is exposed to multiple discrete instances (800-180 draft). E.g., Docker (containerd), rkt

November 18, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.21

OPERATING SYSTEM CONTAINERS

- Virtual environments: share the host kernel
- Provide user space isolation
- Replacement for VMs: run multiple processes, services
- Mix different Linux distros on same host
- Examples: LXC, OpenVZ, Linux Vserver, BSD Jails, Solaris zones



- Credit: https://blog.risingstack.com/operating-system-containers-vs-application-containers/

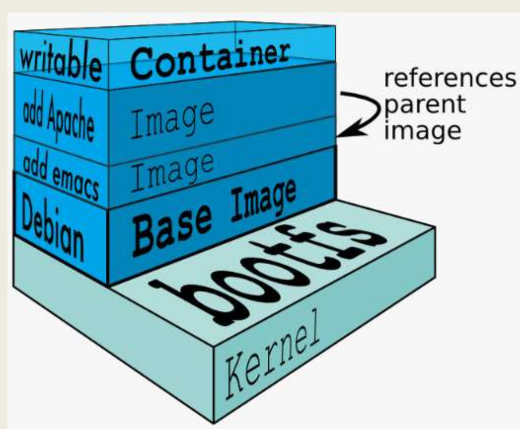
November 18, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.22

APPLICATION CONTAINERS

- Designed to package and run a single service
- All containers share host kernel
- Subtle differences from operating system containers
- Examples: Docker, Rocket
- Docker: runs a single process on creation
- OS containers: run many OS services, for an entire OS
- Create application containers for each component of an app
- Supports a micro-services architecture
- DevOPS: developers can package their own components in application containers
- Supports horizontal and vertical scaling

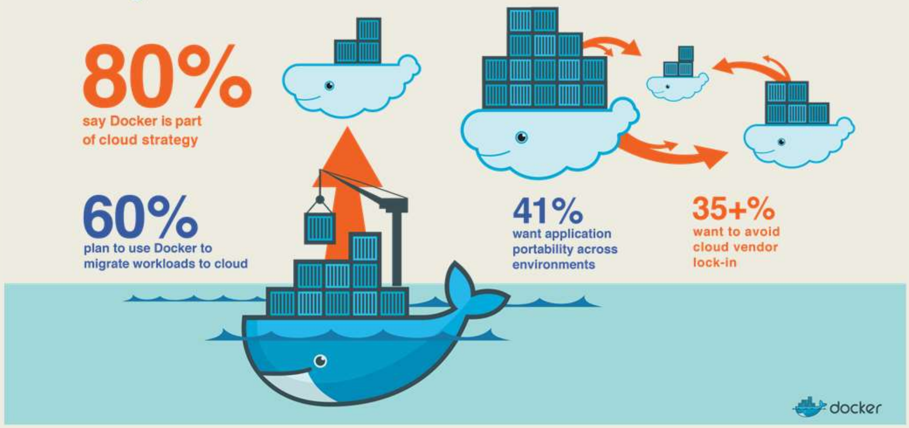
APPLICATION CONTAINERS - 2

- Container images are "layered"
- Base image: common for all components
- Add layers that are specific for components, services as needed
- Layering promotes reuse
- Reduces duplication of data across images



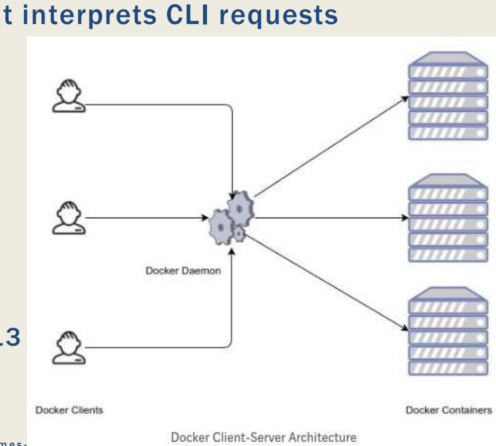
2016 DOCKER SURVEY

- Docker application containers
  - Leading containerization vehicle



DOCKER

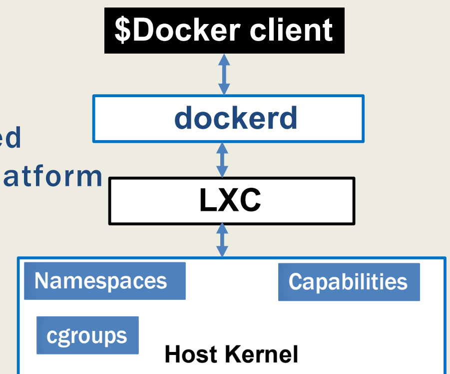
- Docker daemon "dockerd"
  - Implements docker engine that interprets CLI requests and creates/manages containers using backend layered Docker architecture
- Starting in 2017 version numbering switches from 1.x to YR.x
- 2017 releases: 17.03 – 17.12
- 2018 releases: 18.01 – 18.09
- 2019 releases: 19.03.0 – 19.03.13



- Credit: https://hackernoon.com/docker-containerd-standalone-runtimes-

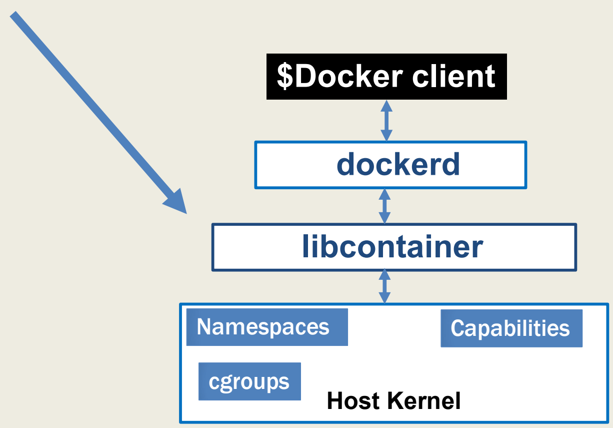
ORIGINAL DOCKER ENGINE IMPLEMENTATION

- (1) Original Docker engine relied on LXC
  - LXC itself is a containerization tool predating Docker
  - Original Docker API just called it
  - LXC originally provided access to Linux kernel features: namespaces and cgroups
  - LXC was Linux specific – caused issues if wanting to be multi-platform
  - Docker implemented their own replacement for LXC



INTRODUCTION OF LIBCONTAINER

- Docker v0.9: libcontainer introduced (~2014) to replace LXC as the default Docker daemon



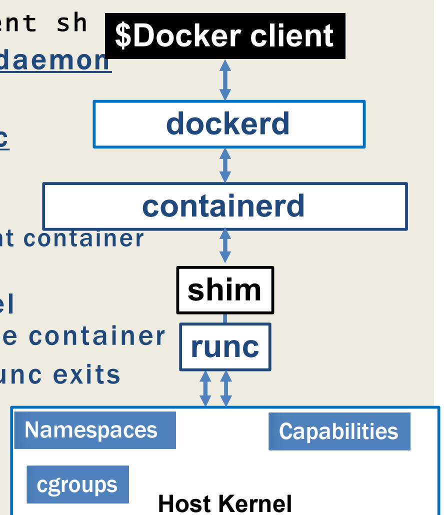
OPEN CONTAINER INITIATIVE (OCI)

- OCI created container standards for:
  - Image specification
  - Container runtime specification
- Docker 1.1 (2016): Docker refactored the docker engine to be compliant with OCI standards
  - Essentially this introduced abstraction layers (i.e. generic interfaces that map to the implementation) so that Docker's design conformed to the OCI standard
- Runc was added to implement the OCI container runtime spec
  - Provides small, lightweight wrapper for libcontainer
  - Can build and run OCI compliant containers directly using runc provided in Docker, but it is "bare bones" and low-level.
    - The Docker API is much more user friendly
- Support for OCI compliant images was added to **Containerd**

November 18, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.29

CREATING A CONTAINER

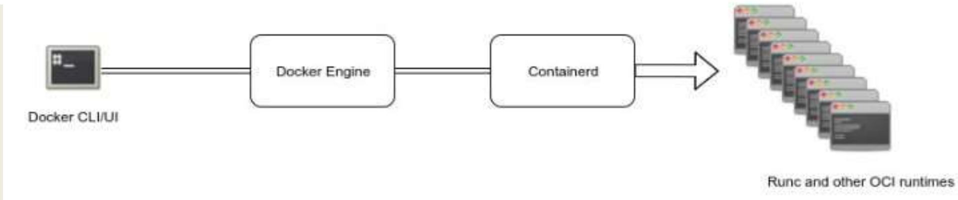
```
$ docker run -it --rm tcss558client sh
```



- Docker CLI posts request to Docker daemon
- Daemon calls containerd
- Containerd passes of request to runc
  - Containerd converts docker image into OCI compliant bundle
  - This step would allow any OCI compliant container to be plugged into the back-end
- Runc interfaces with the Linux kernel (namespaces, cgroups, etc.) to create container
- Shim: once a container is created, runc exits
  - Shim remains as a daemonless stub to implement the container
  - Allows Docker to be upgraded w/o stopping the container !!!

November 18, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.30

CREATING A CONTAINER - 2

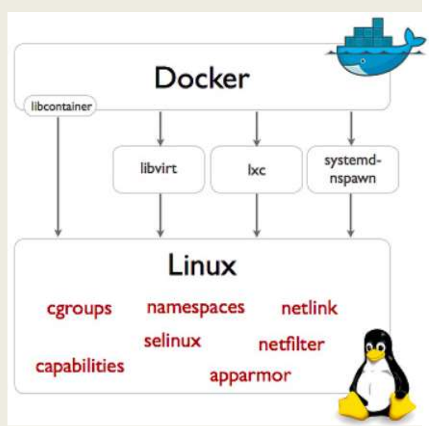


Containerd Integration Architecture

- Docker CLI: interfaces with **dockerd** daemon
- Docker engine: **dockerd** daemon, interfaces with **containerd**
- **Containerd**: simple daemon, interfaces with **runc** to manage containers; CRUD interface for containers, images, volumes, networks, builds; HTTP API → Google RPC (gRPC) interface;
- **runc**: lightweight command-line tool for running containers; Interfaces with Linux cgroups, namespaces; Runs an OCI container

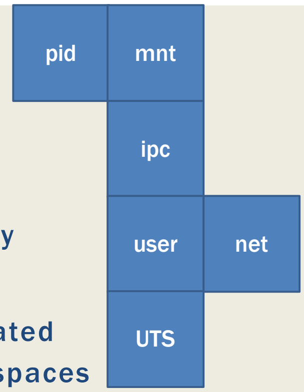
SUPPORT FOR ALTERNATE CONTAINER RUNTIMES

- Modularity of Docker implementation supports "execution drivers concept":
- Enables docker to support many alternate container backends
- OpenVZ, system-nspawn, libvirt-lxc, libvirt-sandbox, qemu/kvm, BSD Jails, Solaris Zones, and chroot

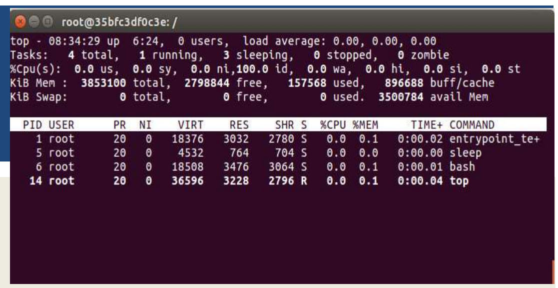


LINUX KERNEL NAMESPACES

- Partitions kernel resources
- Processes see only their set of resources
- Provides isolation
- Namespaces are hierarchical
- Parent processes can see down the hierarchy
- 7 namespaces in Linux (cgroups not shown)
- Each process can only see resources associated with the namespace, and descendent namespaces



NAMESPACES - 2



- Provides isolation of OS entities for containers
- mnt: separate filesystems
- pid: independent PIDs; first process in container is PID 1
- ipc: prevents processes in different IPC namespaces from being able to establish shared memory. Enables processes in different containers to reuse the same identifiers without conflict.
  … provides expected *VM like isolation…*
- user: user identification and privilege isolation among separate containers
- net: network stack virtualization. Multiple loopbacks (lo)
- UTS (UNIX time sharing): provides separate host and domain

CONTROL GROUPS (CGROUPS)

- Collection of Linux processes
- Group-level resource allocation: *CPU, memory, disk I/O, network I/O*
- **Resource limiting**
  - Memory, disk cache
- **Prioritization**
  - CPU share
  - Disk I/O throughput
- **Accounting**
  - Track resource utilization
  - For resource management and/or billing purposes
- **Control**
  - Pause/resume processes
  - Checkpointing → Checkpoint/Restore in Userspace (CRIU)
  - https://criu.org

CGROUPS - 2

- Control groups are hierarchical
- Groups inherent limits from parent groups
- Linux has multiple cgroup controllers (subsystems)
- ls /proc/cgroups
- "memory" controller limits memory use
- "cpuacct" controller accounts for CPU usage
- **cgroup filesystem:**
- /sys/fs/cgroup
- Can browse resource utilization of containers…

#subsys_name	hierarchy	num_cgroups	enabled
cpuset	3	2	1
cpu	5	97	1
cpuacct	5	97	1
blkio	8	97	1
memory	9	218	1
devices	6	97	1
freezer	4	2	1
net_cls	2	2	1
perf_event	10	2	1
net_prio	2	2	1
hugetlb	7	2	1
pids	11	98	1

OVERLAY FILE SYSTEMS

- Docker leverages overlay filesystems
- 1st: AUFS - Advanced multi-layered unification filesystem
- Now: overlay2
- Union mount file system: combine multiple directories into one that appears to contain combined contents

- Idea: Docker uses layered file systems
- Only the top layer is writeable
- Other layers are read-only
- Layers are merged to present the notion of a real file system
- Copy-on-write- implicit sharing
  - Implement duplicate copy

- https://medium.com/@nagarwal/docker-containers-filesystem-demystified-b6ed8112a04a

- https://www.slideshare.net/jpetazzo/scale11x-lxc-talk-1/

LAYERED FS: BUILDING A CONTAINER

- Dockerfile:

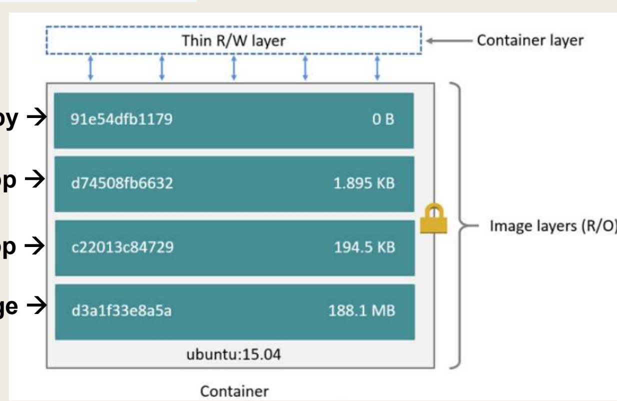
```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Python /app/app.py →

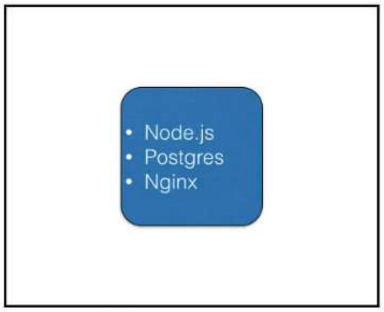
Run make /app →

Copy . /app →

Ubuntu base image →

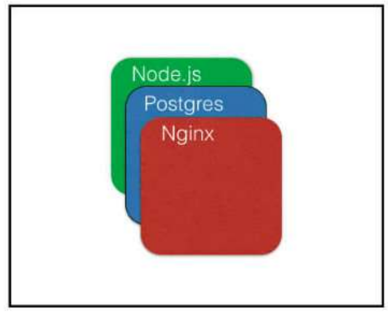


THREE-TIER ARCHITECTURE



OS containers

- Meant to used as an OS - run multiple services
- No layered filesystems by default
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, OpenVZ, Linux VServer, BSD Jails, Solaris Zones



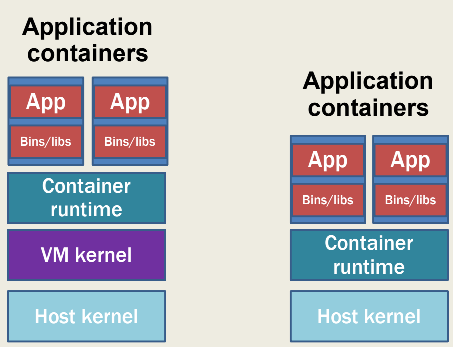
App containers

- Meant to run for a single service
- Layered filesystems
- Built on top of OS container technologies
- Examples - Docker, Rocket

November 18, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.39

CONTAINER ISOLATION

- Is the host isolated from application containers?
- Are application containers isolated from each other?



November 18, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.40

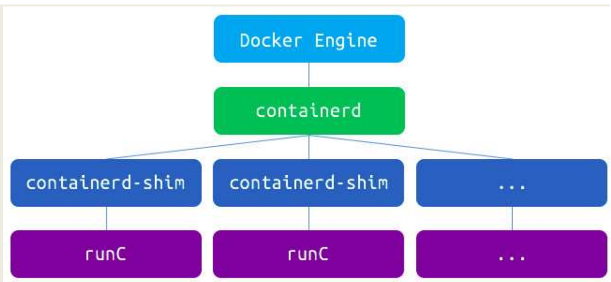
LXC (LINUX CONTAINERS)

- Operating system level virtualization
- Run multiple isolated Linux systems on a host using a single Linux kernel
- Control groups(cgroups)
  - Including in Linux kernels => 2.6.24
  - Limit and prioritize sharing of CPU, memory, block/network I/O
- Linux namespaces
- Docker initially based on LXC

November 18, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.41

OTHER DOCKER TOOLS

- **Docker Machine:** automatically provision and manage sets of docker hosts to form a cluster



- **Docker Swarm**: Clusters multiple docker hosts together to manage as a cluster.
- **Docker Compose**: Config file (YAML) for multi-container application; Describes how to deploy and configure multiple containers

November 18, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.42

CONTAINER ORCHESTRATION FRAMEWORKS

- Framework(s) to deploy multiple containers
- Provide container clusters using cloud VMs
- Similar to "private clusters"
- Reduce VM idle CPU time in public clouds
- Better leverage "sunk cost" resources
- Compact multiple apps onto shared public cloud infrastructure
- Generate to cost savings
- Reduce vendor lock-in

KEY ORCHESTRATION FEATURES

- Management of container hosts
- Launching set of containers
- Rescheduling failed containers
- Linking containers to support workflows
- Providing connectivity to clients outside the container cluster
- Firewall: control network/port accessibility
- Dynamic scaling of containers: horizontal scaling
  - Scale in/out, add/remove containers
- Load balancing over groups of containers
- Rolling upgrades of containers for application

CONTAINER ORCHESTRATION FRAMEWORKS - 2

- Docker swarm
- Apache mesos/marathon
- Kubernetes
  - Many public cloud provides moving to offer Kubernetes-as-a-service
- Amazon elastic container service (ECS)
- Apache aurora

- Container-as-a-Service
  - Serverles containers without managing clusters
  - Azure Container Instances, AWS Fargate…

WE WILL RETURN AT ~7:05PM

OBJECTIVES – 11/18

- Questions from 11/9
- Quiz 2– due Mon 11/23 @ noon (note: *no grace period*)
- Group Presentation Overview:
  Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Mon 11/30 @ 11:59p
- Introduction to Containerization

- 2nd hour:
- Tutorial 7 – to be posted
- Introduction to Containerization cont'd
- Tutorial questions
- Team planning

November 18, 2020	TCSS562:Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L14.47

TUTORIAL #7

DOCKER, CGROUPS, RESOURCE ISOLATION

November 18, 2020 TCSS562: Software Engineering for Cloud Computing [Fa 2020] School of Engineering and Technology, University of Wash gton - Tacoma L14.48

TUTORIAL COVERAGE

- Docker CLI → Docker Enginer (dockerd) → containerd → runc
- Concepts:
- Docker installation
- Working with docker files
- Docker run – create a container
- Docker ps – list containers
- Docker exec –it – run a process in an existing container
- Docker stop –stop container

Docker CLI

```
Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  deploy      Deploy a new stack or update an existing stack
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

TUTORIAL 7

- Linux performance benchmarks

- stress-ng
- 100s of CPU, memory, disk, network stress tests

- Sysbench
- Used in tutorial for memory stress test

QUESTIONS