TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

Containerization

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

1

OVERVIEW

- Tutorial 7 is posted
- Tutorial 8 & 9 to be posted next
- Only 7 tutorials are required
  - Additional tutorials beyond 7 provide extra credit
- Group presentation topics are assigned
  - All papers posted online
  - Nov 25th presenters – submit slides by ~ Sat Nov 23
  - Non-presenters: submit 2-questions/talk to Canvas after class
- Term project checkin - due Sunday 11/24
- Grading: tutorial 4 this week, tutorial 5 next

2

GROUP PRESENTATION SCHEDULE

Monday November 25
1. Group 6 - Amazon Dynamo DB
2. Group 8 - Paper: Serverless computation with Open Lambda
3. Group 2 - Paper: A Programming Model and Middleware for High Throughput Serverless Computing Applications

Monday December 2
1. Group 9 - Paper: Performance comparison of container-based technologies for the Cloud
2. Group 10 - Paper: An Investigation of the Impact of Language Runtime on the Performance and Cost of Serverless Functions
3. Group 4 - Paper: Exploring Serverless Computing for Neural Network Training

Wednesday December 4
1. Group 1 - Paper: Performance evaluation of heterogeneous cloud functions
2. Group 7 - Amazon Cognito
3. Group 3 - Paper: Serverless computing - economics and architecture impact

3

TERM PROJECT DELIVERABLES

- Nine project teams
- Term project lightning presentations
  - Monday December 9th (5:50-7:50pm)
  - Takes place of final exam
  - Presentation length: 5 minutes + questions, total 8 minutes
  - Format and rubric coming soon
- Term project final paper and source code repository
  - Friday December 13 @ 11:59pm
  - Paper template to be provided

4

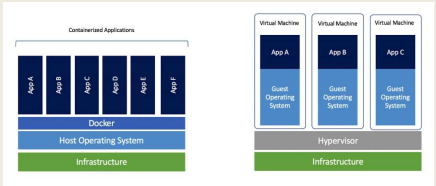
GROUP PRESENTATION

- Cloud technology presentation
- Cloud research paper presentation
- Submit topics and desired dates of presentation via Canvas by Monday November 18th @ 11:59pm
- Presentation dates:
  - Monday November 25 (3 groups)
  - Monday December 2 (3 groups)
  - Wednesday December 4 (3 groups)

5

FEEDBACK FROM 11/18

- Perspective on material: 6.42 (→ more new)
- Pace of class: 5.0 (just right)
- 12 respondents
- **What is the differences between an OS container and a VM?**



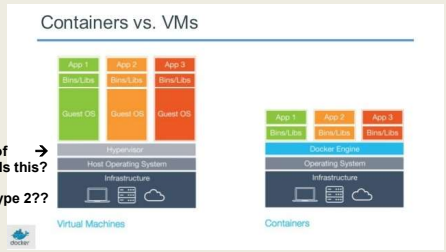
← What type of hypervisor is this?

Type 1 or Type 2??

6

FEEDBACK - 2

- What is the differences between an OS container and a VM?



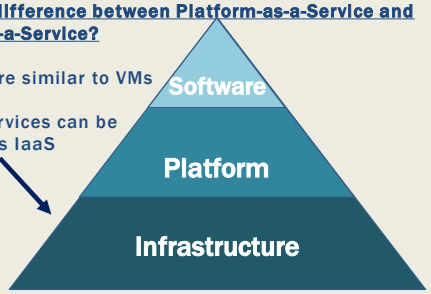
What type of hypervisor is this? →

Type 1 or Type 2??

7

FEEDBACK - 3

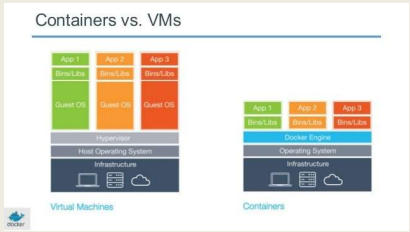
- What is the difference between Platform-as-a-Service and Container-as-a-Service?
- Containers are similar to VMs
- Container services can be considered as IaaS



8

FEEDBACK - 4

- Why do containers have less overhead than VMs?
- Can count the # of layers of abstraction



9

FEEDBACK - 5

- I read in a paper that Azure Functions allocates memory dynamically unlike AWS Lambda starting with 128 MB
- I want to understand how does it allocate memory to a function dynamically
  - My understanding is it doesn't, rather the code runs in an environment (e.g. VM) with ~4 GB ram, and this environment can run multiple function instances, billing is based on how much total RAM is used
  - Azure doesn't expose RAM used by individual functions
- Azure Issue: function memory usage of individual calls is not reported by the platform
  - No easy way to reconcile the bill based on memory use of individual functions
- Issue: Expose memory usage for each function execution
- Open for ~ 2 years:
- https://github.com/Azure/azure-functions-host/issues/1451

10

AZURE FUNCTIONS - ISSUE 1451

- MS Azure Developer – July 2019:
- One option we have been discussing is enabling the export of per execution billing data to Azure Monitor logs. You could then analyze the data using Log Analytics or take advantage of the extensibility features of Azure Monitor to pump this data to another system. This design is likely to be easier for us to implement than some of the other alternatives we've considered.
- One thing to keep in mind is that this would not give you a real-time view of execution cost. There would be at least a few minutes of delay between a function finishing execution and the cost data becoming available in the logs.
- If we took this approach, would this address your needs? Please let us know. Thanks!

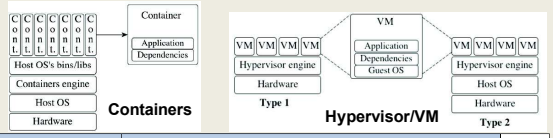
11

CONTAINERIZATION

12

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



13

CONTAINER PERFORMANCE – LU FACTORIZATION PERFORMANCE

- Solve linear equations – matrix algebra

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

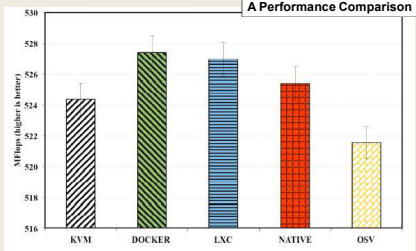
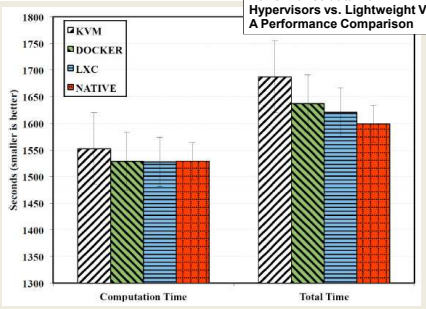


Fig. 4. The value of Linpack results on each platform over 15 runs. This is the particular case of N=1000.

14

CONTAINER PERFORMANCE – Y-CRUNCHER: PI CALCULATOR

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



15

CONTAINER PERFORMANCE – BONNIE++

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

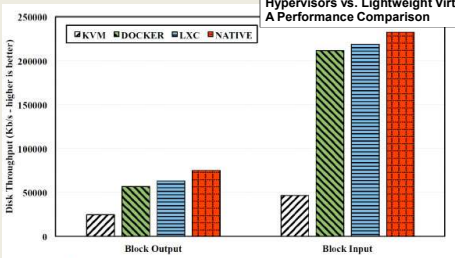


Fig. 6. Disk Throughput achieved by running Bonnie++ (test file of 25 GiB). Results for sequential writes and sequential read are shown.

16

WHAT IS A CONTAINER?

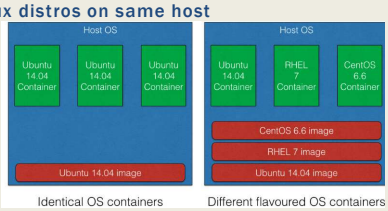
According to NIST (National Institute of Standards Technology)

- **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

17

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/

18

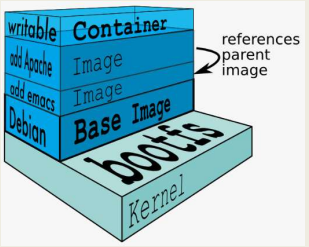
APPLICATION CONTAINERS

- 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

19

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



20

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

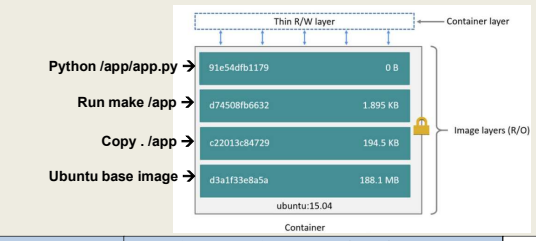
- https://www.slideshare.net/jpetazzo/scale11x-lxc-talk-1/

21

LAYERED FS: BUILDING A CONTAINER

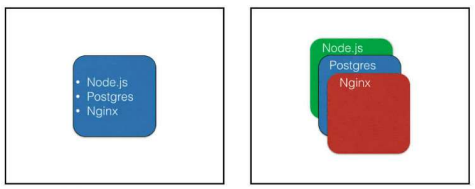
- Dockerfile:

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



22

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

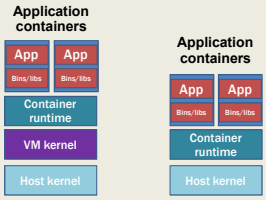
App containers

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

23

CONTAINER ISOLATION

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



24

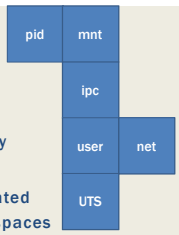
LXC (LINUX CONTAINERS)

- 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

25

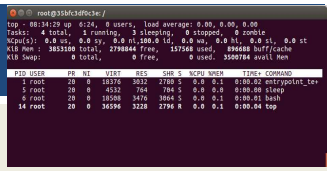
LINUX KERNEL NAMESPACES

- 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



26

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

27

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

28

CGROUPS - 2

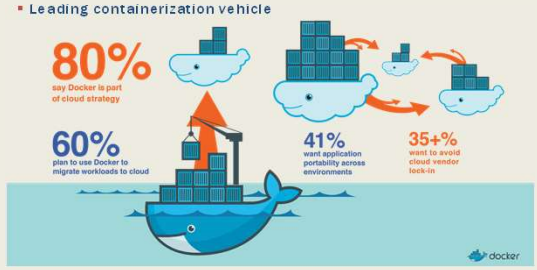
- Control groups are hierarchical
- Groups inherent limits from parent groups
- Linux has multiple cgroup controllers (subsystems)
- ls /proc/cgroups
- "memory" controller limits memory use
- "cpuacct" controller accounts for CPU usage
- **cgroup filesystem:**
- /sys/fs/cgroup
- Can browse resource utilization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

29

2016 DOCKER SURVEY

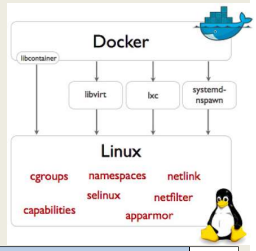
- Docker application containers
  - Leading containerization vehicle



30

DOCKER EXECUTION ENVIRONMENTS

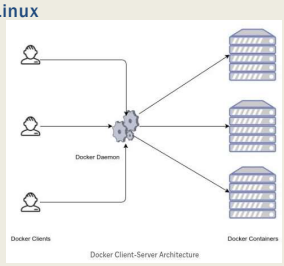
- (1) Original default Docker execution environment: LXC
- (2) Docker v0.9: libcontainer introduced (~2014)
- (3) Now runc (2015)
- Provides Docker access to Linux container APIs
- Execution drivers concept:
- Enable docker to leverage many OS containers as the exec environment
- OpenVZ, system-nspawn, libvirt-lxc, libvirt-sandbox, qemu/kvm, BSD Jails, Solaris Zones, and chroot



31

DOCKER

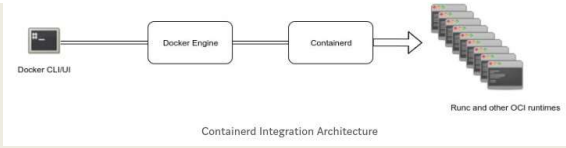
- Docker daemon "dockerd"
  - Provides docker services to Linux
- Docker 1.11+
- Open Container Initiative
- June 2015: Industry standard for container runtimes and formats
- Ensure containers are portable among different execution environments (engines)



- Credit: https://hackernoon.com/docker-containerd-standalone-runtimes-heres-what-you-should-know-b834ef155426

32

DOCKER - 2



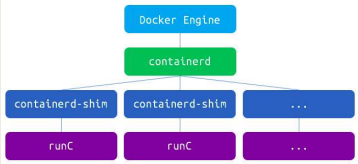
Containerd Integration Architecture

- 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

33

DOCKER - 3

- Docker architecture:
- 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

34

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

35

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

36

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…

37

TUTORIAL #7
DOCKER, CGROUPS, RESOURCE ISOLATION

38

DOCKER CLI

- **Docker CLI → Docker Engine (dockerd) → containerd → runc**

- Docker installation
- Docker file
- Docker run
- Docker ps
- Docker exec -it
- Docker stop

39

```
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
```

40

TUTORIAL 7

- Linux performance benchmarks

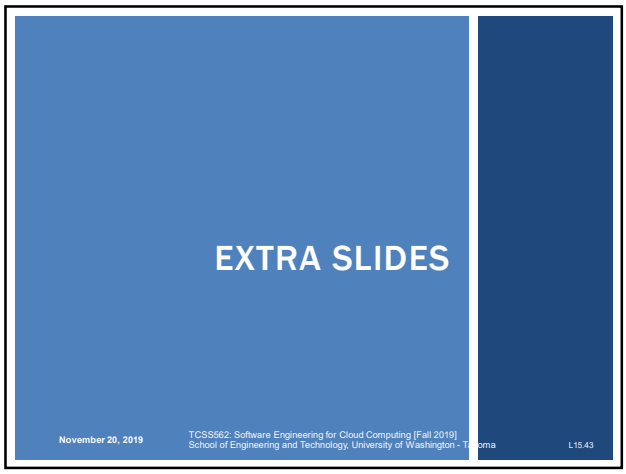
- stress-ng
- 100s of CPU, memory, disk, network stress tests

- Sysbench
- Used in tutorial for memory stress test

41

QUESTIONS

42


EXTRA SLIDES
November 20, 2019
TCSS562: Software Engineering for Cloud Computing [Fall 2019]
School of Engineering and Technology, University of Washington - Tacoma
L15.43

43