


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

November 18, 2020

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

L14.2

ONLINE DAILY FEEDBACK SURVEY

Daily Feedback Quiz in Canvas - Take After Each Class

Extra Credit for completing

Announcements

Assignments

Discussions

Zoom

Grades

People

Pages

Files

Quizzes

Collaborations

UW Libraries

UW Resources

Upcoming Assignments

Class Activity 1 - Implicit vs. Explicit Parallelism
Available until Oct 11 at 11:59pm | Due Oct 7 at 7:50pm | ~10 pts

Tutorial 1 - Linux
Available until Oct 19 at 11:59pm | Due Oct 15 at 11:59pm | ~20 pts

Past Assignments

TCSS 562 - Online Daily Feedback Survey - 10/5
Available until Dec 18 at 11:59pm | Due Oct 6 at 8:59pm | ~1 pts

TCSS 562 - Online Daily Feedback Survey - 9/30
Available until Dec 18 at 11:59pm | Due Oct 4 at 8:59pm | ~1 pts

November 18, 2020

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

L14.3

TCSS 562 - Online Daily Feedback Survey - 10/5

Started: Oct 7 at 1:13am

Quiz Instructions

Question 1

0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

1

2

3

4

5

6

7

8

9

10

Mostly Review To Me

Equal New and Review

Mostly New To Me

Question 2

0.5 pts

Please rate the pace of today's class:

1

2

3

4

5

6

7

8

9

10

Slow

Just Right

Fast

November 18, 2020

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

L14.4

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)

November 18, 2020

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

L14.5

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

November 18, 2020

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

L14.6

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 OPERATING SYSTEM KERNEL AND HANDLED

November 18, 2020

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

L14.7

TUTORIAL QUESTIONS

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

November 18, 2020

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

L14.8

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

November 18, 2020

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

L14.9

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

November 18, 2020

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

L14.10

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

L14.11

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

L14.12

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

November 18, 2020

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

L14.13

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

L14.14

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

L14.15

CONTAINERIZATION



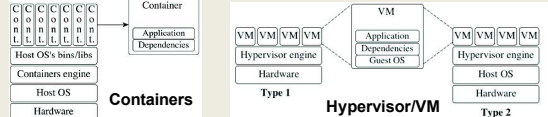
November 18, 2020

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

L14.16

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



November 18, 2020

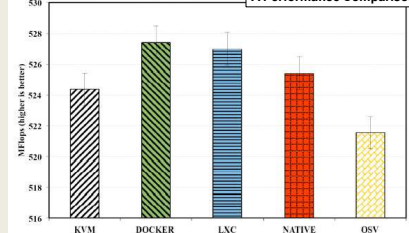
TCSS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L14.17

CONTAINER PERFORMANCE – LU FACTORIZATION PERFORMANCE

- Solve linear equations – matrix algebra

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



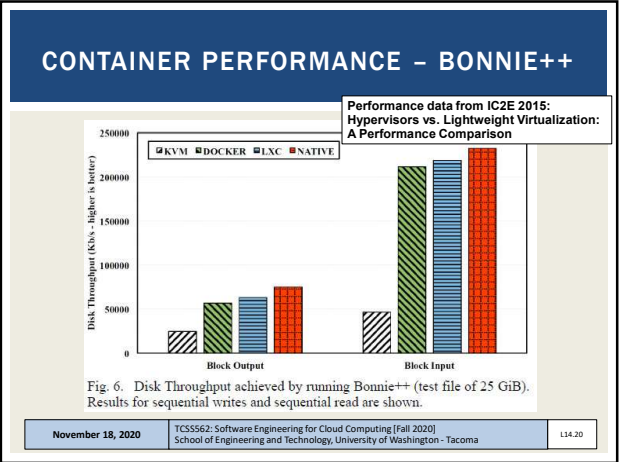
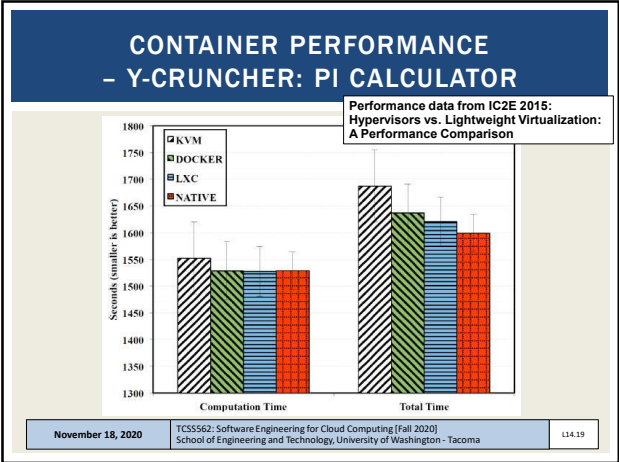
Platform	Linpack Results (Mflops)
KVM	~524
DOCKER	~528
LXC	~527
NATIVE	~525
OSV	~522

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

November 18, 2020

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

L14.18



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

Identical OS containers Different flavoured OS 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

November 18, 2020

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

L14.23

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

November 18, 2020

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

L14.24

2016 DOCKER SURVEY

■ Docker application containers

■ Leading containerization vehicle

80%

say Docker is part of cloud strategy

60%

plan to use Docker to migrate workloads to cloud

41%

want application portability across environments

35+%

want to avoid cloud vendor lock-in

November 18, 2020

TCCS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L14.25

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

Docker Clients

Docker Client-Server Architecture

Docker Containers

■ Credit: <https://hackernoon.com/docker-container-standalone-runtime>

November 18, 2020

TCCS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L14.26

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

\$Docker client

dockerd

LXC

Namespaces

Capabilities

cgroups

Host Kernel

November 18, 2020

TCCS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L14.27

INTRODUCTION OF LIBCONTAINER

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

\$Docker client

dockerd

libcontainer

Namespaces

Capabilities

cgroups

Host Kernel

November 18, 2020

TCCS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L14.28

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

TCCS562: 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 client

■ Docker CLI posts request to **Docker daemon**

■ Daemon calls **containerd**

■ **Containerd** passes off 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 !!!

dockerd

containerd

shim

runc

Namespaces

Capabilities

cgroups

Host Kernel

November 18, 2020

TCCS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L14.30

Slides by Wes J. Lloyd

L14.5

CREATING A CONTAINER - 2

```
graph LR
    CLI[Docker CLI/CLI] --> Engine[Docker Engine]
    Engine --> Containerd[Containerd]
    Containerd --> Runc[Runc and other OCI runtimes]
```

- 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

November 18, 2020

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

L14.31

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

```
graph TD
    Docker --> libcontainer
    Docker --> libvirt
    Docker --> lxc
    Docker --> systemd_nspawn[systemd-nspawn]
    libcontainer --> Linux
    libvirt --> Linux
    lxc --> Linux
    systemd_nspawn --> Linux
    Linux --- cgroups
    Linux --- namespaces
    Linux --- netlink
    Linux --- capabilities
    Linux --- selinux
    Linux --- netfilter
    Linux --- apparmor
```

November 18, 2020

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

L14.32

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

pid	mnt
	ipc
user	net
UTS	

November 18, 2020

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

L14.33

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

```
top - 08:34:28 up 6:29, 0 users, load average: 0.00, 0.00, 0.00
Tasks: 0 total, 0 running, 3 sleeping, 0 stopped, 0 zombie
Mem(s): 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
Mem Mem - 355100 total, 279644 free, 25764 used, 99468 buff/cache
Mem Swap: 0 total, 0 free, 0 used, 350704 avail Mem

PID USER PP NT VIRT RES SPO S RCU MNEN TIME COMMAND
1 root 0 0 12376 2028 0 0 0 0 0.00.00 sleep
5 root 20 0 4532 784 0 0 0 0 0.00.00 sleep
6 root 20 0 18596 3408 2048 0 0 0 0.00.00 bash
14 root 20 0 36596 3228 2796 8 0 0 0.00.04 top
```

November 18, 2020

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

L14.34

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>

November 18, 2020

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

L14.35

CGROUPS - 2

- Control groups are hierarchical
- Groups inherit 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
1	cpuacct	3	1	1
2	cpu	5	97	1
3	cpuacct	5	97	1
4	blkio	8	97	1
5	memory	9	218	1
6	devices	6	97	1
7	freezer	4	2	1
8	net_cls	2	2	1
9	perf_event	10	2	1
10	net_prio	2	2	1
11	hugetlb	7	2	1
12	pids	11	98	1

November 18, 2020

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

L14.36

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-file-system-demystified-b6ed8112a04a>
- <https://www.slideshare.net/jpetazzo/scale11x-lxc-talk-1/>

November 18, 2020

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

L14.37

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 →

Thin R/W layer

Container layer

Image layers (R/O)

Container

November 18, 2020

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

L14.38

THREE-TIER ARCHITECTURE

• Node.js
• Postgres
• Nginx

OS containers

- Meant to be 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

Node.js
Postgres
Nginx

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?

Application containers

App App

Binaries Binaries

Container runtime

VM kernel

Host kernel

Application containers

App App

Binaries Binaries

Container runtime

Host kernel

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

Docker Engine

containerd

containerd-shim containerd-shim ...

runC runC ...

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

November 18, 2020

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

L14.43

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

November 18, 2020

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

L14.44

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...

November 18, 2020

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

L14.45

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 [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L14.48

TUTORIAL COVERAGE

- Docker CLI → Docker Engine (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

November 18, 2020

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

L14.49

Commands:

attach
build
commit
cp
create
deploy
diff
events
exec
export
history
images
import
info
inspect
kill
load
login
logout
logs
pause
port
ps
pull
push
rename
restart
rm
run
save
search
start
stats
stop
tag
top
unpause
update
version
wait

Attach local standard input, output, and error streams to a running container
Build an image from a Dockerfile
Create a new image from a container's changes
Copy files/folders between a container and the local filesystem
Create a new container
Deploy a new stack or update an existing stack
Inspect changes to files or directories on a container's filesystem
Get real time events from the server
Run a command in a running container
Export a container's filesystem as a tar archive
Show the history of an image
List images
Import the contents from a tarball to create a filesystem image
Display system-wide information
Return low-level information on Docker objects
Kill one or more running containers
Load an image from a tar archive or STDIN
Log in to a Docker registry
Log out from a Docker registry
Fetch the logs of a container
Pause all processes within one or more containers
List port mappings or a specific mapping for the container
List containers
Pull an image or a repository from a registry
Push an image or a repository to a registry
Rename a container
Restart one or more containers
Remove one or more containers
Remove one or more images
Run a command in a new container
Save one or more images to a tar archive (streamed to STDOUT by default)
Search the Docker Hub for images
Start one or more stopped containers
Display a live stream of container(s) resource usage statistics
Stop one or more running containers
Create a tag IMAGE_ID that refers to SOURCE_IMAGE
Display the running processes of a container
Unpause all processes within one or more containers
Update configuration of one or more containers
Show the Docker version information
Block until one or more containers stop, then print their exit codes

Docker CLI

TUTORIAL 7


- Linux performance benchmarks
- stress-ng
- 100s of CPU, memory, disk, network stress tests
- Sysbench
- Used in tutorial for memory stress test

November 18, 2020

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

L14.51

QUESTIONS



November 18, 2020

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

L14.52