

# TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

## Cloud Enabling Technology & Docker Containerization

Wes J. Lloyd  
School of Engineering and Technology  
University of Washington - Tacoma  
TR 5:00-7:00 PM



1

## OBJECTIVES - 11/18

- **Questions from 11/16**
- Term Project Proposals - update by 11/19
- Tutorial 5 - Intro to FaaS II - Files in S3, CloudWatch
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- Quiz 1 / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 - 12/9
- Term Project Check-in - due Thur 12/2 @ 11:59p
- Ch. 5: Cloud Enabling Technology - wrap up - wrap up
- Containerization
- Team planning

November 16, 2021	TCSS562:Software Engineering for Cloud Computing [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L13.2
-------------------	---	-------

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 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.3

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:

12345678910

Mostly Review To MeEqual New and ReviewMostly New to Me

Question 2

0.5 pts

Please rate the pace of today's class:

12345678910

SlowJust RightFast

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.4

4

MATERIAL / PACE

- Please classify your perspective on material covered in today’s class (25 respondents):
  - 1-mostly review, 5-equal new/review, 10-mostly new
  - **Average – 6.08** (↓ - *previous 6.32*)
- Please rate the pace of today’s class:
  - 1-slow, 5-just right, 10-fast
  - **Average – 5.20** (↓ - *previous 5.60*)

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.5

5

FEEDBACK FROM 11/9

- ***I noticed that the content of the class (lecture) doesn't correspond to HW tutorials. This makes it hard to complete tutorials.***
- We did complete most of Tutorial 4 in class.
- In addition, the AWS review discussed various aspects relating to EC2 and EBS (Tutorial 3)
- ***Also, sometimes tutorial doesn't explain steps well, leading to numerous questions and doubts.***
- Please do ask questions by: email, canvas message, Zoom chat, Slack channel, or verbally during lecture/office hours
- Tutorials are living documents - there is always potential for improvement with your feedback !

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.6

6

OBJECTIVES – 11/18

- Questions from 11/16
- **Term Project Proposals – update by 11/19**
- Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- Quiz 1 / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Thur 12/2 @ 11:59p
- Ch. 5: Cloud Enabling Technology - wrap up
- Containerization
- Team planning

November 16, 2021

TCSS562:Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.7

7

OBJECTIVES – 11/18

- Questions from 11/16
- Term Project Proposals – update by 11/19
- **Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch**
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- Quiz 1 / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Thur 12/2 @ 11:59p
- Ch. 5: Cloud Enabling Technology - wrap up
- Containerization
- Team planning

November 16, 2021

TCSS562:Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.8

8

OBJECTIVES – 11/18

- Questions from 11/16
- Term Project Proposals – update by 11/19
- Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch
- **Tutorial 6 - Intro to FaaS III - Serverless Databases**
- Quiz 1 / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Thur 12/2 @ 11:59p
- Ch. 5: Cloud Enabling Technology - wrap up
- Containerization
- Team planning

November 16, 2021

TCSS562:Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.9

9

OBJECTIVES – 11/18

- Questions from 11/16
- Term Project Proposals – update by 11/19
- Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- **Quiz 1** / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Thur 12/2 @ 11:59p
- Ch. 5: Cloud Enabling Technology - wrap up
- Containerization
- Team planning

November 16, 2021

TCSS562:Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.10

10

OBJECTIVES – 11/18

- Questions from 11/16
- Term Project Proposals – update by 11/19
- Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- Quiz 1 / **Tutorial 7 - Docker Containerization**
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Thur 12/2 @ 11:59p
- Ch. 5: Cloud Enabling Technology - wrap up
- Containerization
- Team planning

November 16, 2021

TCSS562:Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.11

11

OBJECTIVES – 11/18

- Questions from 11/16
- Term Project Proposals – update by 11/19
- Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- Quiz 1 / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
**Cloud Technology or Research Paper for 11/30 – 12/9**
- Term Project Check-in – due Thur 12/2 @ 11:59p
- Ch. 5: Cloud Enabling Technology - wrap up
- Containerization
- Team planning

November 16, 2021

TCSS562:Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.12

12

OBJECTIVES – 11/18

- Questions from 11/16
- Term Project Proposals – update by 11/19
- Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- Quiz 1 / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 – 12/9
- **Term Project Check-in – due Thur 12/2 @ 11:59p**
- Ch. 5: Cloud Enabling Technology - wrap up
- Containerization
- Team planning

November 16, 2021

TCSS562:Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.13

13

OBJECTIVES – 11/18

- Questions from 11/16
- Term Project Proposals – update by 11/19
- Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- Quiz 1 / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Thur 12/2 @ 11:59p
- **Ch. 5: Cloud Enabling Technology - wrap up**
- Containerization
- Team planning

November 16, 2021

TCSS562:Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.14

14

# CLOUD ENABLING TECHNOLOGY

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma



L13.15

15

# CLOUD ENABLING TECHNOLOGY

- Adapted from Ch. 5 from *Cloud Computing Concepts, Technology & Architecture*
- Broadband networks and internet architecture
- Data center technology
- Virtualization technology
- Multitenant technology
- Web/web services technology

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.16

16



5. WEB SERVICES/WEB

- Web services technology is a key foundation of cloud computing’s “**as-a-service**” cloud delivery model
- SOAP – “Simple” object access protocol
  - First generation web services
  - WSDL – web services description language
  - UDDI – universal description discovery and integration
  - SOAP services have their own unique interfaces
- REST – instead of defining a custom technical interface REST services are built on the use of HTTP protocol
- HTTP GET, PUT, POST, DELETE

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.17

17

HYPertext TRANSPORT PROTOCOL (HTTP)

- An ASCII-based request/reply protocol for transferring information on the web
- HTTP request includes:
  - request method (GET, POST, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - headers—extra info regarding transfer request
- HTTP response from server
  - Protocol version & status code →
  - Response headers
  - Response body

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.18

HTTP status codes:

2xx — all is well

3xx — resource moved

4xx — access problem

5xx — server error

18

## REST: REPRESENTATIONAL STATE TRANSFER

- Web services protocol
- *Supersedes SOAP* – Simple Object Access Protocol
- Access and manipulate web resources with a predefined set of stateless operations (known as web services)
- Requests are made to a URI
- Responses are most often in JSON, but can also be HTML, ASCII text, XML, no real limits as long as text-based
- HTTP verbs: GET, POST, PUT, DELETE, ...

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.19

19

```
// SOAP REQUEST

POST /InStock HTTP/1.1
Host: www.bookshop.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
  <soap:Body xmlns:m="http://www.bookshop.org/prices">
    <m:GetBookPrice>
      <m:BookName>The Fleamarket</m:BookName>
    </m:GetBookPrice>
  </soap:Body>
</soap:Envelope>
```

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.20

20

```
// SOAP RESPONSE
POST /InStock HTTP/1.1
Host: www.bookshop.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.bookshop.org/prices">
    <m:GetBookPriceResponse>
      <m: Price>10.95</m: Price>
    </m:GetBookPriceResponse>
  </soap:Body>
</soap:Envelope>
```

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
 School of Engineering and Technology, University of Washington - Tacoma

L13.21

21

```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DayOfWeek"
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="date" type="xsd:date"/>
  </message>
  <message name="DayOfWeekResponse">
    <part name="dayOfWeek" type="xsd:string"/>
  </message>
  <portType name="DayOfWeekPortType">
    <operation name="GetDayOfWeek">
      <input message="tns:DayOfWeekInput"/>
      <output message="tns:DayOfWeekResponse"/>
    </operation>
  </portType>
  <binding name="DayOfWeekBinding" type="tns:DayOfWeekPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetDayOfWeek">
      <soap:operation soapAction="getdayofweek"/>
      <input>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </input>
      <output>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </output>
    </operation>
  </binding>
  <service name="DayOfWeekService" >
    <documentation>
      Returns the day-of-week name for a given date
    </documentation>
    <port name="DayOfWeekPort" binding="tns:DayOfWeekBinding">
      <soap:address location="http://localhost:8090/dayofweek/DayOfWeek"/>
    </port>
  </service>
</definitions>
```

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
 School of Engineering and Technology, University of Washington - Tacoma

L13.22

22

## REST CLIMATE SERVICES EXAMPLE

■ **USDA  
Lat/Long  
Climate  
Service  
Demo**

```
// REST/JSON
// Request climate data for Washington

{
  "parameter": [
    {
      "name": "latitude",
      "value": 47.2529
    },
    {
      "name": "longitude",
      "value": -122.4443
    }
  ]
}
```

■ **Just provide  
a Lat/Long**

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.23

23

## REST - 2

- App manipulates one or more types of resources.
- Everything the app does can be characterized as some kind of operation on one or more resources.
- Frequently services are CRUD operations (create/read/update/delete)
  - Create a new resource
  - Read resource(s) matching criterion
  - Update data associated with some resource
  - Destroy a particular a resource
- Resources are often implemented as objects in OO languages

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.24

24

## REST ARCHITECTURAL ADVANTAGES

- **Performance:** component interactions can be the dominant factor in user-perceived performance and network efficiency
- **Scalability:** to support large numbers of services and interactions among them
- **Simplicity:** of the Uniform Interface
- **Modifiability:** of services to meet changing needs (even while the application is running)
- **Visiblility:** of communication between services
- **Portability:** of services by redeployment
- **Reliability:** resists failure at the system level as redundancy of infrastructure is easy to ensure

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.25

25

## OBJECTIVES – 11/18


- Questions from 11/16
- Term Project Proposals – update by 11/19
- Tutorial 5 – Intro to FaaS II – Files in S3, CloudWatch
- Tutorial 6 - Intro to FaaS III - Serverless Databases
- Quiz 1 / Tutorial 7 - Docker Containerization
- **Group Presentation Overview:**  
Cloud Technology or Research Paper for 11/30 – 12/9
- Term Project Check-in – due Thur 12/2 @ 11:59p
- Ch. 5: Cloud Enabling Technology - wrap up
- **Containerization**
- Team planning

November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.26

26



CONTAINERIZATION

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.27

27

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

Containers

C

o

n

t.

C

o

n

t.

C

o

n

t.

C

o

n

t.

C

o

n

t.

Host OS's bins/libs

Containers engine

Host OS

Hardware

Container

Application

Dependencies

Containers

VM

VM

VM

VM

Hypervisor engine

Hardware

Type 1

VM

Application

Dependencies

Guest OS

VM

VM

VM

VM

Hypervisor engine

Host OS

Hardware

Type 2

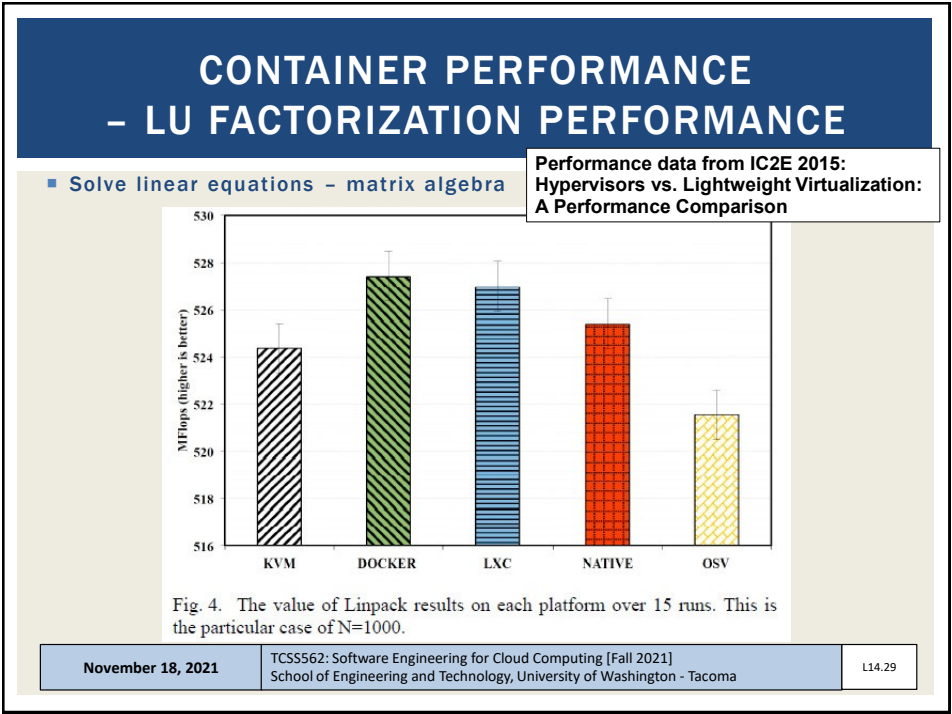
Hypervisor/VM

November 18, 2021

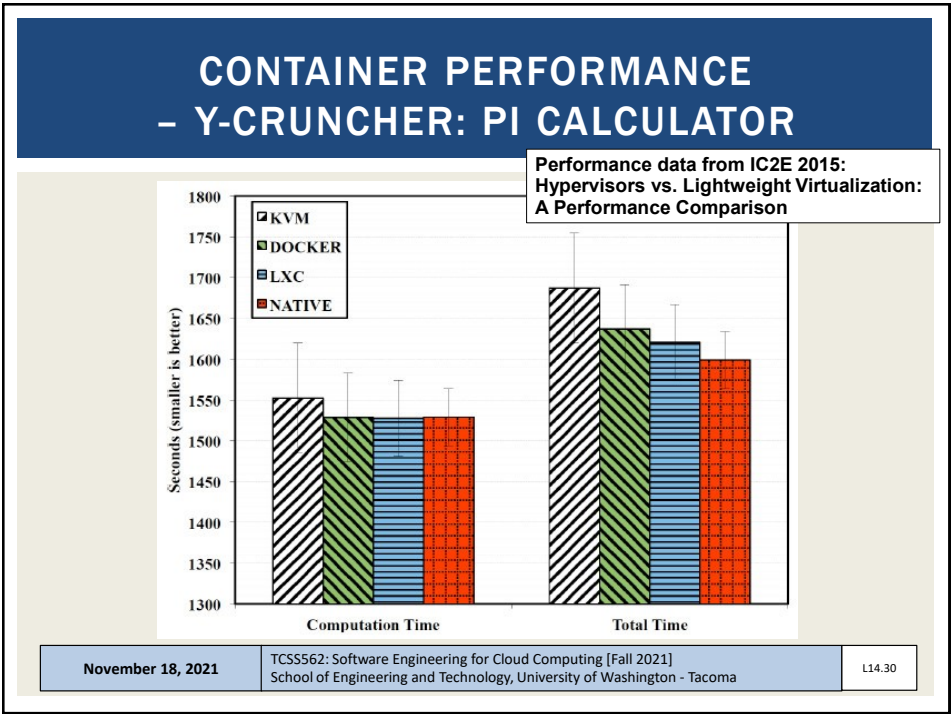
TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.28

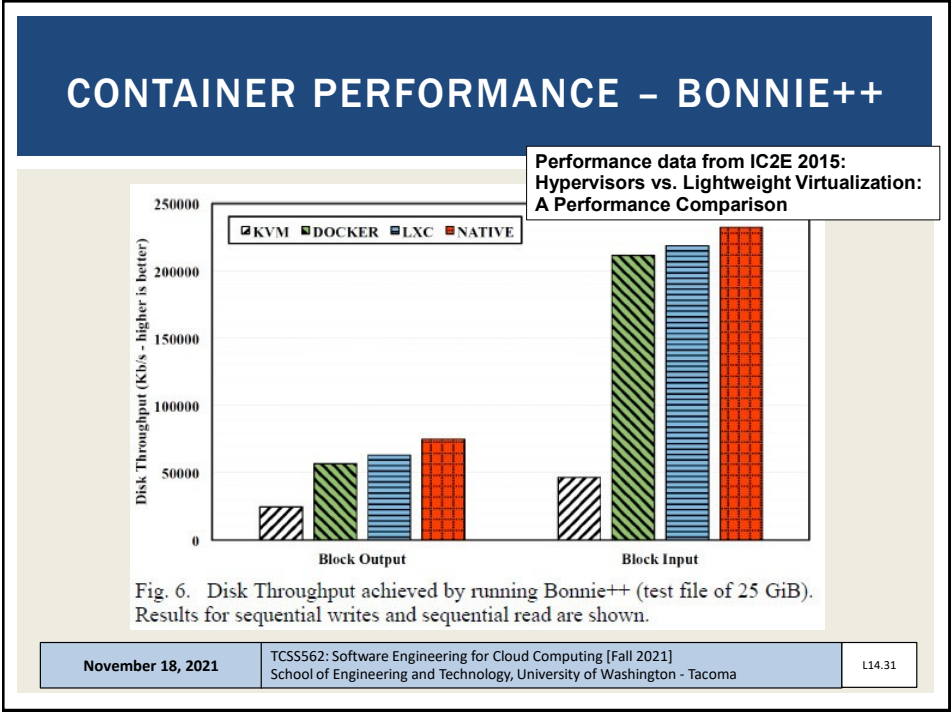
28



29



30



31

## 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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.32

32



## 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

Host OS

Ubuntu 14.04 Container

Ubuntu 14.04 Container

Ubuntu 14.04 Container

Ubuntu 14.04 image

Identical OS containers

Host OS

Ubuntu 14.04 Container

RHEL 7 Container

CentOS 6.6 Container

CentOS 6.6 image

RHEL 7 image

Ubuntu 14.04 image

Different flavoured OS containers

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

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.33

33

## 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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.34

34

Slides by Wes J. Lloyd

L13.17

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

The diagram illustrates the layered structure of container images. It shows a stack of four blue rectangular blocks. The bottom-most block is labeled 'Base Image' and 'Debian'. Above it are two more blocks labeled 'add emacs' and 'add Apache'. The top-most block is labeled 'Container Image' and 'writable'. A curved arrow points from the 'Container Image' block to the 'Base Image' block, with the text 'references parent image' next to it. The entire stack of blocks sits on a light blue base labeled 'boottfs' and 'Kernel'.

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.35

35

2016 DOCKER SURVEY

- Docker application containers
  - Leading containerization vehicle

The infographic features a blue whale carrying a stack of blue containers on its back, with a red crane lifting a container. To the right, three clouds also carry stacks of containers, with red arrows indicating movement between them. The infographic includes the following statistics:

- 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

The Docker logo is in the bottom right corner of the infographic.

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.36

36

# 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

The diagram illustrates the Docker Client-Server Architecture. On the left, three user icons represent Docker Clients. Arrows point from these clients to a central gear icon labeled 'Docker Daemon'. From the Docker Daemon, three arrows point to server icons labeled 'Docker Containers'. The entire diagram is captioned 'Docker Client-Server Architecture'.

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

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.37

37

# 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

The diagram shows the stack of the original Docker engine. At the top is a black box labeled '\$Docker client'. A double-headed arrow connects it to a white box labeled 'dockerd'. Another double-headed arrow connects 'dockerd' to a white box labeled 'LXC'. Below 'LXC', a double-headed arrow connects to a large white box labeled 'Host Kernel'. Inside the 'Host Kernel' box, there are three smaller boxes: 'Namespaces', 'Capabilities', and 'cgroups'.

November 18, 2021

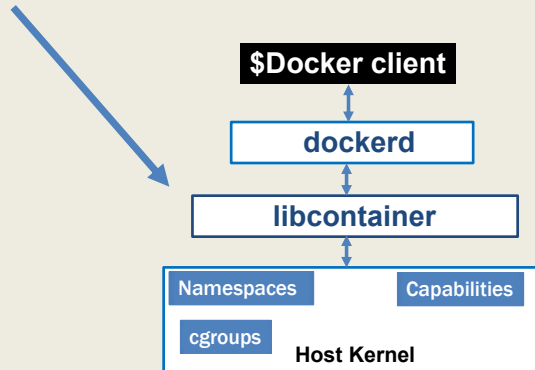
TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.38

38

## INTRODUCTION OF LIBCONTAINER

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



November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.39

39

## 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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.40

40

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 !!!

```
graph TD
    client["$ Docker client"] <--> dockerd
    dockerd <--> containerd
    containerd <--> shim
    shim <--> runc
    runc <--> kernel[Host Kernel]
    subgraph HostKernel [Host Kernel]
        namespaces[Namespaces]
        capabilities[Capabilities]
        cgroups[cgroups]
    end
```

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.41

41

CREATING A CONTAINER - 2

```
graph LR
    cli[Docker CLI/UI] --> engine[Docker Engine]
    engine --> containerd[Containerd]
    containerd --> runtimes[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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.42

42

## 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; libcontainer --> Linux; libvirt --> Linux; lxc --> Linux; systemd_nspawn --> Linux; subgraph Linux; cgroups; namespaces; netlink; selinux; netfilter; capabilities; apparmor; end;
```

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.43

43

## 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, 2021

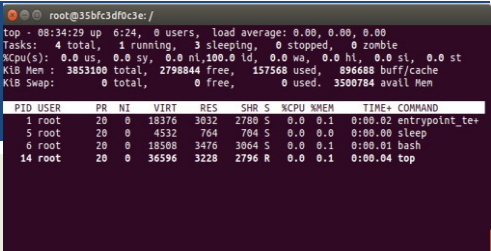
TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.44

44

## 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



The screenshot shows a terminal window with the prompt 'root@35bfc3df0c3e: /'. It displays the output of the 'top' command, showing system statistics and a list of processes. The statistics include: 08:34:29 up 6:24, 0 users, load average: 0.00, 0.00, 0.00; Tasks: 4 total, 1 running, 3 sleeping, 0 stopped, 0 zombie; 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st; Kib Mem: 3853100 total, 2798844 free, 157568 used, 896688 buff/cache; Kib Swap: 0 total, 0 free, 0 used, 3500784 avail Mem. The process list shows: PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND, with entries for root processes 1 (entrypoint\_te), 5 (sleep), 6 (bash), and 14 (top).

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.45

45

## 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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.46

46

WE WILL RETURN AT  
~6:16 PM



47

CGROUPS - 2

- Control groups are hierarchical
- Groups inherit limits from parent groups
- Linux has multiple cgroup controllers (subsystems)
- Is /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

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.48

48



OVERLAY FILE SYSTEMS

- Docker leverages overlay filesystems
- 1<sup>st</sup>: 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/>

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.49

49

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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.50

50

# THREE-TIER ARCHITECTURE

- Node.js
- Postgres
- Nginx

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

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, 2021

TCCS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.51

51

# CONTAINER ISOLATION

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

Application containers

App

Bins/lib

App

Bins/lib

Container runtime

VM kernel

Host kernel

Application containers

App

Bins/lib

App

Bins/lib

Container runtime

Host kernel

November 18, 2021

TCCS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.52

52

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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.53

53

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

```
graph TD;
    DE[Docker Engine] --> C[containerd];
    C --> CS1[containerd-shim];
    C --> CS2[containerd-shim];
    C --> CS3[...];
    CS1 --> R1[runC];
    CS2 --> R2[runC];
    CS3 --> R3[...];
```

November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.54

54

## 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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.55

55

## 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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.56

56

## 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, 2021	TCSS562: Software Engineering for Cloud Computing [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L14.57
-------------------	--	--------

57

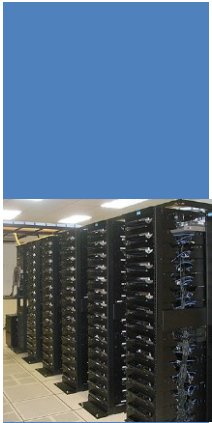
## WE WILL RETURN AT ~7:05PM



58

# TUTORIAL #7

## DOCKER, CGROUPS, RESOURCE ISOLATION



November 18, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.59

59

# 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, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.60

60

Commands:

attach

Build an image from a Dockerfile

build

Create a new image from a container's changes

commit

Copy files/folders between a container and the local filesystem

cp

Create a new container

create

Deploy a new stack or update an existing stack

deploy

Inspect changes to files or directories on a container's filesystem

diff

Get real time events from the server

events

Run a command in a running container

exec

Export a container's filesystem as a tar archive

export

Show the history of an image

history

List images

images

Import the contents from a tarball to create a filesystem image

import

Display system-wide information

info

Return low-level information on Docker objects

inspect

Kill one or more running containers

kill

Load an image from a tar archive or STDIN

load

Log in to a Docker registry

login

Log out from a Docker registry

logout

Fetch the logs of a container

logs

Pause all processes within one or more containers

pause

List port mappings or a specific mapping for the container

port

List containers

ps

Pull an image or a repository from a registry

pull

Push an image or a repository to a registry

push

Rename a container

rename

Restart one or more containers

restart

Remove one or more containers

rm

Remove one or more images

rmi

Run a command in a new container

run

Save one or more images to a tar archive (streamed to STDOUT by default)

save

Search the Docker Hub for images

search

Start one or more stopped containers

start

Display a live stream of container(s) resource usage statistics

stats

Stop one or more running containers

stop

Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

tag

Display the running processes of a container

top

Unpause all processes within one or more containers

unpause

Update configuration of one or more containers

update

Show the Docker version information

version

Block until one or more containers stop, then print their exit codes

wait

Docker CLI

61

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, 2021

TCCS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L14.62

62

# QUESTIONS-



November 16, 2021

TCSS562: Software Engineering for Cloud Computing [Fall 2021]  
School of Engineering and Technology, University of Washington - Tacoma

L13.63