TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

Cloud Enabling Technology IV & Containerization

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

TR 5:50-7:50 PM

1

OFFICE HOURS – FALL 2022

- **THIS WEEK**
- **Tuesday:**
  - 4:30 to 5:30 pm - CP 229 and Zoom
- **Friday**
  - 12:00 to 1:00 pm - CP 229 and Zoom
- Or email for appointment

> Office Hours set based on Student Demographics survey feedback

2

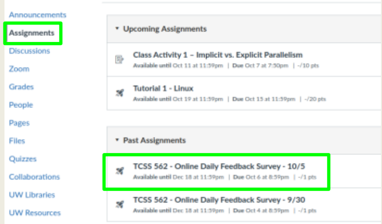
OBJECTIVES – 11/15

- **Questions from 11/10**
- Tutorials Questions
- Class Presentations: Cloud Technology or Research Paper Review
- Quiz 1
- Ch. 5: Cloud Enabling Technology
- Tutorial 7
- Containerization

3

ONLINE DAILY FEEDBACK SURVEY

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



▾ Upcoming Assignments

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

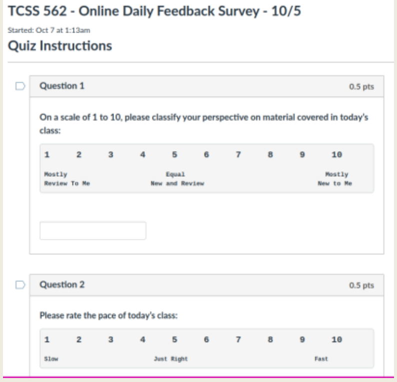
Tutorial 1 - Linux
Available until Oct 19 at 11:59pm | Due Oct 15 at 11:59pm | -/20 pts

▾ 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

4



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                     Equal                              Mostly
Review To Me           New and Review                      New to Me
```

Question 2 0.5 pts

Please rate the pace of today's class:

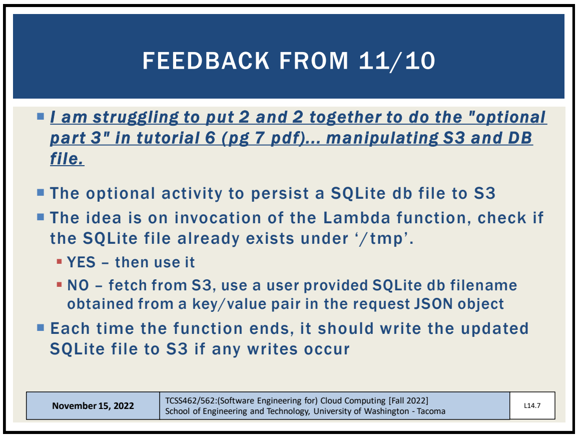
```
1      2      3      4      5      6      7      8      9      10
Slow                    Just Right                         Fast
```

5

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (**43** respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.77 (↑ - *previous* 6.38)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.40 (↑ - previous 5.38)**
- **Response rates:**
- TCSS 462: 22/33 – 66.67%
- TCSS 562: 21/26 – 80.77%

6

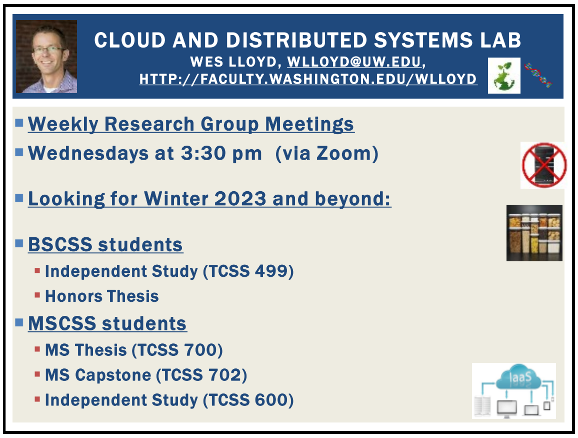


FEEDBACK FROM 11/10

- ***I am struggling to put 2 and 2 together to do the "optional part 3" in tutorial 6 (pg 7 pdf)... manipulating S3 and DB file.***
- The optional activity to persist a SQLite db file to S3
- The idea is on invocation of the Lambda function, check if the SQLite file already exists under '/tmp'.
  - YES – then use it
  - NO – fetch from S3, use a user provided SQLite db filename obtained from a key/value pair in the request JSON object
- Each time the function ends, it should write the updated SQLite file to S3 if any writes occur

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022]
School of Engineering and Technology, University of Washington - Tacoma | L14.7

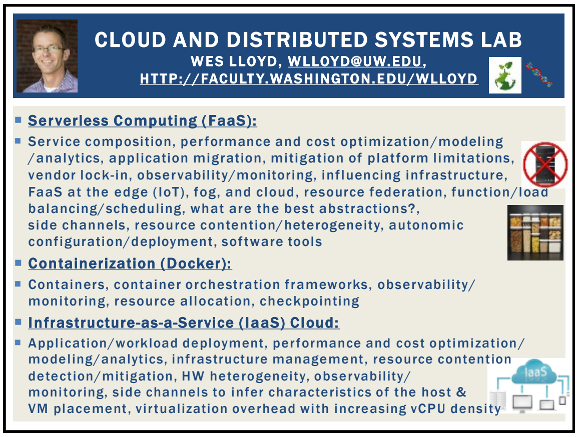
7



CLOUD AND DISTRIBUTED SYSTEMS LAB
WES LLOYD, WLLOYD@UW.EDU, HTTP://FACULTY.WASHINGTON.EDU/WLLOYD

- **Weekly Research Group Meetings**
- **Wednesdays at 3:30 pm (via Zoom)**
- **Looking for Winter 2023 and beyond:**
- **BSCSS students**
  - Independent Study (TCSS 499)
  - Honors Thesis
- **MSCSS students**
  - MS Thesis (TCSS 700)
  - MS Capstone (TCSS 702)
  - Independent Study (TCSS 600)

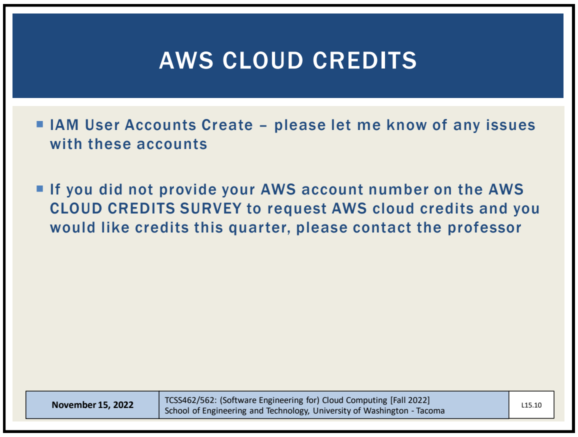
8



CLOUD AND DISTRIBUTED SYSTEMS LAB
WES LLOYD, WLLOYD@UW.EDU, HTTP://FACULTY.WASHINGTON.EDU/WLLOYD

- **Serverless Computing (FaaS):**
- Service composition, performance and cost optimization/modeling /analytics, application migration, mitigation of platform limitations, vendor lock-in, observability/monitoring, influencing infrastructure, FaaS at the edge (IoT), fog, and cloud, resource federation, function/load balancing/scheduling, what are the best abstractions?, side channels, resource contention/heterogeneity, autonomic configuration/deployment, software tools
- **Containerization (Docker):**
- Containers, container orchestration frameworks, observability/ monitoring, resource allocation, checkpointing
- **Infrastructure-as-a-Service (IaaS) Cloud:**
- Application/workload deployment, performance and cost optimization/ modeling/analytics, infrastructure management, resource contention detection/mitigation, HW heterogeneity, observability/ monitoring, side channels to infer characteristics of the host & VM placement, virtualization overhead with increasing vCPU density

9

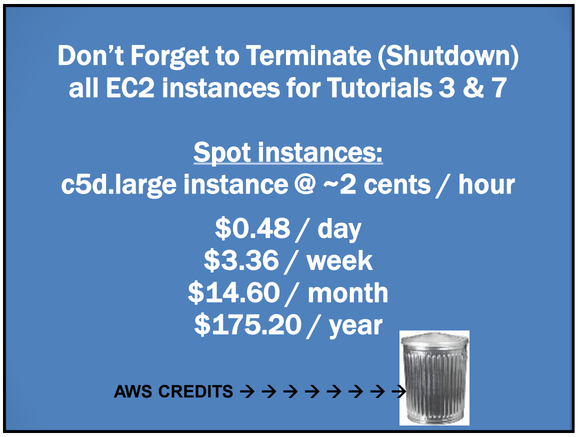


AWS CLOUD CREDITS

- IAM User Accounts Create – please let me know of any issues with these accounts
- If you did not provide your AWS account number on the AWS CLOUD CREDITS SURVEY to request AWS cloud credits and you would like credits this quarter, please contact the professor

November 15, 2022 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2022]
School of Engineering and Technology, University of Washington - Tacoma | L15.10

10



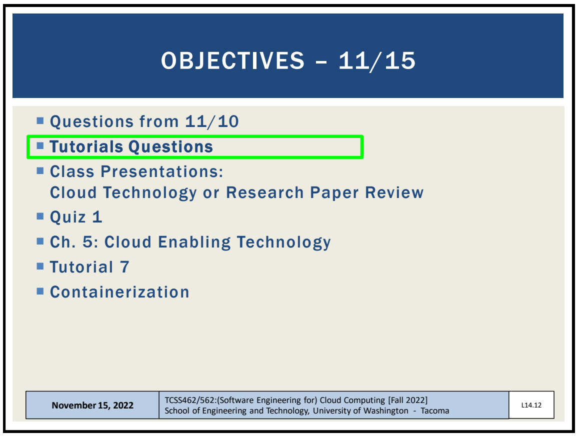
Don't Forget to Terminate (Shutdown) all EC2 instances for Tutorials 3 & 7

Spot instances:
c5d.large instance @ ~2 cents / hour

$0.48 / day
$3.36 / week
$14.60 / month
$175.20 / year

AWS CREDITS → → → → → → → →

11

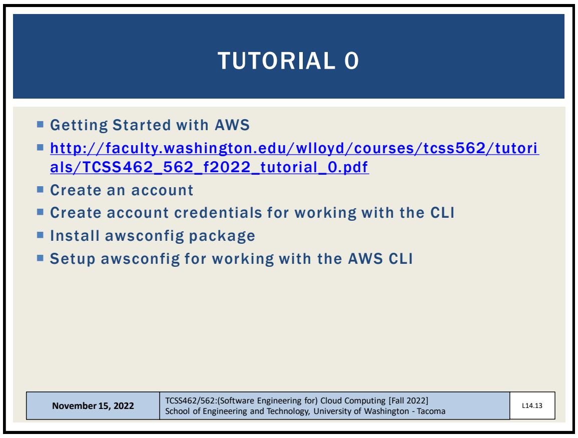


OBJECTIVES – 11/15

- Questions from 11/10
- **Tutorials Questions**
- Class Presentations:
  Cloud Technology or Research Paper Review
- Quiz 1
- Ch. 5: Cloud Enabling Technology
- Tutorial 7
- Containerization

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022]
School of Engineering and Technology, University of Washington - Tacoma | L14.12

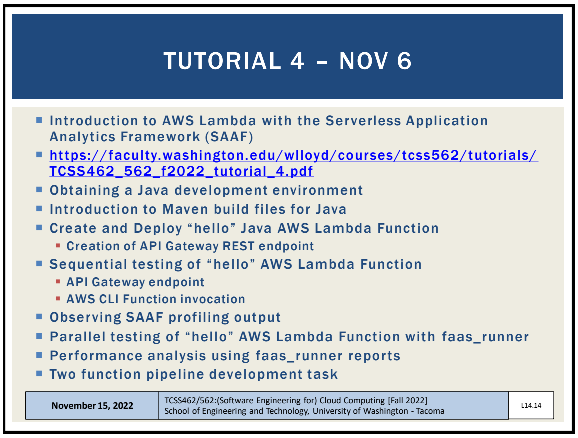
12



TUTORIAL 0

- Getting Started with AWS
- http://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_0.pdf
- Create an account
- Create account credentials for working with the CLI
- Install awsconfig package
- Setup awsconfig for working with the AWS CLI

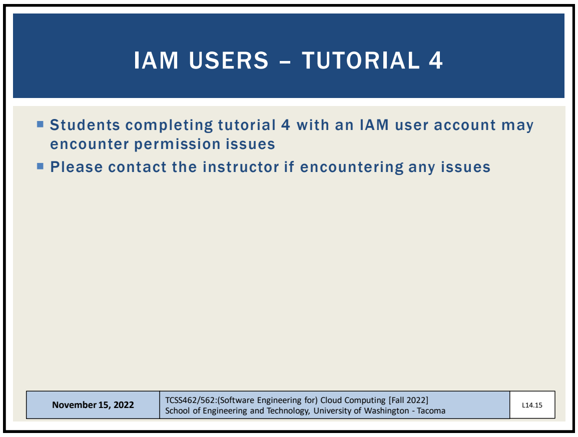
13



TUTORIAL 4 – NOV 6

- Introduction to AWS Lambda with the Serverless Application Analytics Framework (SAAF)
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_4.pdf
- Obtaining a Java development environment
- Introduction to Maven build files for Java
- Create and Deploy “hello” Java AWS Lambda Function
  - Creation of API Gateway REST endpoint
- Sequential testing of “hello” AWS Lambda Function
  - API Gateway endpoint
  - AWS CLI Function invocation
- Observing SAAF profiling output
- Parallel testing of “hello” AWS Lambda Function with faas_runner
- Performance analysis using faas_runner reports
- Two function pipeline development task

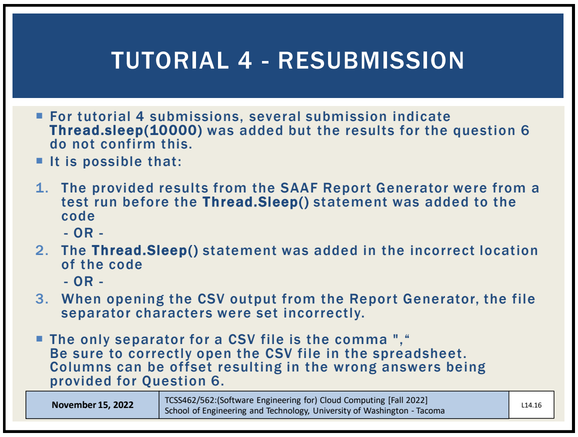
14



IAM USERS – TUTORIAL 4

- Students completing tutorial 4 with an IAM user account may encounter permission issues
- Please contact the instructor if encountering any issues

15



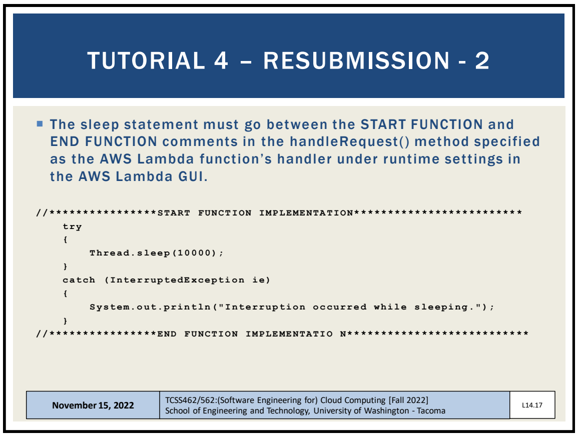
TUTORIAL 4 - RESUBMISSION

- For tutorial 4 submissions, several submission indicate **Thread.sleep(10000)** was added but the results for the question 6 do not confirm this.
- It is possible that:

1. The provided results from the SAAF Report Generator were from a test run before the **Thread.Sleep()** statement was added to the code
   - OR -
2. The **Thread.Sleep()** statement was added in the incorrect location of the code
   - OR -
3. When opening the CSV output from the Report Generator, the file separator characters were set incorrectly.

- The only separator for a CSV file is the comma ",“
  Be sure to correctly open the CSV file in the spreadsheet.
  Columns can be offset resulting in the wrong answers being provided for Question 6.

16

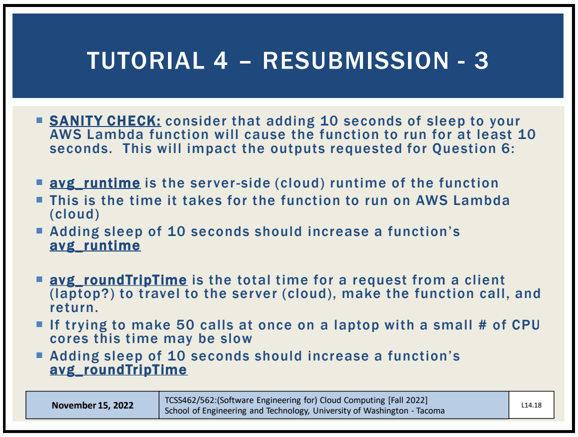


TUTORIAL 4 – RESUBMISSION - 2

- The sleep statement must go between the START FUNCTION and END FUNCTION comments in the handleRequest() method specified as the AWS Lambda function’s handler under runtime settings in the AWS Lambda GUI.

```
//****************START FUNCTION IMPLEMENTATION*************************
    try
    {
        Thread.sleep(10000);
    }
    catch (InterruptedException ie)
    {
        System.out.println("Interruption occurred while sleeping.");
    }
//****************END FUNCTION IMPLEMENTATIO N***************************
```

17



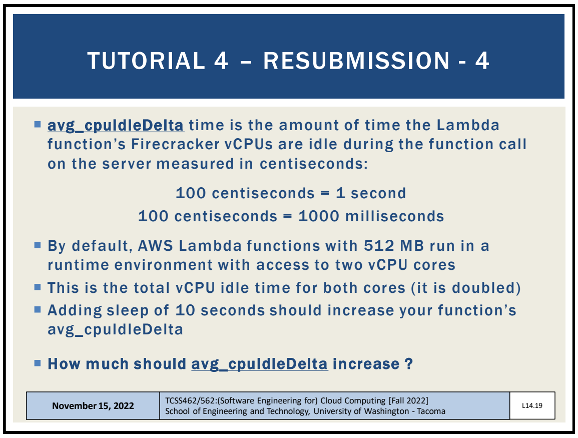
TUTORIAL 4 – RESUBMISSION - 3

- **SANITY CHECK:** consider that adding 10 seconds of sleep to your AWS Lambda function will cause the function to run for at least 10 seconds. This will impact the outputs requested for Question 6:

- **avg_runtime** is the server-side (cloud) runtime of the function
- This is the time it takes for the function to run on AWS Lambda (cloud)
- Adding sleep of 10 seconds should increase a function’s **avg_runtime**

- **avg_roundTripTime** is the total time for a request from a client (laptop?) to travel to the server (cloud), make the function call, and return.
- If trying to make 50 calls at once on a laptop with a small # of CPU cores this time may be slow
- Adding sleep of 10 seconds should increase a function’s **avg_roundTripTime**

18

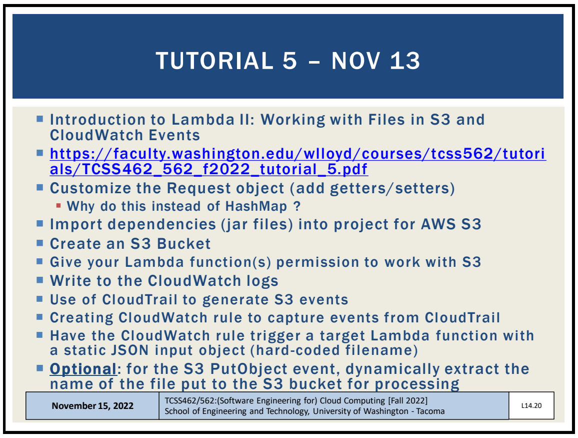


TUTORIAL 4 – RESUBMISSION - 4

- **avg_cpuIdleDelta** time is the amount of time the Lambda function's Firecracker vCPUs are idle during the function call on the server measured in centiseconds:

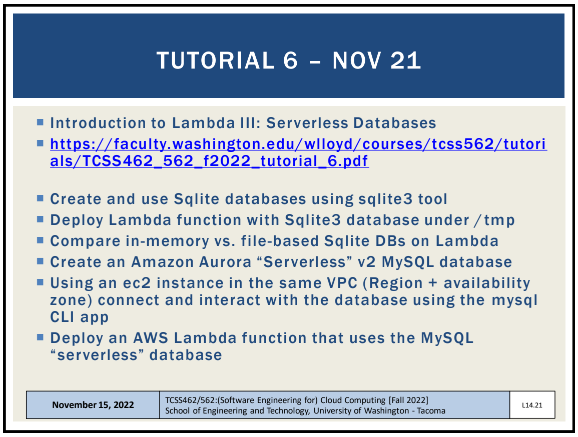
  100 centiseconds = 1 second
  100 centiseconds = 1000 milliseconds

- By default, AWS Lambda functions with 512 MB run in a runtime environment with access to two vCPU cores
- This is the total vCPU idle time for both cores (it is doubled)
- Adding sleep of 10 seconds should increase your function's avg_cpuIdleDelta
- **How much should avg_cpuIdleDelta increase ?**



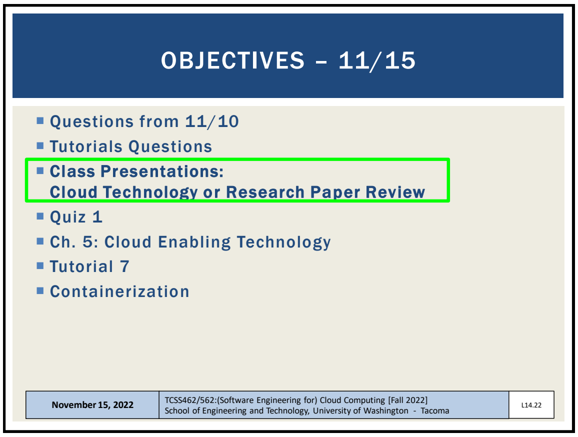
TUTORIAL 5 – NOV 13

- Introduction to Lambda II: Working with Files in S3 and CloudWatch Events
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_5.pdf
- Customize the Request object (add getters/setters)
  - Why do this instead of HashMap ?
- Import dependencies (jar files) into project for AWS S3
- Create an S3 Bucket
- Give your Lambda function(s) permission to work with S3
- Write to the CloudWatch logs
- Use of CloudTrail to generate S3 events
- Creating CloudWatch rule to capture events from CloudTrail
- Have the CloudWatch rule trigger a target Lambda function with a static JSON input object (hard-coded filename)
- **Optional**: for the S3 PutObject event, dynamically extract the name of the file put to the S3 bucket for processing



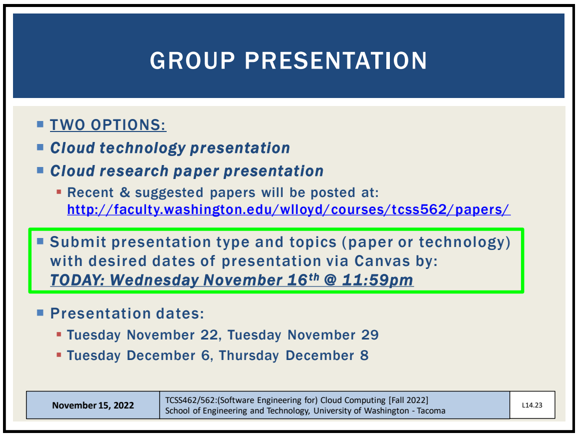
TUTORIAL 6 – NOV 21

- Introduction to Lambda III: Serverless Databases
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_6.pdf
- Create and use Sqlite databases using sqlite3 tool
- Deploy Lambda function with Sqlite3 database under /tmp
- Compare in-memory vs. file-based Sqlite DBs on Lambda
- Create an Amazon Aurora "Serverless" v2 MySQL database
- Using an ec2 instance in the same VPC (Region + availability zone) connect and interact with the database using the mysql CLI app
- Deploy an AWS Lambda function that uses the MySQL "serverless" database



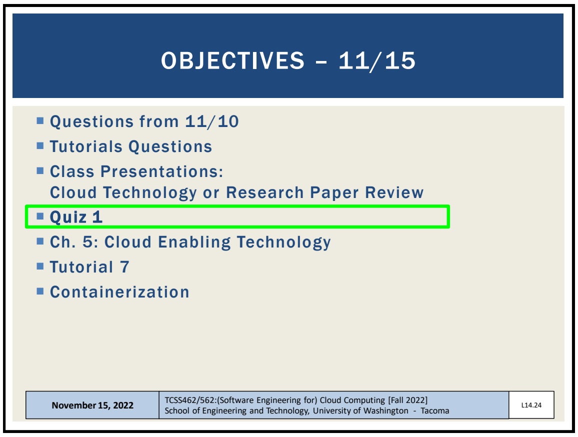
OBJECTIVES – 11/15

- Questions from 11/10
- Tutorials Questions
- **Class Presentations: Cloud Technology or Research Paper Review**
- Quiz 1
- Ch. 5: Cloud Enabling Technology
- Tutorial 7
- Containerization



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: ***TODAY: Wednesday November 16th @ 11:59pm***
- Presentation dates:
  - Tuesday November 22, Tuesday November 29
  - Tuesday December 6, Thursday December 8



OBJECTIVES – 11/15

- Questions from 11/10
- Tutorials Questions
- Class Presentations: Cloud Technology or Research Paper Review
- **Quiz 1**
- Ch. 5: Cloud Enabling Technology
- Tutorial 7
- Containerization

QUIZ 1

- Opened **Monday Nov 14 at 8:00 am**
- Closes **Friday November 18 at 11:59 am**
- Individual work only
- Please answer every question
- Book, notes, slides, calculator, and internet are allowed

- **Grading:**
- The Canvas autograder produces a preliminary score, not the final score.
- The instructor will manually review all quizzes and add partial credit
- A curve adjustment will also be applied as appropriate
- These updates may not occur until several days after the quiz closes
- Please report suspected grading problems to the instructor

- **Attempts:**
- 1 quiz attempt, 120 minute limit, 20 questions.
- Coverage is inclusive of Lectures ~1-8
- Please plan accordingly. Once started, there will be 2 hours to complete

OBJECTIVES – 11/15

- Questions from 11/10
- Tutorials Questions
- Class Presentations:
  Cloud Technology or Research Paper Review
- Quiz 1
- **Ch. 5: Cloud Enabling Technology**
- Tutorial 7
- Containerization

CLOUD ENABLING TECHNOLOGY

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

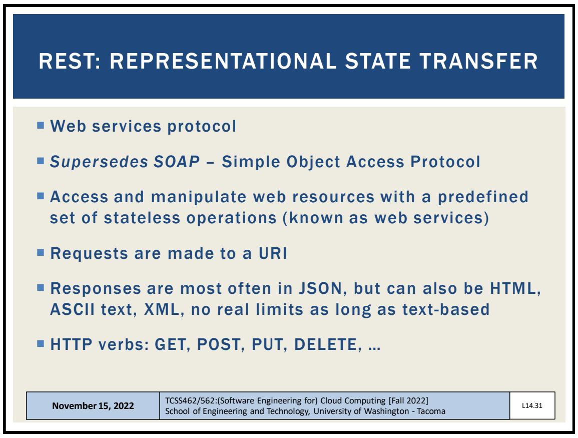
5. WEB SERVICES/WEB

- 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

HYPERTEXT TRANSPORT PROTOCOL (HTTP)

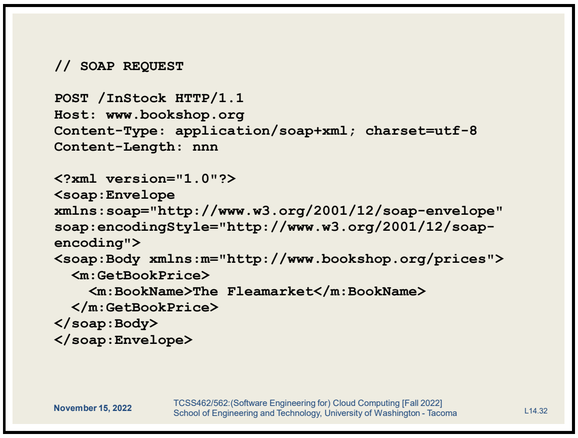
- An ASCII-based request/reply protocol for transferring information on the web
- HTTP request includes:
  - request method (GET, POST, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - headers—extra info regarding transfer request
- HTTP response from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

HTTP status codes:
2xx — *all is well*
3xx — *resource moved*
4xx — *access problem*
5xx — *server error*

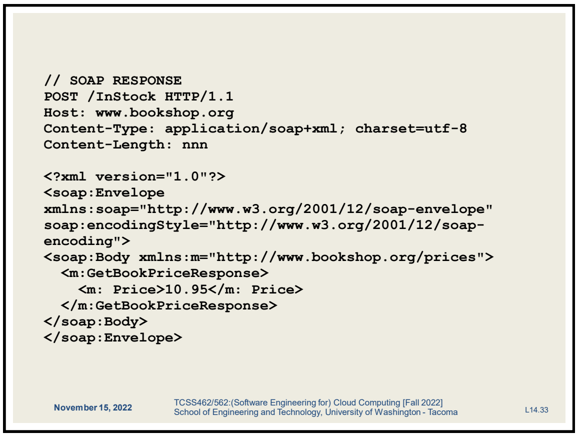


REST: REPRESENTATIONAL STATE TRANSFER

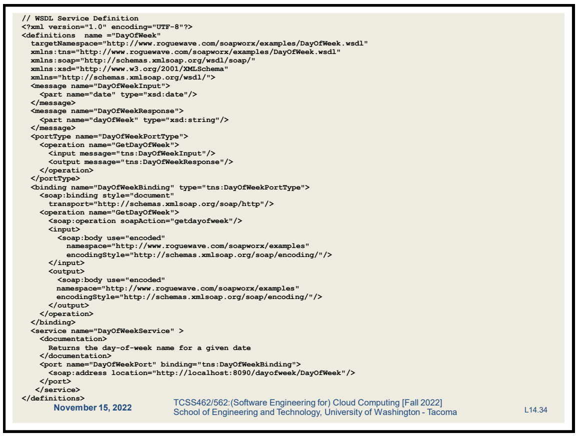
- Web services protocol
- *Supersedes SOAP* – Simple Object Access Protocol
- Access and manipulate web resources with a predefined set of stateless operations (known as web services)
- Requests are made to a URI
- Responses are most often in JSON, but can also be HTML, ASCII text, XML, no real limits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



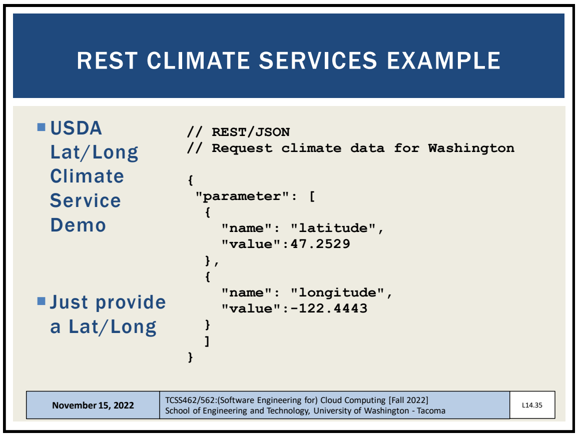
```
// SOAP REQUEST

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>
```



```
// 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>
```



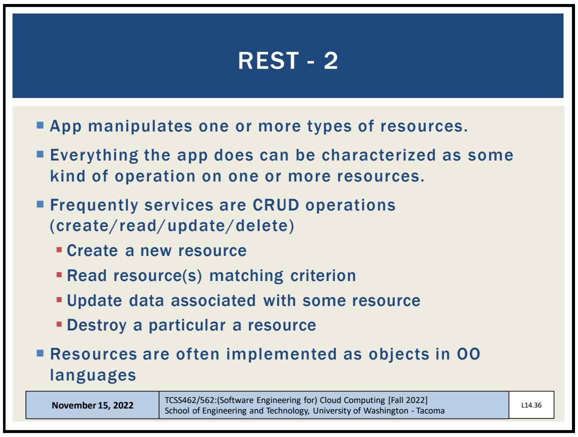
```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions  name ="DayOfWeek"
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="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>
```



REST CLIMATE SERVICES EXAMPLE

- USDA Lat/Long Climate Service Demo
- Just provide a Lat/Long

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

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



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

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)
- **Visibility**: of communication between services
- **Portability**: of services by redeployment
- **Reliability**: resists failure at the system level as redundancy of infrastructure is easy to ensure

OBJECTIVES – 11/15

- Questions from 11/10
- Tutorials Questions
- Class Presentations:
  Cloud Technology or Research Paper Review
- Quiz 1
- Ch. 5: Cloud Enabling Technology
- **Tutorial 7**
- Containerization

TUTORIAL #7

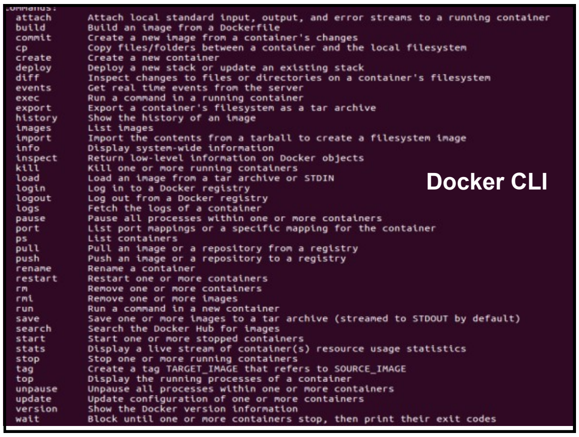
DOCKER, CGROUPS, RESOURCE ISOLATION

TUTORIAL 7 – DEC 5

- Introduction to Docker
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_7.pdf
- Must complete using Ubuntu 22.04 (for cgroups v2)
- Use docx file for copying and pasting Docker install commands
- Installing Docker
- Creating a container using a Dockerfile
- Using cgroups virtual filesystem to monitor CPU utilization of a container
- Persisting container images to Docker Hub image repository
- Container vertical scaling of CPU/memory resources
- Testing container CPU and memory isolation

TUTORIAL COVERAGE

- Docker CLI → Docker Engine (dockerd) → containerd → runc
- Working with the docker CLI:
- docker run   create a container
- docker ps -a   list containers, find CONTAINER ID
- docker exec --it   run a process in an existing container
- docker stop   stop a container
- docker kill   kill a container
- docker help   list available commands
- man docker   Docker Linux manual pages



```
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
```

Docker CLI

TUTORIAL 7

- Tutorial introduces use of two common Linux performance benchmark applications
- stress-ng
- 100s of CPU, memory, disk, network stress tests
- Sysbench
- Used in tutorial for memory stress test

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.43

43

OBJECTIVES – 11/15

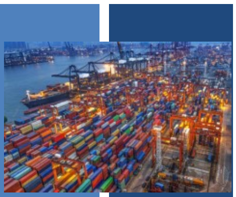
- Questions from 11/10
- Tutorials Questions
- Class Presentations:
  Cloud Technology or Research Paper Review
- Quiz 1
- Ch. 5: Cloud Enabling Technology
- Tutorial 7
- **Containerization**

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.44

44

WE WILL RETURN AT ~7:15 PM

45



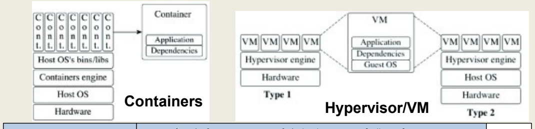
CONTAINERIZATION

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.46

46

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 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.47

47

CONTAINER PERFORMANCE – LU FACTORIZATION PERFORMANCE

- Solve linear equations – matrix algebra

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

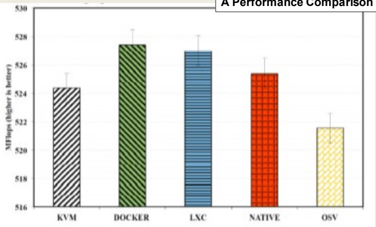
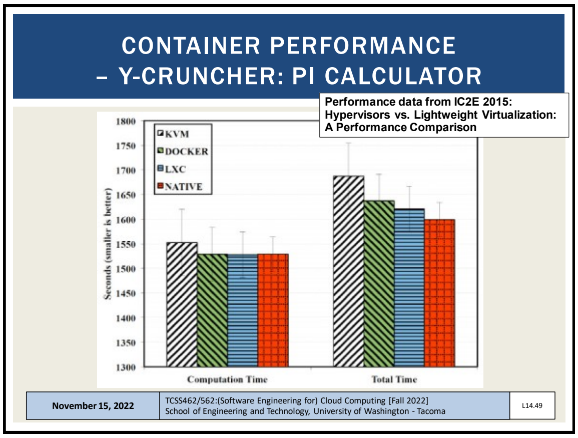


Fig. 4. The value of Linpack results on each platform over 15 runs. This is the particular case of N=1000.

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.48

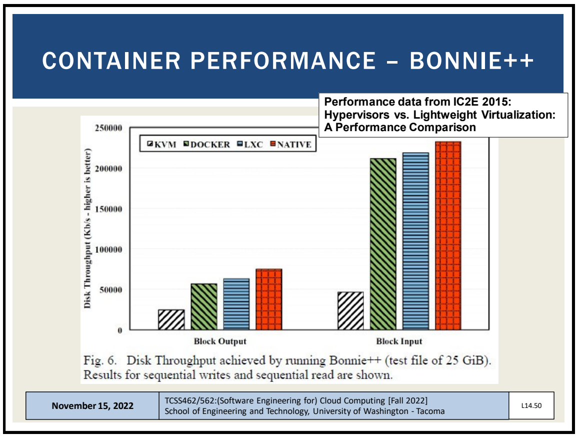
48



CONTAINER PERFORMANCE – Y-CRUNCHER: PI CALCULATOR

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

49

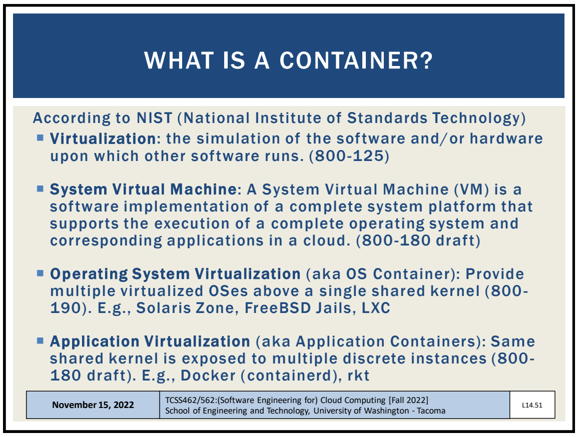


CONTAINER PERFORMANCE – BONNIE++

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.

50

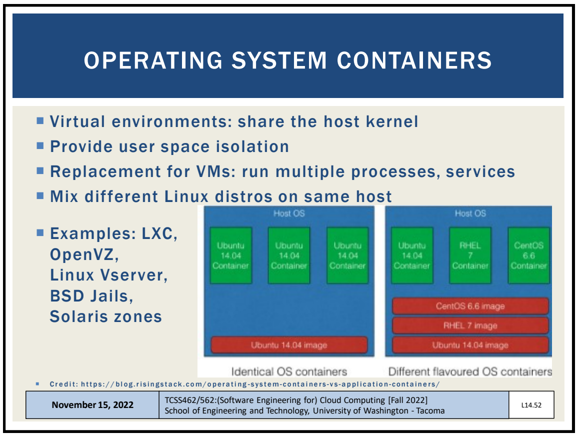


WHAT IS A CONTAINER?

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

51

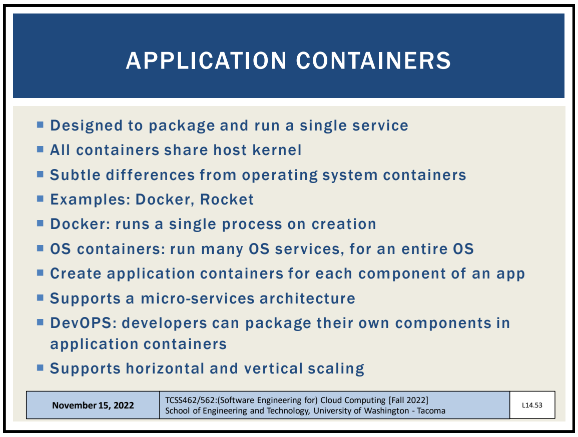


OPERATING SYSTEM CONTAINERS

- 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/

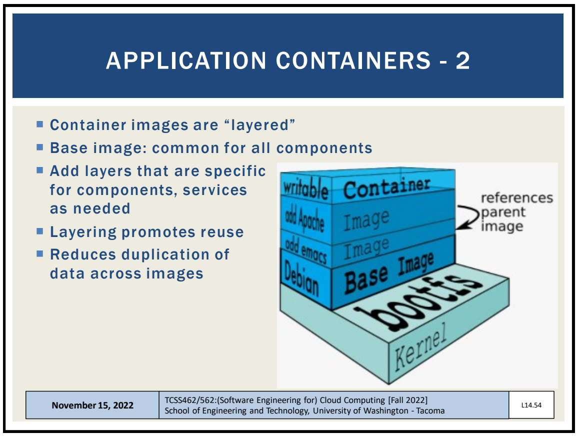
52



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

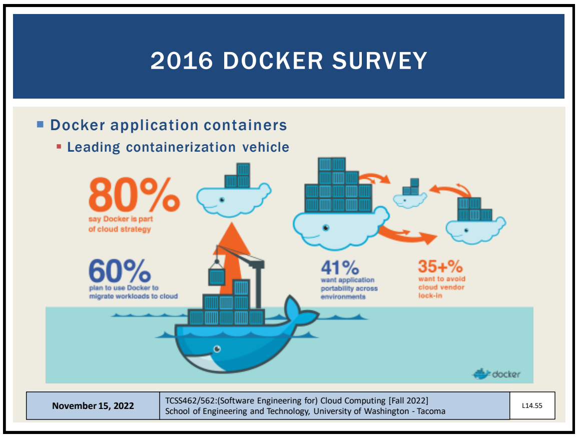
53



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

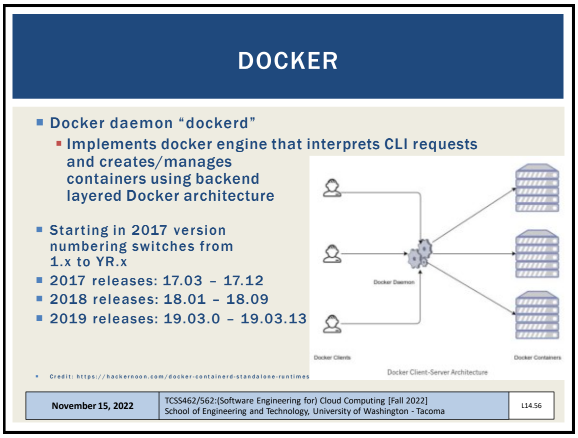
54



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
docker

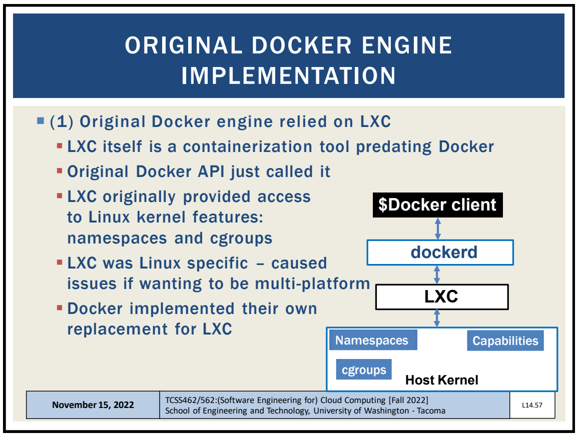


DOCKER

- 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

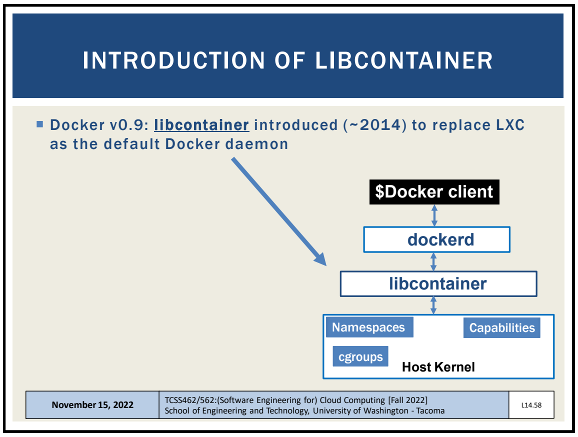
Docker Client-Server Architecture

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



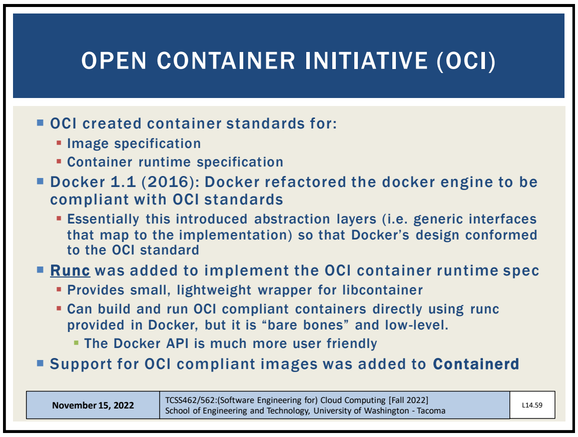
ORIGINAL DOCKER ENGINE IMPLEMENTATION

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



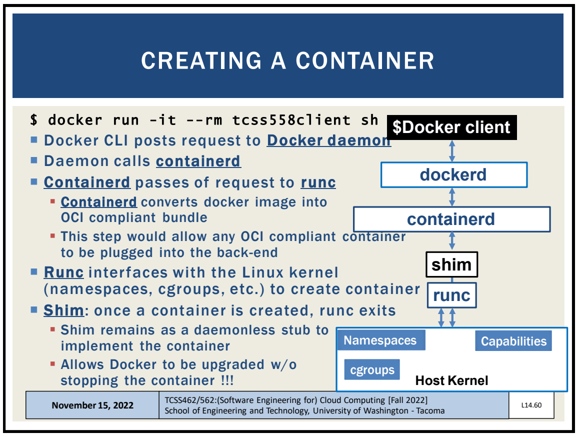
INTRODUCTION OF LIBCONTAINER

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



OPEN CONTAINER INITIATIVE (OCI)

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

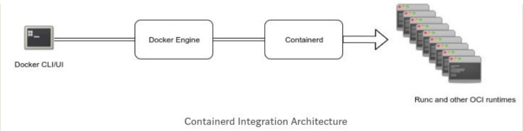


CREATING A CONTAINER

```
$ 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 !!!

CREATING A CONTAINER - 2

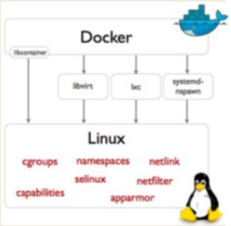


Containerd Integration Architecture

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

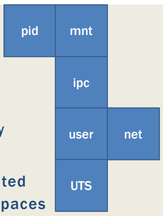
SUPPORT FOR ALTERNATE CONTAINER RUNTIMES

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

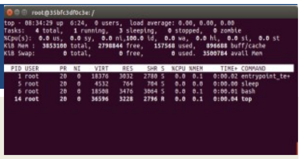


LINUX KERNEL NAMESPACES

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


pid
mnt
ipc
user
net
UTS

NAMESPACES - 2



```
top - 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
%Cpu(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
KiB Mem :  3853100 total,  2798844 free,   157568 used,   896688 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  3500784 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   18376   3032   2780 S   0.0  0.1   0:00.02 entrypoint_te+
    5 root      20   0    4532    764    704 S   0.0  0.0   0:00.00 sleep
    6 root      20   0   18508   3476   3064 S   0.0  0.1   0:00.01 bash
   14 root      20   0   36596   3228   2796 R   0.0  0.1   0:00.04 top
```

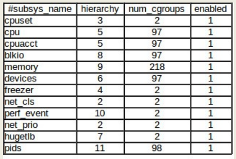
- **Provides isolation of OS entities for containers**
- **mnt**: separate filesystems
- **pid**: independent PIDs; first process in container is PID 1
- **ipc**: prevents processes in different IPC namespaces from being able to establish shared memory. Enables processes in different containers to reuse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

CONTROL GROUPS (CGROUPS)

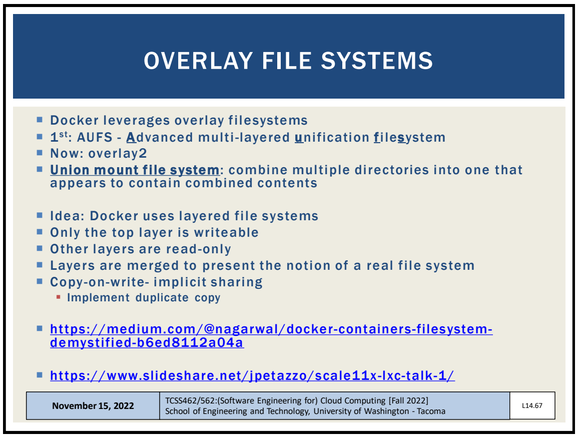
- Collection of Linux processes
- Group-level resource allocation: *CPU, memory, disk I/O, network I/O*
- **Resource limiting**
  - Memory, disk cache
- **Prioritization**
  - CPU share
  - Disk I/O throughput
- **Accounting**
  - Track resource utilization
  - For resource management and/or billing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

CGROUPS - 2

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



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

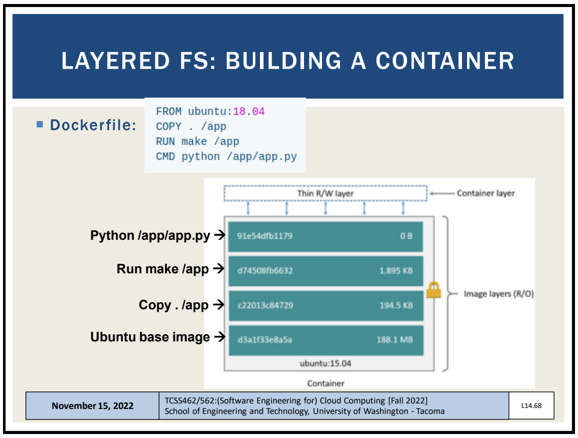


OVERLAY FILE SYSTEMS

- Docker leverages overlay filesystems
- 1st: AUFS - Advanced multi-layered unification filesystem
- Now: overlay2
- **Union mount file system**: combine multiple directories into one that appears to contain combined contents
- Idea: Docker uses layered file systems
- Only the top layer is writeable
- Other layers are read-only
- Layers are merged to present the notion of a real file system
- Copy-on-write- implicit sharing
  - Implement duplicate copy
- https://medium.com/@nagarwal/docker-containers-filesystem-demystified-b6ed8112a04a
- https://www.slideshare.net/jpetazzo/scale11x-lxc-talk-1/

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.67

67



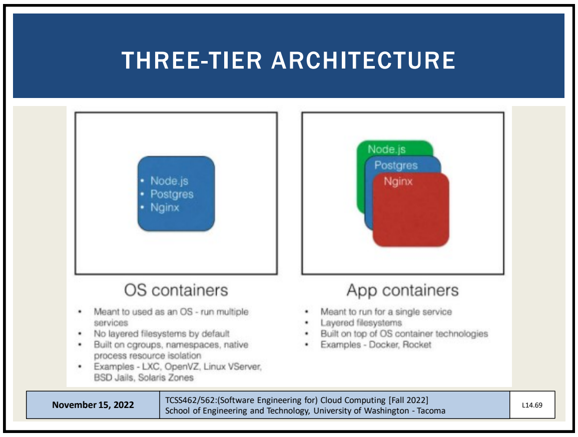
LAYERED FS: BUILDING A CONTAINER

- Dockerfile:

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

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.68

68



THREE-TIER ARCHITECTURE

OS containers

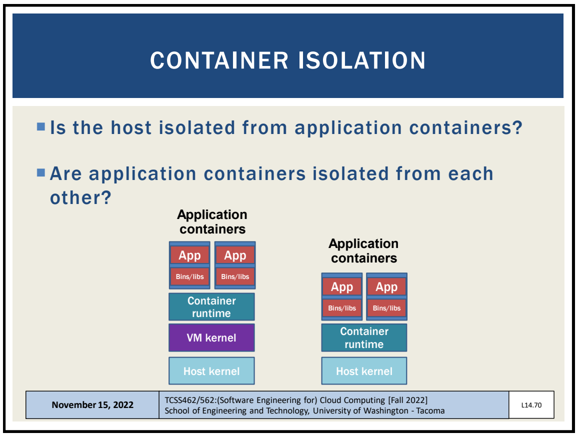
- Meant to used as an OS - run multiple services
- No layered filesystems by default
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, OpenVZ, Linux VServer, BSD Jails, Solaris Zones

App containers

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

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.69

69

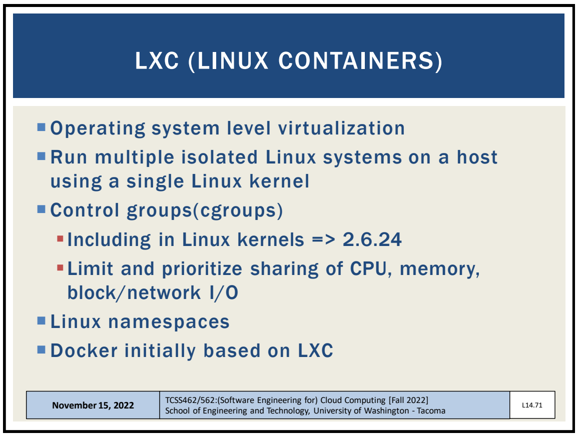


CONTAINER ISOLATION

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

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.70

70

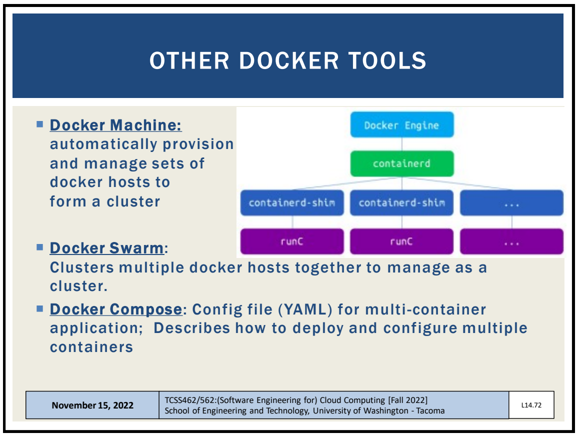


LXC (LINUX CONTAINERS)

- 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 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.71

71

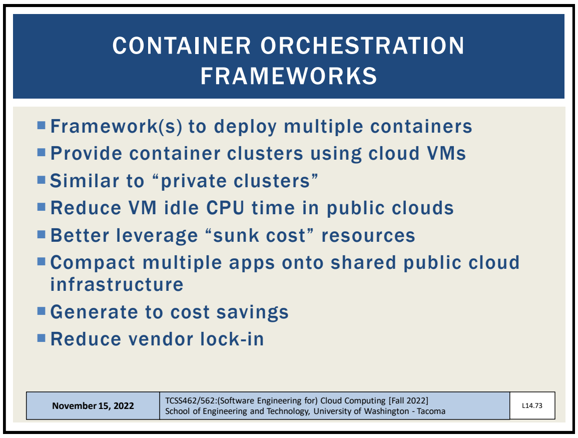


OTHER DOCKER TOOLS

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

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L14.72

72

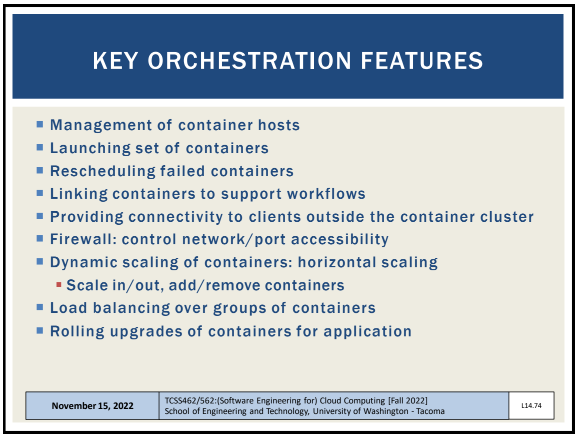


CONTAINER ORCHESTRATION FRAMEWORKS

- 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 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022]
School of Engineering and Technology, University of Washington - Tacoma | L14.73

73

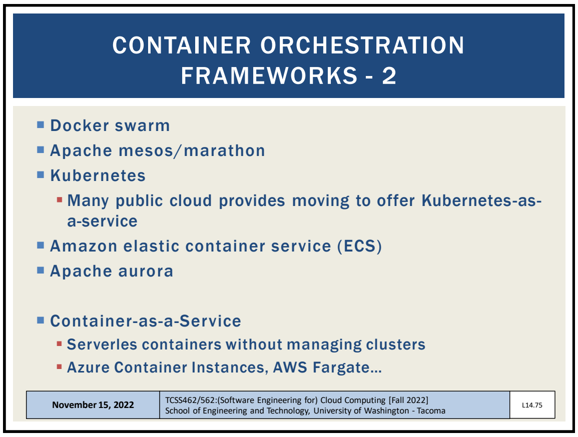


KEY ORCHESTRATION FEATURES

- 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 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022]
School of Engineering and Technology, University of Washington - Tacoma | L14.74

74



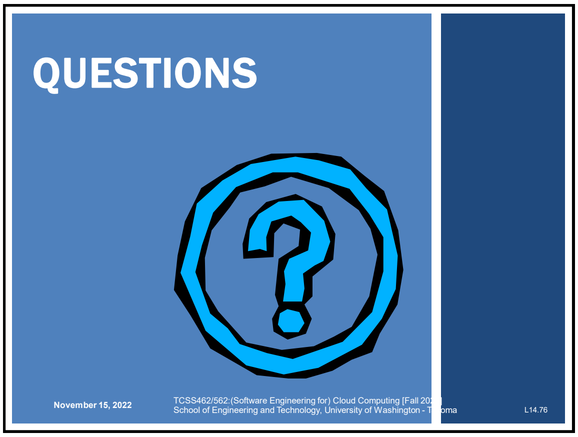
CONTAINER ORCHESTRATION FRAMEWORKS - 2

- 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 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022]
School of Engineering and Technology, University of Washington - Tacoma | L14.75

75



QUESTIONS

November 15, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022]
School of Engineering and Technology, University of Washington - Tacoma | L14.76

76