TCSS 462/562: (SOFTWARE ENGINEERING FOR) CLOUD COMPUTING

Clolud Enabling Technology & Containerization

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

1

OFFICE HOURS – FALL 2025

- **Thursdays:**
  - 6:00 to 7:00 pm - CP 229 & Zoom
- **Friday – *** THIS WEEK *****
  - 11:00 am to 12:00 pm – ONLINE via Zoom
- Or email for appointment

> *Office Hours set based on Student Demographics survey feedback*
> ** - Friday office hours may be adjusted or canceled due meeting conflicts or other obligations. Adjustments will be announced via Canvas.*

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.2

2

OBJECTIVES – 11/20

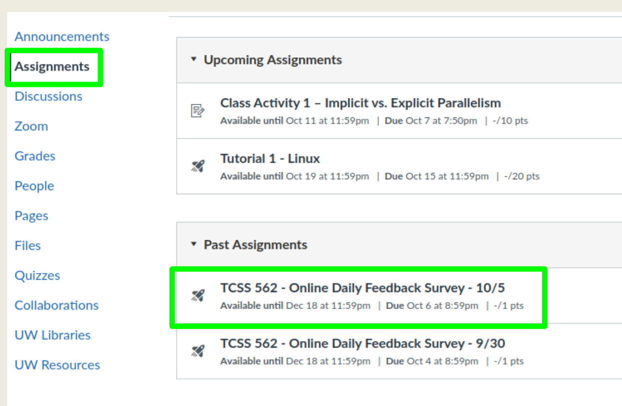
- Questions from 11/18
- Tutorials Questions
- Class Presentations Schedule -
  Cloud Technology or Research Paper Review
- Ch. 5: Cloud Enabling Technology
- Containerization
- Container Profiler

November 20, 2025 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025]
School of Engineering and Technology, University of Washington - Tacoma | L15.3

3

ONLINE DAILY FEEDBACK SURVEY

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



November 20, 2025 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025]
School of Engineering and Technology, University of Washington - Tacoma | L15.4

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 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

5

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (41 respondents, 27 in-person, 14 online):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.46 (↑ - *previous* 5.64)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.29 (↑ - previous 4.96)**

6

FEEDBACK FROM 11/18

- ***Can a Virtual Private Cloud (VPC) span region?***
  We saw it can span availability zone A and B, can it span across regions like us-east-1 and us-east-2 ?
- No. Currently VPCs can only span across multiple availability zones (Azs) within a single region
- This limitation forces deployments to be fully replicated in distinct regions
- If one region fails, application hosting can fail-over to another region
- This often involves Domain-Name-Server (DNS) level load balancing
- Amazon Route 53 is a managed DNS service for this purpose

OBJECTIVES – 11/20

- Questions from 11/18
- **Tutorials Questions**
- Class Presentations Schedule -
  Cloud Technology or Research Paper Review
- Ch. 5: Cloud Enabling Technology
- Containerization
- Container Profiler

TUTORIAL 5 – CLOSING NOV 21 4:59AM

- Introduction to Lambda II: Working with Files in S3, Cloud Trail, and Amazon Event Bridge Rules
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2025_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 Event Bridge rule to capture events from CloudTrail
- Have the Event Bridge rule trigger a 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 23

- Introduction to Lambda III: Serverless Databases
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2025_tutorial_6.pdf
- Create and use Sqlite databases using sqlite3
- 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 the AWS CloudShell in the same VPC (Region + availability zone) connect and interact your Aurora serverless database using the mysql CLI app
- Deploy an AWS Lambda function that uses the MySQL “serverless” database

TUTORIAL 7 – DEC 4

- Introduction to Docker
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2025_tutorial_7.pdf
- Must complete using c7i-flex.large ec2 instance & Ubuntu 24.04 (for cgroups v2)
- Use DOCX file for copying and pasting Docker install commands
- Topics:
  - 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

13

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 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.14

14

OBJECTIVES – 11/20

- Questions from 11/18
- Tutorials Questions
- **Class Presentations Schedule - Cloud Technology or Research Paper Review**
- Ch. 5: Cloud Enabling Technology
- Containerization
- Container Profiler

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.15

15

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: ***Tuesday November 18th @ 11:59pm***
- Presentation dates:
  - Tuesday November 25
  - Tuesday December 2*, Thursday December 4
    - ** - day of quiz 2. only 1 presentation slot*

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.16

16

PRESENTATION SCHEDULE

- **<Tuesday November 25>**

1. Team 4: Xiaoling Wei, Bohan Xiong, Xu Zhu
Research paper: ***Serverless Replication of Object Storage across Multi-Vendor Clouds and Regions***
2. Team 1: William Hay
Cloud Technology: ***Amazon Athena***

- **<Tuesday December 2>**

1. Team 5: Sparsha Jha, Chris Biju
Cloud Technology: ***Intelligent Optimization of Distributed Pipeline Execution in Serverless Platforms: A Predictive Model Approach***

PRESENTATION SCHEDULE - 2

- **<Thursday December 4>**

1. Team 3: Jiameng Li, Naomi Nottingham, Headley Brissett
Research paper: ***A Perfect Fit? – Towards Containers on Microkernels***
2. Team 2: Ruby Plangphatthanaphanit, Junjia Li, Ari Yin
Cloud Technology: ***CI/CD in the Cloud (GitHub Actions + Cloud Deploy)***
3. Team 8: Aamena Suzzane, Dhruva Bhat
Research paper: ***CoFaaS: Automatic Transformation-based Consolidation of Serverless Functions***
4. Team 6: Han Zhang, Sahil Bhatt, Pengcheng Cao
Cloud Technology: ***AWS Amplify***

OBJECTIVES – 11/20

- Questions from 11/18
- Tutorials Questions
- Class Presentations Schedule -
  Cloud Technology or Research Paper Review
- **Ch. 5: Cloud Enabling Technology**
- Containerization
- Container Profiler

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.19

19

CLOUD ENABLING TECHNOLOGY

- Broadband networks and internet architecture
- Data center technology
- **Virtualization technology**
- Multitenant technology
- Web/web services technology

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.20

20

VIRTUALIZATION MANAGEMENT

- Virtual infrastructure management (VIM) tools
- Tools that manage pools of virtual machines, resources, etc.
- Private cloud software systems can be considered as a VIM

- Considerations:
- Performance overhead
  - Paravirtualization: custom OS kernels, I/O passed directly to HW w/ special drivers
- Hardware compatibility for virtualization
- Portability: virtual resources tend to be difficult to migrate cross-clouds

21

VIRTUAL INFRASTRUCTURE MANAGEMENT (VIM)

- Middleware to manage virtual machines and infrastructure of IaaS "clouds"

- Examples
  - OpenNebula
  - Nimbus
  - Eucalyptus
  - OpenStack

22

VIM FEATURES

- Create/destroy VM Instances
- Image repository
  - Create/Destroy/Update images
  - Image persistence
- Contextualization of VMs
  - Networking address assignment
    - DHCP / Static IPs
  - Manage SSH keys

VIM FEATURES - 2

- Virtual network configuration/management
  - Public/Private IP address assignment
  - Virtual firewall management
  - Configure/support isolated VLANs (private clusters)
- Support common virtual machine managers (VMMs)
  - XEN, KVM, VMware
  - Support via libvirt library

VIM FEATURES - 3

- Shared “Elastic” block storage
  - Facility to create/update/delete VM disk volumes
    - Amazon EBS
    - Eucalyptus SC
    - OpenStack Volume Controller

CONTAINER ORCHESTRATION FRAMEWORKS

- Middleware to manage Docker application container deployments across virtual clusters of Docker hosts (VMs)
- Considered Infrastructure-as-a-Service

- **Opensource**
- Kubernetes framework
- Docker swarm
- Apache Mesos/Marathon

- **Proprietary**
- Amazon Elastic Container Service

CONTAINER SERVICES

- **Public cloud container cluster services**
- Azure Kubernetes Service (AKS)
- Amazon Elastic Container Service for Kubernetes (EKS)
- Google Kubernetes Engine (GKE)

- **Container-as-a-Service**
- Azure Container Instances (ACI – April 2018)
- AWS Fargate (November 2017)
- Google Kubernetes Engine Serverless Add-on (alpha-July 2018)

27

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

28

4. MULTITENANT APPLICATIONS

- Each tenant (like in an apartment) has their own view of the application
- Tenants are unaware of their neighbors
- Tenants can only access their data, no access to data and configuration that is not their own

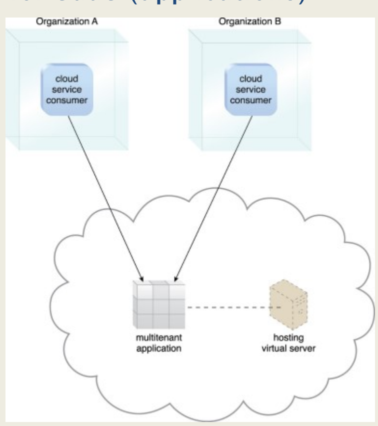
- Customizable features
  - UI, business process, data model, access control
- Application architecture
  - User isolation, data security, recovery/backup by tenant, scalability for a tenant, for tenants, metered usage, data tier isolation

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L14.29

29

MULTITENANT APPS - 2

- Forms the basis for SaaS (applications)



November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L14.30

30

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 20, 2025 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma | L14.31

31

5. WEB SERVICES/WEB

- Web services technology is a key foundation of cloud computing's "<u>**as-a-service**</u>" 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 20, 2025 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma | L14.32

32

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>
```


35

```
// 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>
```


36

```
// 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>
```

November 20, 2025 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025]
School of Engineering and Technology, University of Washington - Tacoma | L14.37

37

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
  }
  ]
}
```

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L14.38

38

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

WE WILL RETURN AT ~4:50 PM

41

OBJECTIVES – 11/20

- Questions from 11/18
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Ch. 5: Cloud Enabling Technology
- **Containerization**
- Container Profiler

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.42

42

CONTAINERIZATION

43

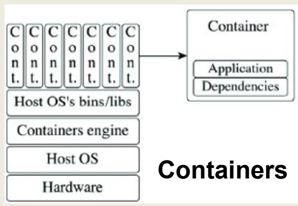
OBJECTIVES – 11/20

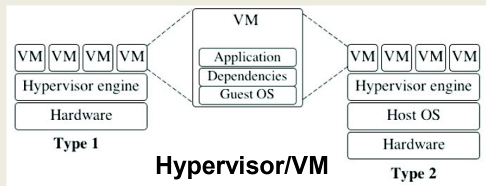
- Questions from 11/18
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Quiz 1
- Tutorial 7
- **Containerization**
- Container Profiler

44

MOTIVATION FOR CONTAINERIZATION

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





45

CONTAINER PERFORMANCE – LU FACTORIZATION PERFORMANCE

- Solve linear equations – matrix algebra

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

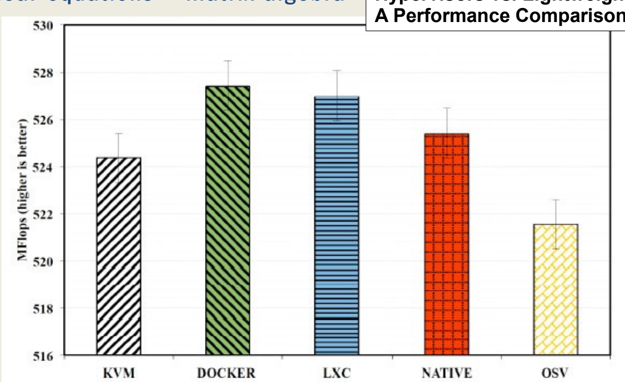
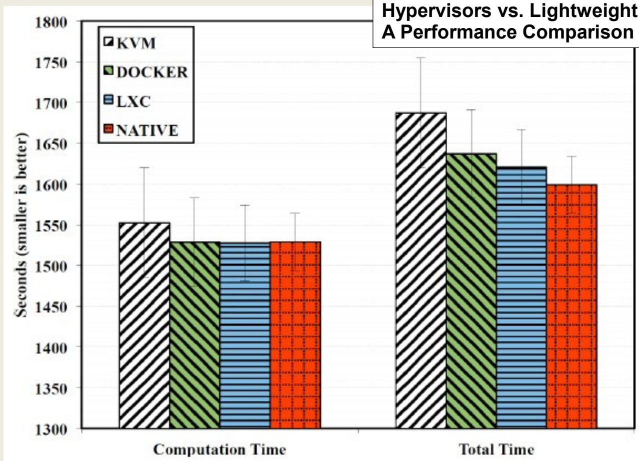


Fig. 4. The value of Linpack results on each platform over 15 runs. This is the particular case of N=1000.

46

CONTAINER PERFORMANCE – Y-CRUNCHER: PI CALCULATOR

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



47

CONTAINER PERFORMANCE – BONNIE++

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

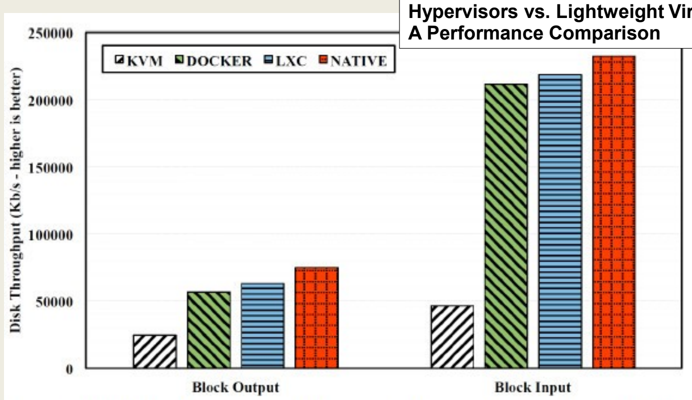


Fig. 6. Disk Throughput achieved by running Bonnie++ (test file of 25 GiB). Results for sequential writes and sequential read are shown.

48

WHAT IS A CONTAINER?

According to NIST (National Institute of Standards Technology)

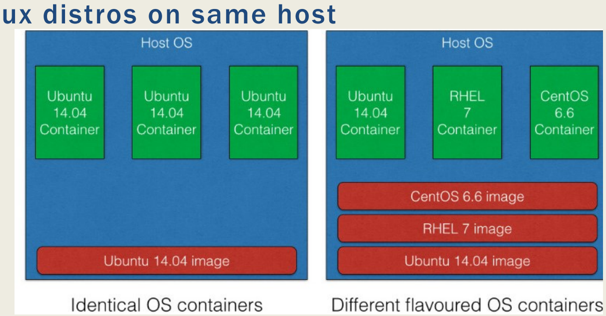
- **Virtualization**: the simulation of the software and/or hardware upon which other software runs. (800-125)
- **System Virtual Machine**: A System Virtual Machine (VM) is a software implementation of a complete system platform that supports the execution of a complete operating system and corresponding 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 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.49

49

OPERATING SYSTEM CONTAINERS

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



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

November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.50

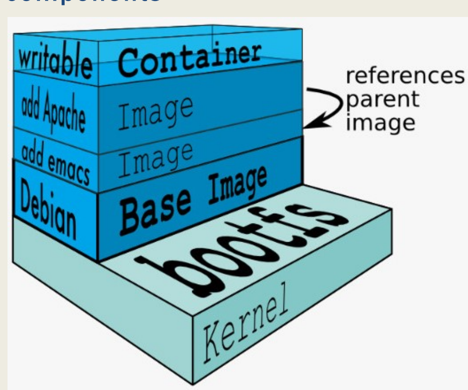
50

APPLICATION CONTAINERS

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

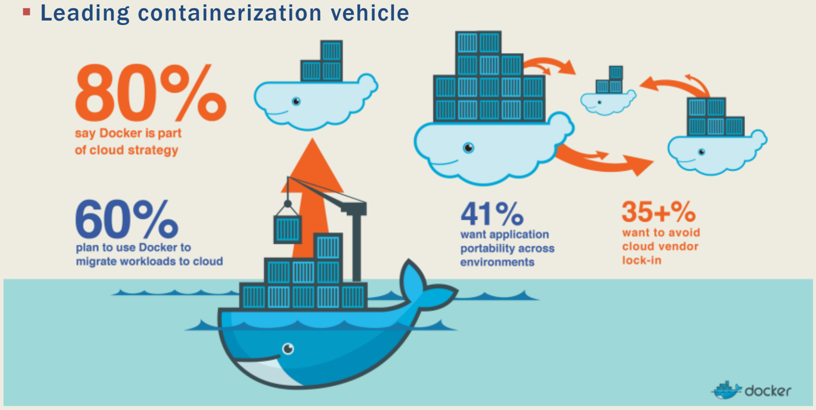
APPLICATION CONTAINERS - 2

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



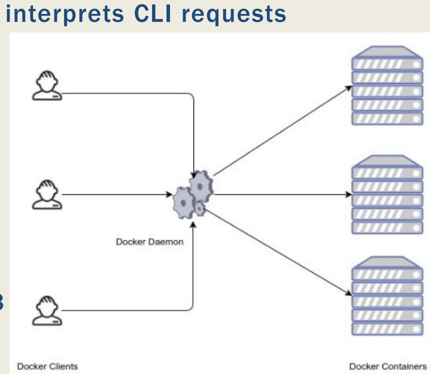
2016 DOCKER SURVEY

- Docker application containers
  - Leading containerization vehicle



DOCKER

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

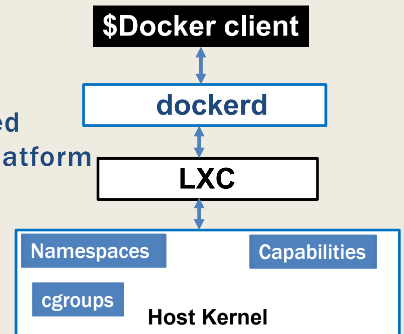


Docker Client-Server Architecture

- 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

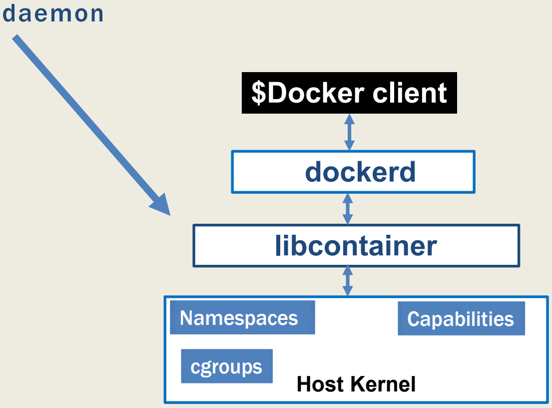


November 20, 2025 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma | L15.55

55

INTRODUCTION OF LIBCONTAINER

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



November 20, 2025 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma | L15.56

56

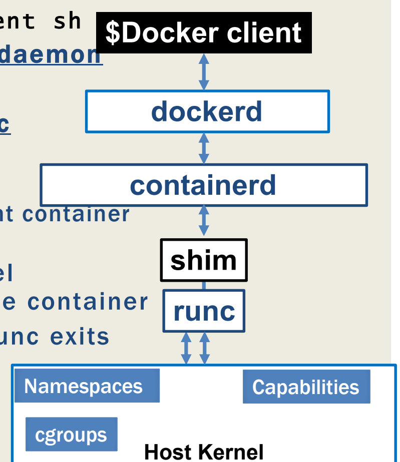
OPEN CONTAINER INITIATIVE (OCI)

- 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**

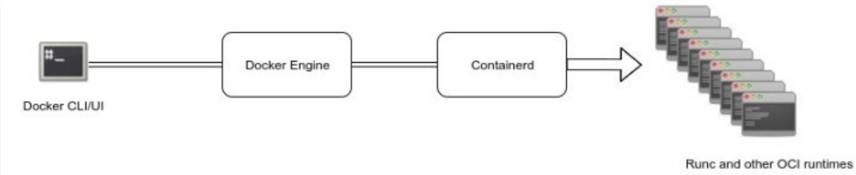
CREATING A CONTAINER

```
$ 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



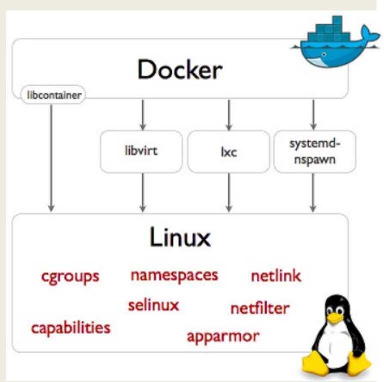
Containerd Integration Architecture

- 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

59

SUPPORT FOR ALTERNATE CONTAINER RUNTIMES

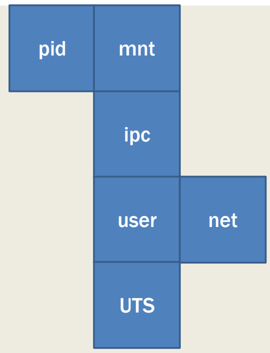
- Modularity of Docker implementation supports "execution drivers concept":
- Enables docker to support many alternate container backends
- OpenVZ, system-nspawn, libvirt-lxc, libvirt-sandbox, qemu/kvm, BSD Jails, Solaris Zones, and chroot



60

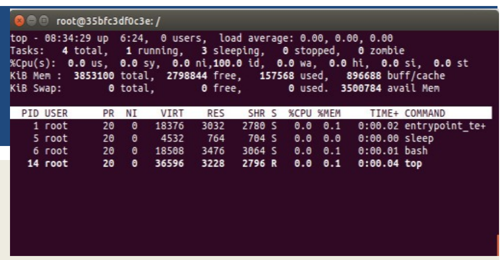
LINUX KERNEL NAMESPACES

- 7 different namespaces in Linux (cgroups not shown)
  - pid, mnt, ipc, user, net, UTS
- Partitions kernel resources



61

NAMESPACES - 2

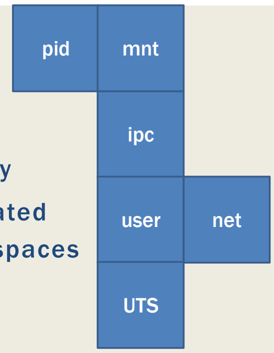


- **Provides isolation of OS entities for containers**
- **mnt**: separate filesystems
- **pid**: independent PIDs; first process in container is PID 1
- **ipc**: prevents processes in different IPC namespaces from being able to establish shared memory. Enables processes in different containers to reuse 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

62

LINUX KERNEL NAMESPACES - 3

- Processes see only their set of resources
- Provides isolation
- Namespaces are hierarchical
- Parent processes can see down the hierarchy
- Each process can only see resources associated with the namespace, and descendent namespaces



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

#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

- **cgroup filesystem:**
- /sys/fs/cgroup
- Can browse resource utilization of containers…

OVERLAY FILE SYSTEMS

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

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

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

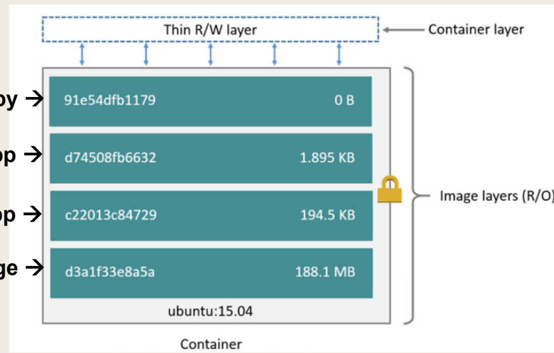
- https://www.slideshare.net/jpetazzo/scale11x-lxc-talk-1/

LAYERED FS: BUILDING A CONTAINER

- **Dockerfile:**

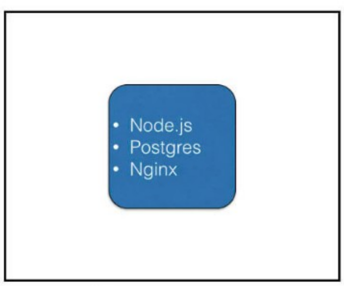
```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Python /app/app.py →
Run make /app →
Copy . /app →
Ubuntu base image →



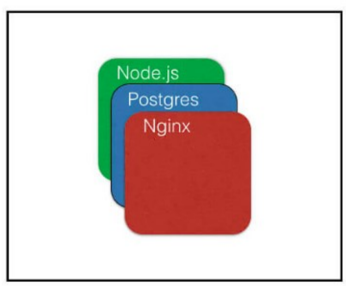
67

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



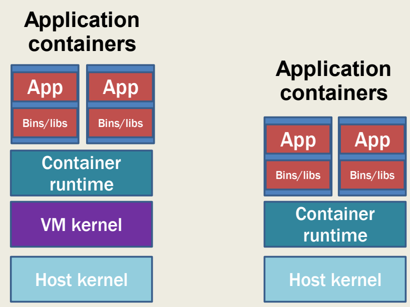
App containers

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

68

CONTAINER ISOLATION

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



69

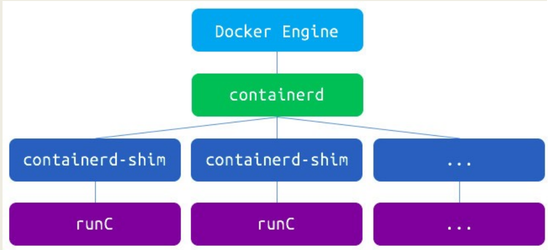
LXC (LINUX CONTAINERS)

- 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

70

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 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.71

71

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 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.72

72

KEY ORCHESTRATION FEATURES

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

CONTAINER ORCHESTRATION FRAMEWORKS - 2

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

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

OBJECTIVES – 11/20

- Questions from 11/18
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Ch. 5: Cloud Enabling Technology
- Containerization
- Container Profiler

75

https://github.com/wlloyduw/ContainerProfiler

CONTAINER PROFILER

OXFORD (GIGA)^n SCIENCE
GigaScience, 2023, 12, 1–11
DOI: 10.1093/gigascience/giad069
Tech Note

Container Profiler: Profiling resource utilization of containerized big data pipelines

Varik Hoang[†], Ling-Hong Hung[†], David Perez, Huazeng Deng, Raymond Schooley, Niharika Arumilli, Ka Yee Yeung and Wes Lloyd[*]

School of Engineering and Technology, University of Washington, Tacoma, WA 98402, USA
[*]Correspondence address. Wes Lloyd, 1900 Commerce St #358426, Tacoma, WA 98402, USA. E-mail: wlloyd@uw.edu
[†]Contributed equally.

Abstract
Background: This article presents the *Container Profiler*, a software tool that measures and records the resource usage of any container-

76

CONTAINER PROFILER

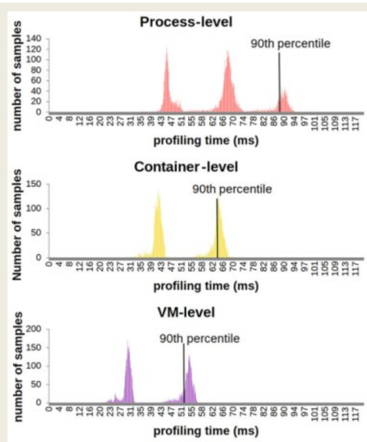
- Captures resource utilization metrics for containers
- Profiles CPU, memory, disk, and network utilization collecting over 60 metrics available from the Linux OS
- Supports two types of profiling
  - **Δ "Delta" Resource Utilization**: Records and calculates total resource utilization from when an initial selection is provided before implementation is verified.
  - **Time series sampling**: supports a configurable sampling interval for continuous monitoring of resources consumed by containers
- Similar profiling techniques compared to SAAF
- Uses Linux proc filesystem "man procfs"
- Implemented with a combination of custom code and the Python-based **psutil** library to obtain resource utilization data rapidly

November 23, 2016	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L19.77

77

CONTAINER PROFILER: PROFILING OVERHEAD

- Profiling overhead *(9,000 samples)*:
- Use case: RNA-sequencing data processing pipeline *(containerized)*
- Hardware: IBM Cloud dual bx2d metal 96 vCPUs processors, 384 GB RAM
  - **Process-level**: 3 peaks indicate different behavior presumably based on the number of processes running inside the containerd cpuIdle time.
  - Process level collects and reports all available metrics
- Other Supported Profiling Modes:
  - Container-level profiling
    - Does not collect process-level metrics
    - Faster
  - VM-level profiling:
    - Even faster
    - Only collects host-level metrics

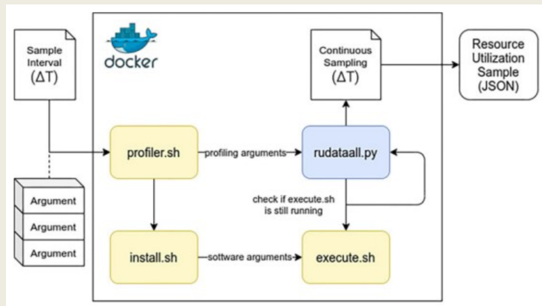


November 20, 2025	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2025] School of Engineering and Technology, University of Washington - Tacoma	L15.78

78

CONTAINER PROFILER: PROFILING OVERHEAD EXAMPLE

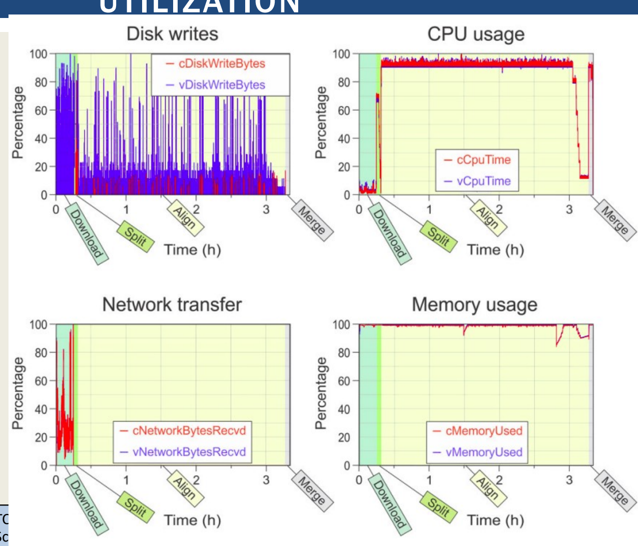
- 99.95% of process level samples were collected under 100ms
- All container-level samples collected under 74ms
- All host (VM-level) samples collected under 60ms
- UMI RNA-sequencing pipeline use case required 2.5 hours to execute with 1-second sampling at full verbosity (all CPU, network I/O, disk I/O, and memory metrics collected)



79

CONTAINER PROFILER: TIME SERIES ANALYSIS OF RESOURCE UTILIZATION

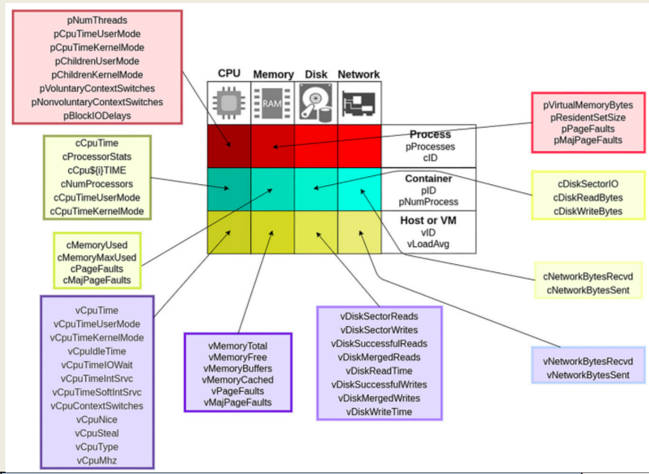
- CPU utilization
- Memory utilization
- Disk writes
- Network transfer



80

CONTAINER PROFILER: METRICS

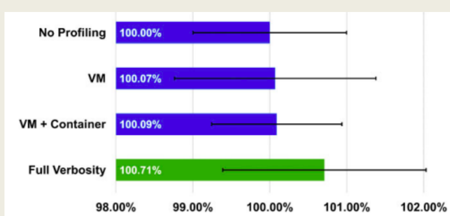
- Measures up to ~60 metrics
- Configurable Verbosity
- Can rapidly
- Change vebosity



81

CONTAINER PROFILER: USE CASE

- Profiling overhead for jobs profiled by the ContainerProfiler:

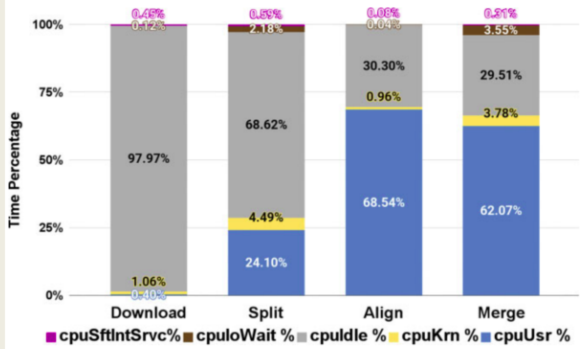


- Use case: Uniform Molecular Identifier (UMI) RNA-seq pipeline
- Four state: download, split, align, merge

82

CONTAINER PROFILER: DELTA PROFILING

- Delta profiling supports viewing total resource utilization (CPU, disk, network) for a containerized task or mutli-stage pipeline
- Task = 1 container
- Pipeline = many containers
- Delta captures the perceived system resources used by the task



- Here is the delta profiling graph for resource utilization:
- Four tasks: Download, Split, Align, Merge
- Graph shows % time in different CPU modes (cpuUsr, cpuKrn, cpuIdle, etc.)

83

QUESTIONS

84