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

Containerization II & Kubernetes

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

TR 5:50-7:50 PM

1

OFFICE HOURS – COMING UP

- **Friday 11/18**
  - 11:30 to 1:30 pm - Zoom
- **Monday 11/21 with Zening Zhao**
  - 12:30 to 1:30 pm - Zoom
- **Tuesday 11/22**
  - 3:30 to 5:30 pm - CP 229 and Zoom
- Or email for appointment

**- Extra Office Hours ADDED: moving to 5/hrs/wk for remainder of quarter*
> Office Hours set based on Student Demographics survey feedback

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.2

2

OBJECTIVES – 11/17

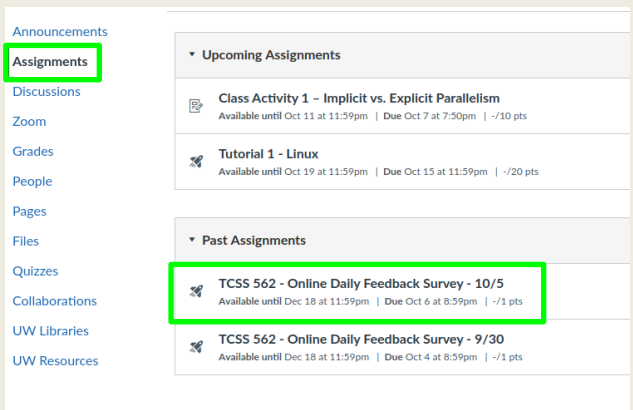
- Questions from 11/15
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Quiz 1
- Tutorial 7
- Tutorial 8: Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction (UW Research Study)
- Containerization
- Kubernetes

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] 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 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] 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):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.65 (↓ - *previous 6.77*)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.60 (↑ - previous 5.40)**

- **Response rates:**
- TCSS 462: 20/33 – 60.61%
- TCSS 562: 21/26 – 80.77%

6

FEEDBACK FROM 11/15

- ..

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 17, 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 – 11/17

- Questions from 11/15
- **Tutorials Questions**
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Quiz 1
- Tutorial 7
- Tutorial 8: Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction (UW Research Study)
- Containerization
- Kubernetes

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.12

12

TUTORIAL 4 UPDATE

- Tutorial 4 (originally due Nov 6, extended to Nov 23)
- 30 graded - Average is 100 % !
- 9 x TCSS 462 students: **no submission yet** (0 attempts)
- 16 x have been asked to resubmit but have not yet done so
  - Some submissions have required items missing in the original submission which prevent a grade from being assigned. The grader has detailed the missing items.
- 4 students have resubmitted Tutorial 4 and a grade is pending

- Please drop into office hours, or contact instructor via Canvas/email to resolve issues and complete Tutorial 4 !

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.13

13

TUTORIAL 5 & 6 UPDATE

- Tutorial 5 (originally due Nov 13, now extended to Nov 29)
- 10 x TCSS 462 students: no submission yet (0 attempts)
- 49 submissions to date, grading is pending

- Tutorial 6 (originally due Nov 20, now extended to Dec 2)
- 6 submissions to date

- The final term project is due **Friday December 16 at 11:59pm**
- It is important to **complete** Tutorials 4, 5, and 6 ASAP to give time to segue to working on the term project
- Please note while assignment extensions are possible, a rushed term project can be obvious and fail to deliver many case study insights

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.14

14

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

15

TUTORIAL 4 – ~~NOV 6~~ NOV 23

- 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

16

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

17

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.

18

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***************************
```

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.19

19

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**

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.20

20

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 ?**

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.21

21

TUTORIAL 5 – ~~NOV 13~~ NOV 29

- 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

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.22

22

TUTORIAL 6 – ~~NOV 21~~ DEC 2

- 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

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.23

23

OBJECTIVES – 11/17

- Questions from 11/15
- Tutorials Questions
- **Class Presentations Schedule - Cloud Technology or Research Paper Review**
- Quiz 1
- Tutorial 7
- Tutorial 8: Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction (UW Research Study)
- Containerization
- Kubernetes

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.24

24

GROUP PRESENTATIONS

- TWO OPTIONS:
- *Cloud technology presentation*
- *Cloud research paper presentation*
  - Recent & suggested papers will be posted at: http://faculty.washington.edu/wlloyd/courses/tcss562/papers/
- Presentation dates:
  - Tuesday November 22, Tuesday November 29
  - Tuesday December 6, Thursday December 8
- Peer Reviews
  - Word DOCX form will be provided, fill out, submit PDF on Canvas
  - Feedback shared with groups
  - TCSS 462: 1 review/day required, additional are extra credit
  - TCSS 562: same as 462, but no peer review req'd on day of your talk

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.25

25

GROUP PRESENTATIONS

- 11 Presentation Teams
- 3 Cloud Technology Talks
- 8 Cloud Research Paper Presentations
- Thank you for the submissions
- Two students – we are not sure of your team and need clarification:
- Alan Liu (team 15 ???)
- Andrew Moreno-Escareno (team 8 ???)

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.26

26

PRESENTATION SCHEDULE

- **Tuesday November 22**

1. Bhagyashree Aras, Dhruvi Kaswala (team 3)
Research paper: Towards a Serverless Bioinformatics Cyberinfrastructure Pipeline
2. * Yafei Li, Sue Yang (team 5)
Research paper: Cypress: Input size –Sensitive Container Provisioning and Request Scheduling for Serverless
(can exchange with team 8 or team 12 for Dec 8 if both teams agree)*

- **Tuesday November 29**

1. Divya Jacob, Nehaa Vuppala, Nandhini Dhanasekaran (team 10)
Research paper: Efficient GPU Sharing for Serverless Workflows
2. Jasleen Kaur, Naman Bhaia (team 4)
Research paper: A Serverless Publish/Subscribe System
3. Jeffrey Stockman, Rick Morrow, Mahmoud Ali Elkamhawy (team 1)
Research paper: Migrating from Microservices to Serverless: An IoT Platform Case Study

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.27

27

PRESENTATION SCHEDULE - 2

- **Tuesday December 6**

1. Yuan Huang, Yifan Xie (is Alan Liu in this group?) (team 15)
Research paper: A Prediction based Autoscaling in Serverless Computing
2. Angela Mu, Xiaojie Li, Ruigeng Zhang (team 6)
Research paper: Apollo: Modular and Distributed Runtime System for Serverless Function Compositions on Cloud, Edge, and IoT Resources
3. Jui Wang, Jinming Yu (team 7)
Cloud Technology: AWS Rekognition

- **Thursday December 8**

1. Mohammed Alshayeb (team 2)
Research paper (2021 list) Towards Federated Learning using FaaS Fabric
2. Nicole Guobadia (team 8)
Cloud Technology: AzureML (Machine Learning as a Service)
3. RamaSoumya Naraparaju, Sathwika Suddala, Chhavi Gupta (team 12)
Cloud Technology: Amazon Redshift

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.28

28

OBJECTIVES – 11/17

- Questions from 11/15
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Quiz 1
- Tutorial 7
- Tutorial 8: Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction (UW Research Study)
- Containerization
- Kubernetes

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.29

29

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

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.30

30

OBJECTIVES – 11/17

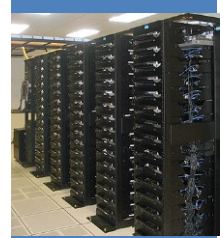
- Questions from 11/15
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Quiz 1
- **Tutorial 7**
- Tutorial 8: Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction (UW Research Study)
- Containerization

November 17, 2022 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma | L15.31

31

TUTORIAL #7

DOCKER, CGROUPS, RESOURCE ISOLATION



November 17, 2022

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

L15.32

32

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

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.33

33

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

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.34

34

```
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

35

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 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.36

36

OBJECTIVES – 11/17

- Questions from 11/15
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Quiz 1
- Tutorial 7
- **Tutorial 8: Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction (UW Research Study)**
- Containerization
- Kubernetes

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.37

37

WE WILL RETURN AT ~7:00 PM

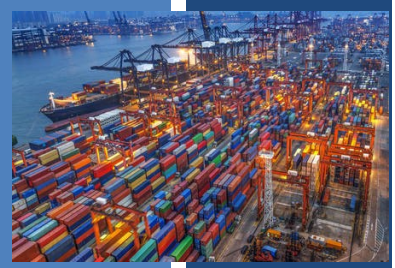
38

OBJECTIVES – 11/17

- Questions from 11/15
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Quiz 1
- Tutorial 7
- Tutorial 8: Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction (UW Research Study)
- Containerization
- Kubernetes

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.39

39



CONTAINERIZATION

November 17, 2022

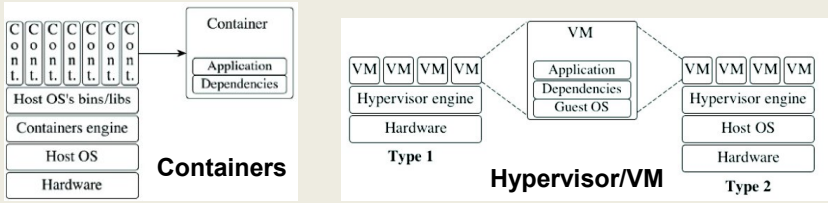
TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022]
School of Engineering and Technology, University of Washington - Tacoma

L15.40

40

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



41

CONTAINER PERFORMANCE – LU FACTORIZATION PERFORMANCE

- Solve linear equations – matrix algebra

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

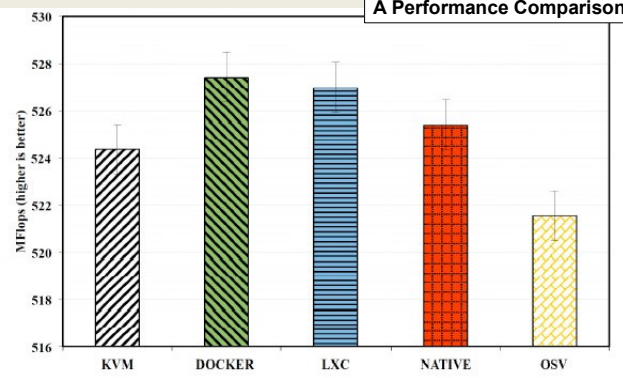
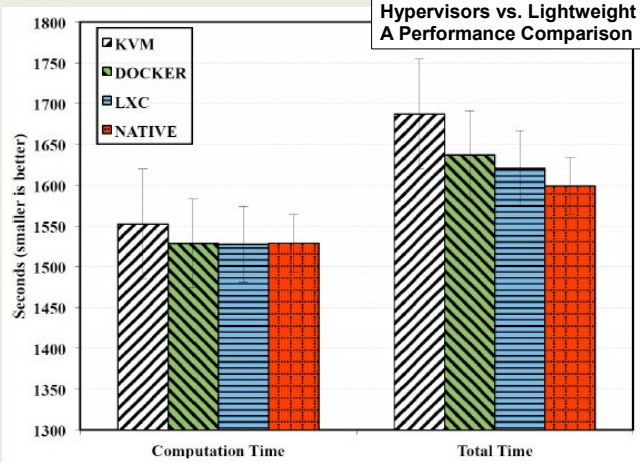


Fig. 4. The value of Linpack results on each platform over 15 runs. This is the particular case of N=1000.

42

CONTAINER PERFORMANCE – Y-CRUNCHER: PI CALCULATOR

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



CONTAINER PERFORMANCE – BONNIE++

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

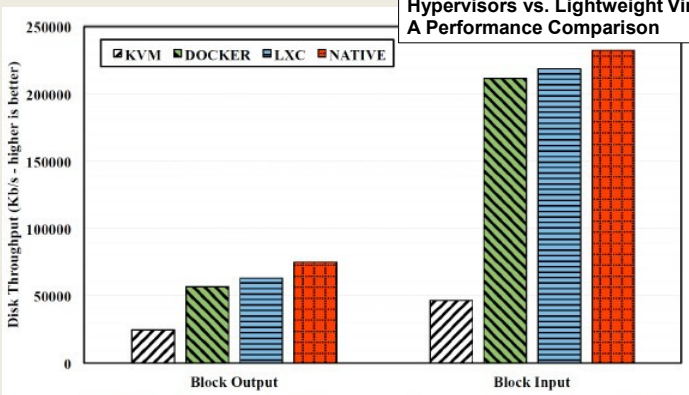


Fig. 6. Disk Throughput achieved by running Bonnie++ (test file of 25 GiB). Results for sequential writes and sequential read are shown.

WHAT IS A CONTAINER?

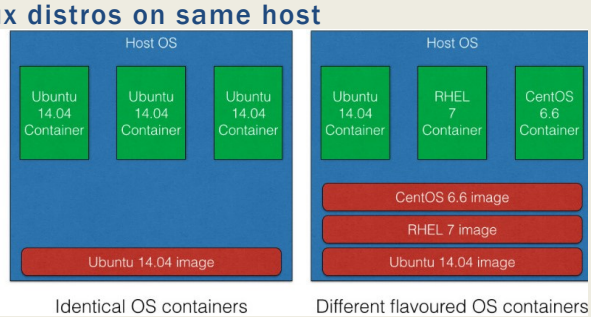
According to NIST (National Institute of Standards Technology)

- **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

45

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/

46

APPLICATION CONTAINERS

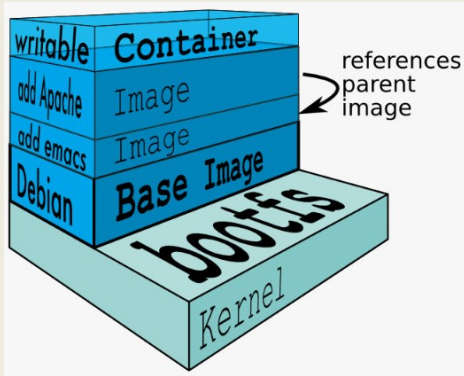
- Designed to package and run a single service
- All containers share host kernel
- Subtle differences from operating system containers
- Examples: Docker, Rocket
- Docker: runs a single process on creation
- OS containers: run many OS services, for an entire OS
- Create application containers for each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

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.47

47

APPLICATION CONTAINERS - 2

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

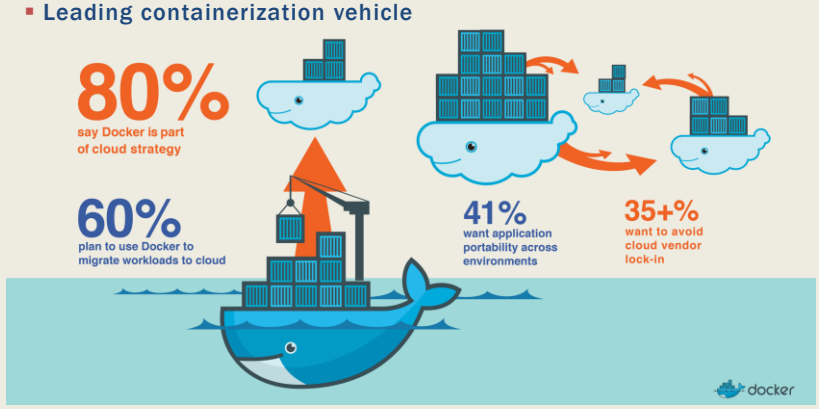


November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.48

48

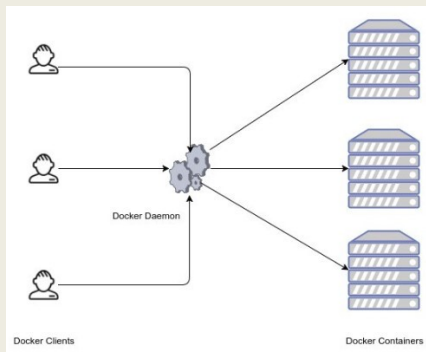
2016 DOCKER SURVEY

- 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

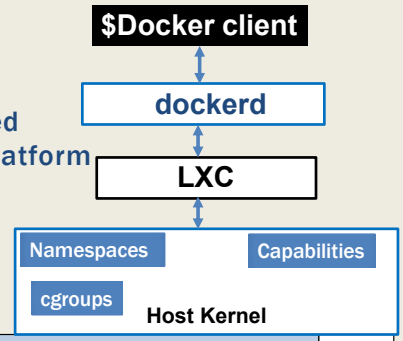


Docker Client-Server Architecture

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

ORIGINAL DOCKER ENGINE IMPLEMENTATION

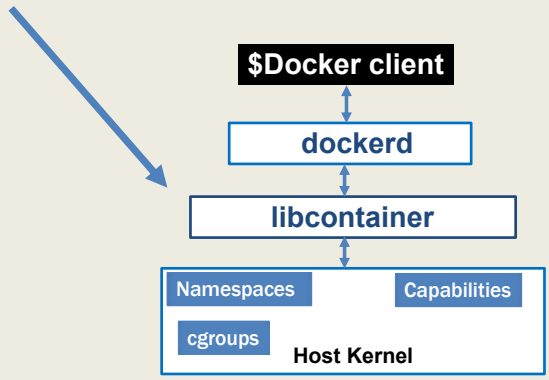
- (1) Original Docker engine relied on LXC
  - LXC itself is a containerization tool predating Docker
  - Original Docker API just called it
  - LXC originally provided access to Linux kernel features: namespaces and cgroups
  - LXC was Linux specific – caused issues if wanting to be multi-platform
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



51

INTRODUCTION OF LIBCONTAINER

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



52

OPEN CONTAINER INITIATIVE (OCI)

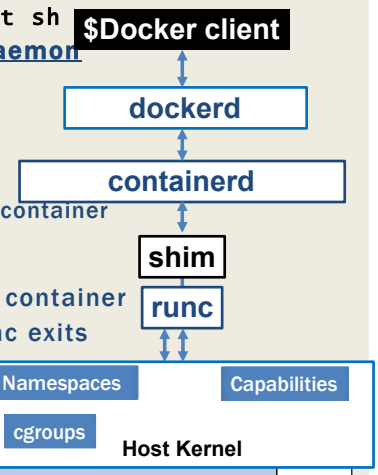
- OCI created container standards for:
  - Image specification
  - Container runtime specification
- Docker 1.1 (2016): Docker refactored the docker engine to be compliant with OCI standards
  - Essentially this introduced abstraction layers (i.e. generic interfaces that map to the implementation) so 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**

53

CREATING A CONTAINER

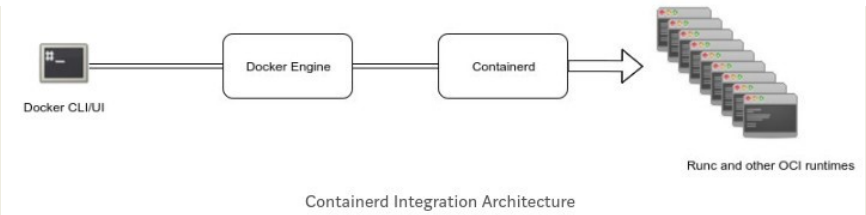
```
$ docker run -it --rm tcss558client sh
```

- 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 !!!



54

CREATING A CONTAINER - 2

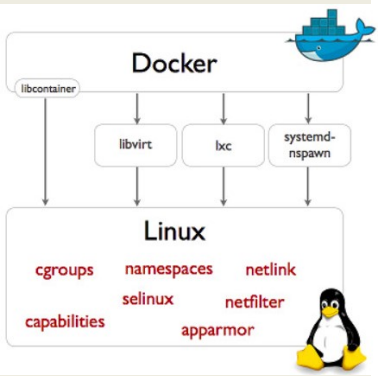


- Docker CLI: interfaces with **dockerd** daemon
- Docker engine: **dockerd** daemon, interfaces with **containerd**
- **Containerd**: simple daemon, interfaces with **runc** to manage containers; CRUD interface for containers, images, volumes, networks, builds; HTTP API → Google RPC (gRPC) interface;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

55

SUPPORT FOR ALTERNATE CONTAINER RUNTIMES

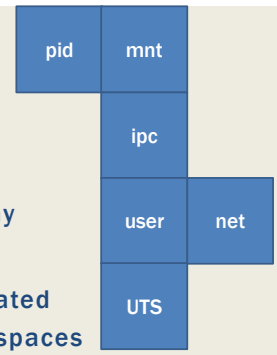
- Modularity of Docker implementation supports "execution drivers concept":
- Enables docker to support many alternate container backends
- OpenVZ, system-nspawn, libvirt-lxc, libvirt-sandbox, qemu/kvm, BSD Jails, Solaris Zones, and chroot



56

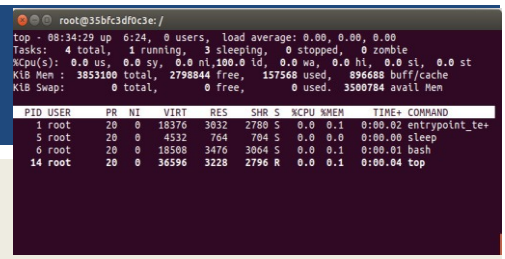
LINUX KERNEL NAMESPACES

- 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



57

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

58

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

59

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

60

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

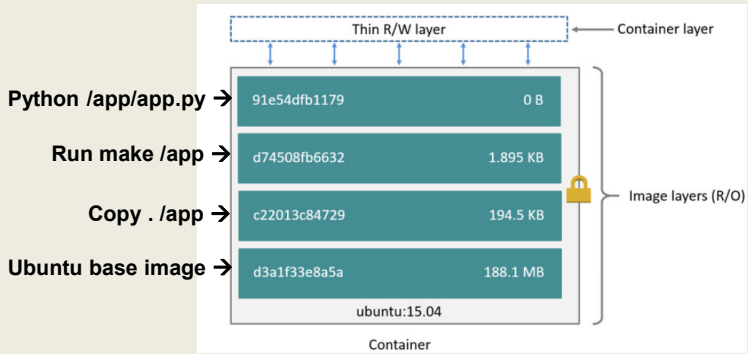
- https://www.slideshare.net/jpetazzo/scale11x-lxc-talk-1/

61

LAYERED FS: BUILDING A CONTAINER

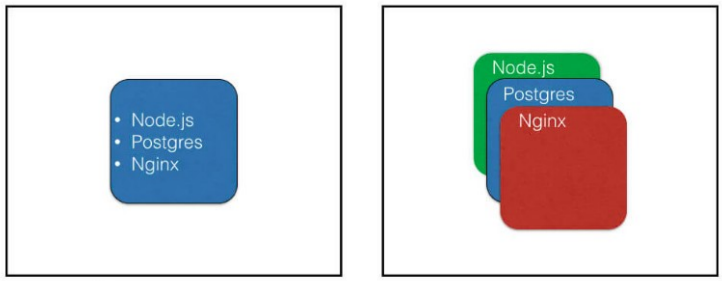
- Dockerfile:

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



62

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

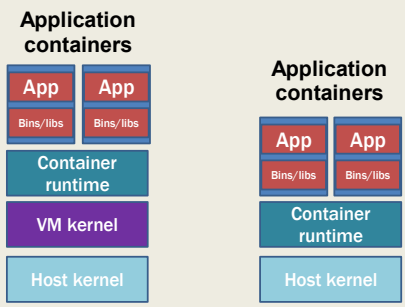
App containers

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

63

CONTAINER ISOLATION

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



64

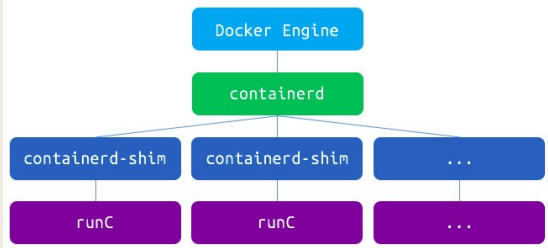
LXC (LINUX CONTAINERS)

- 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

65

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

66

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

67

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

68

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 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.69

69

OBJECTIVES – 11/17

- Questions from 11/15
- Tutorials Questions
- Class Presentations Schedule -
  Cloud Technology or Research Paper Review
- Quiz 1
- Tutorial 7
- Tutorial 8: Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction (UW Research Study)
- Containerization
- Kubernetes

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.70

70

KUBERNETES

from: "The Kubernetes Book", Nigel Poulton and Pushkar Joglekar, Version 7.0, September 2020

L15.71

71

KUBERNETES

- Name is from the Greek word meaning Helmsman
  - The person who steers a seafaring ship
  - The logo reinforces this theme
- Kubernetes is also sometimes called K8s
- Kubernetes is an application orchestrator
- Most common use case is to containerize cloud-native microservices applications
- What is an orchestrator?
  - System that deploys and manages applications

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.72

72

KUBERNETES – 2

Why does Google want to give Kubernetes away for free?

- Initially developed by Google
- **Goal:** *make it easier for potential customers to use Google Cloud*
- Kubernetes leverages knowledge gained from two internal container management systems developed at Google
  - Borg and Omega
- Google donated Kubernetes to the Cloud Native Computing Foundation in 2014 as an open-source project
- Kubernetes is written in Go (Golang)
- Kubernetes is available under the Apache 2.0 license
- Releases were previously maintained for only 8 months!
- Starting w/ v 1.19 (released Aug 2020) support is 1 year

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.73

73

GOALS OF KUBERNETES

1. Deploy your application
2. Scale it up and down dynamically according to demand
3. Self-heal it when things break
4. Perform zero-downtime rolling updates and rollbacks

- These features represent automatic infrastructure management
- Containerized applications run in container(s)
- Compared to VMs, containers are thought of as being:
  - Faster
  - More light-weight
  - More suited to rapidly evolving software requirements

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.74

74

CLOUD NATIVE APPLICATIONS

- Applications designed to meet modern software requirements including:
  - **Auto-scaling:** resources to meet demand
  - **Self-healing**: *required for high availability (HA) and fault tolerance*
  - **Rolling software updates**: with no application downtime for DevOPS
  - **Portability**: can run anywhere there's a Kubernetes cluster

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.75

75

WHAT IS A MICROSERVICES APP?

- Application consisting of many specialized parts that communicate and form a meaningful application
- Example components of a microservice eCommerce app:

Web front-end	Catalog service
Shopping cart	Authentication service
Logging service	Persistent data store

- **KEY IDEAS:**
- Each microservice can be coded/maintained by different team
- Each has its own release cadence
- Each is deployed/scaled separately
- Can patch & scale the log service w/o impacting others

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.76

76

KUBERNETES - 3

- Provides "an operating system for the cloud"
- Offers the de-facto standard platform for deploying and managing cloud-native applications
- OS: abstracts physical server, schedules processes
- Kubernetes: ***abstracts the cloud***, schedules microservices
- Kubernetes abstracts differences between private and public clouds
- Enable cloud-native applications to be cloud agnostic
  - i.e. they don't care *WHAT* cloud they run on
  - Enables fluid application migration between clouds
- Kubernetes provides rich set of tools/APIs to introspect (observe and examine) your apps

77

KUBERNETES - 4

- Features:
- A "control plane" – brain of the cluster
  - Implements autoscaling, rolling updates w/o downtime, self-healing
- A "bunch of nodes" – workers (muscle) of the cluster
- Provides orchestration
- The process of organizing everything into a useful application
- And also keeping it running smoothly

78

KUBERNETES - CLUSTER MANAGEMENT

- Master node(s) manage the cluster by:
  - Making scheduling decisions
  - Performing monitoring
  - Implementing changes
  - Responding to events
- *Masters implement the control plane of a Kubernetes cluster*
- Recipe for deploying to Kubernetes:
- Write app as independent microservices in preferred language
- Package each microservice in a container
- Create a manifest to encapsulate the definition of a Pod
- Deploy Pods to the cluster w/ a higher-level controller such as "Deployments" or "DaemonSets"

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.79

79

DECLARATIVE SERVICE APPROACH

- **Imperative definition**: sets of commands and operations
  - Example: BASH script, Dockerfile
- **Declarative definition**: specification of a service's properties
  - What level of service it should sustain, etc.
  - Example: Kubernetes YAML files
- Kubernetes manages resources **declaratively**
- How apps are deployed and run are defined with YAML files
- YAML files are POSTed to Kubernetes endpoints
- Kubernetes deploys and manages applications based on declarative service requirements
- If something isn't as it should be: *Kubernetes automatically tries to fix it*

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.80

80

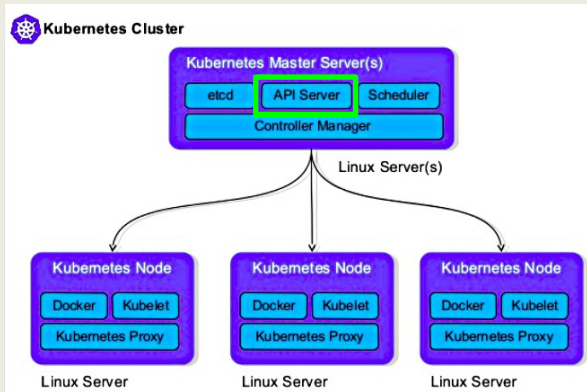
KUBERNETES MASTERS

- Provide system services to host the control plane
- Simplest clusters use only 1 master – no replication
  - Suitable for lab and dev/test environments
- Production environments: masters are replicated ~3-5x
  - Provides fault tolerance and high availability (HA)
  - Cloud-based managed Kubernetes services offer HA deployments

81

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



82

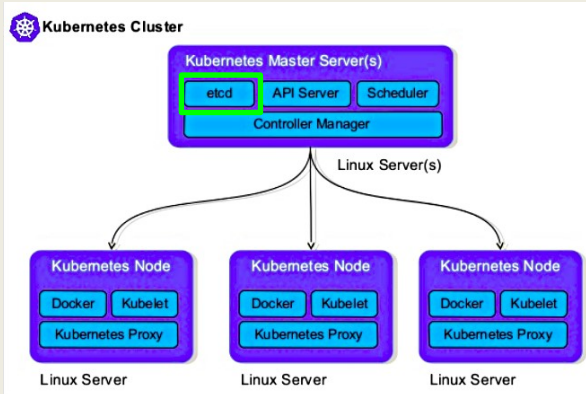
API SERVER

- Can run on 1-node for lab, test/dev environments
- Default port is 443
- Exposes a RESTful API where YAML configuration files are POST(ed) to
- YAML files (manifests) describe desired state of an application
  - Which container image(s) to use
  - Which ports to expose
  - How many POD replicas to run

83

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



84

CLUSTER STORE

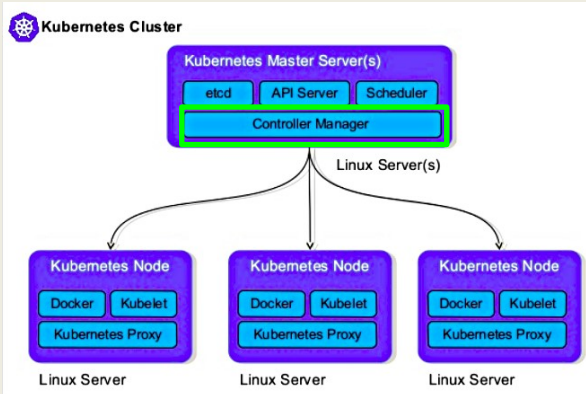
- Used to persist Kubernetes cluster state
- Persistently stores entire configuration and state of the cluster
- Currently implemented with **etcd**
  - Popular distributed key/value store (db) supporting replication
  - HA deployments may use ~3-5 replicas
  - Is the authority on true state of the cluster
- etcd prefers consistency over availability
- etcd failure: apps continue to run, nothing can be reconfigured
- Consistency of writes is vital
- Employs RAFT consensus protocol to negotiate which replica has correct view of the system in the event of replica failure

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.85

85

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.86

86

CONTROLLER MANAGER

- Provides a "controller" of the controllers
  - Implements background control loops to monitor cluster and respond to events
  - Control loops include: node controller, endpoints controller, replicaset controller, etc…
- **GOAL**: ***ensure cluster current state matches desired state***
- Control Loop Logic:

1. Obtain desired state (defined in manifest YAMLs)
2. Observe the current state
3. Determine differences
4. Reconcile differences

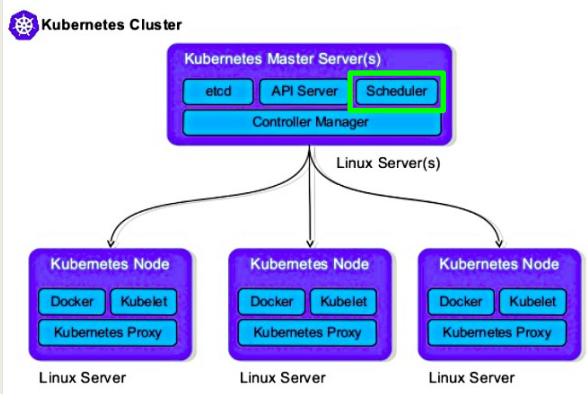
- Controllers are specialized to manage a specific resource type
  - They are not aware/concerned with of other parts of the system

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.87

87

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.88

88

TASK SCHEDULER

- Scheduler's job is to identify the best node to run a task
  - Scheduler does not actually run tasks itself
- Assigns work tasks to appropriate healthy nodes
- Implements complex logic to filter out nodes incapable of running specified task(s)
- Capable nodes are ranked
- Node with highest ranking is selected to run the task

89

ENFORCING SCHEDULING PREDICATES

- Scheduler performs predicate (property) checks to verify how/where to run tasks
  - Is a node tainted?
  - Does task have affinity (deploy together), anti-affinity (separation) requirements?
  - Is a required network port available on the node?
  - Does node have sufficient free resources?
- Nodes incapable of running the task are eliminated as candidate hosts

90

RANKING NODES

- Remaining nodes are ranked based on for example:
1. Does the node have the required images?
   - Cached images will lead to faster deployment time
2. How much free capacity (CPU, memory) does the node have?
3. How many tasks is the node already running?

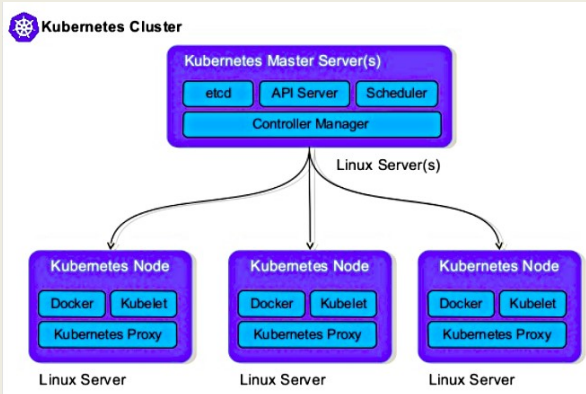
- Each criterion is worth points
- ***Node with most points is selected***
- If there is no suitable node, task is not scheduled, but marked as pending
- **PROBLEM**: *There is no one-sized fits all solution to selecting the best node. How weights are assigned to conditions may not reflect what is best for the task*

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.91

91

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.92

92

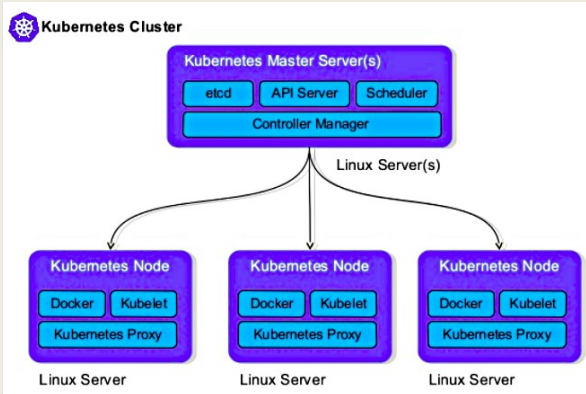
CLOUD CONTROLLER MANAGER

- Abstracts and manages integration with specific cloud(s)
- Manages vendor specific cloud infrastructure to provide instances (VMs), load balancing, storage, etc.
- Support for AWS, Azure, GCP, Digital Ocean, IBM, etc.

93

MASTER SERVICES

- API Server
- Cluster store
- Controller Manager
- Scheduler
- Cloud controller



94

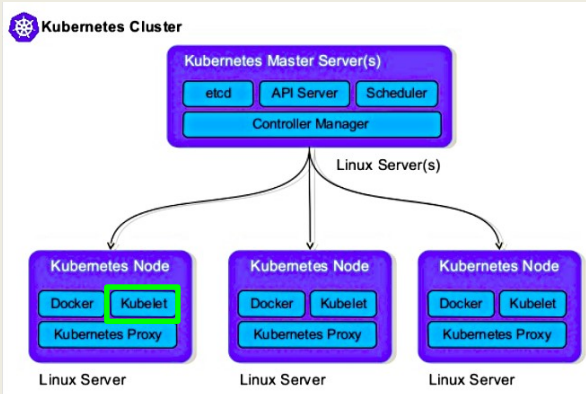
WORKER NODES

- Nodes perform tasks (i.e. host containers & services)
- Three primary functions:
  1. Wait for the scheduler to assign work
  2. Execute work (host containers, etc.)
  3. Report back state information, etc.
- Nodes are considerably simpler than masters

95

WORKER NODES

- Kubelet
- Container runtime (*Docker, etc.*)
- Kubernetes Proxy



96

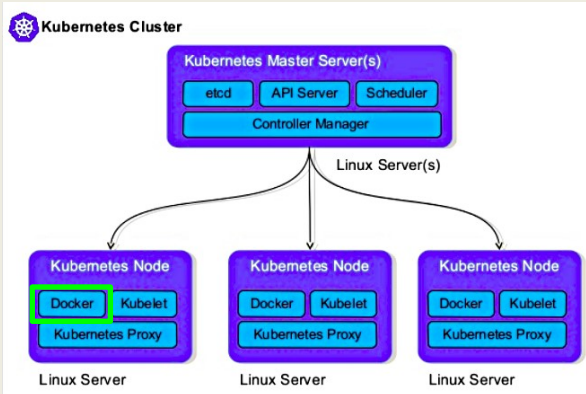
KUBELET

- Main Kubernetes agent
- Runs on every node
- Adding a new node installs the kubelet onto the node
- Kubelet registers the node with the cluster
- Monitors API server for new work assignments
- Maintains reporting back to control plane
- When a node can't run a task, kubelet is NOT responsible for finding an alternate node

97

WORKER NODES

- Kubelet
- Container runtime (*Docker, etc.*)
- Kubernetes Proxy



98

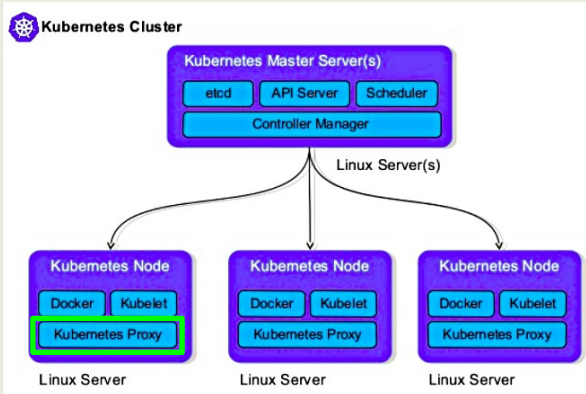
CONTAINER RUNTIME(S)

- Each node requires a container runtime to run containers
- Early versions had custom support for a limited number of container types, e.g. Docker
- Kubernetes now provides a standard Container Runtime Interface (CRI)
- CRI exposes a clean interface for 3rd party container runtimes to plug-in to
- Popular container runtimes: Docker, containerd, Kata

99

WORKER NODES

- Kubelet
- Container runtime (*Docker, etc.*)
- Kubernetes Proxy



100

KUBE-PROXY

- Runs on every node in the cluster
- Responsible for managing the cluster's networking
- Ensures each node obtains a unique IP address
- Implemented local IPTABLES and IPVS rules to route and load-balance traffic
- IPTABLES (ipv4) – enables configuration of IP packet filtering rules of the Linux kernel firewall
- IPVS – IP Virtual Server: provides transport-layer (layer 4) load balancing as part of the Linux kernel; Configured using ipvsadm tool in Linux

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.101

101

CORE KUBERNETES COMPONENTS

- Kubernetes DNS
- Pods
- Services

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.102

102

KUBERNETES DNS

- Every Kubernetes cluster has an internal DNS service
- Accessed with a static IP
- Hard-coded so that every container can find it
- Every service is registered with the DNS so that all components can find every Service on the cluster by **NAME**
- Is based on CoreDNS (https://coredns.io)

103

CORE KUBERNETES COMPONENTS

- Kubernetes DNS
- Pods
- Services

104

PODS

- Pod – atomic unit of deployment & scheduling in Kubernetes
- A Kubernetes Pod is defined to run a containerized application
- Kubernetes manages Pods, not individual containers
- Cannot run a container directly on Kubernetes
- All containers run through Pods

- Pod comes from "pod of whales"
- Docker logo shows a whale with containers stacked on top
- Whale represents the Docker engine that runs on a single host
- Pods encapsulate the definition of a single microservice for hosting purposes
- Pods can have a single container, or multiple containers if the service requires more than one

105

PODS - 2

- Examples of multi-container Pods:
  - Service meshes
  - Web containers with a helper container that pulls latest content
  - Containers with a tightly coupled log scraper or profiler
- YAML manifest files are used to provide a declarative description for how to run and manage a Pod

- To run a pod, POST a YAML to the API Server: "kubectl run <NAME>" where NAME is the service
- A Pod runs on a single node (host)
- Pods share:
  - Interprocess communication (IPC) namespace
  - Memory, Volumes, Network stack

106

PODS - 3

- Pods provide a "fenced" environment to run containers
- Provide a "sandbox"
- Only tightly coupled containers are deployed with a single pod
- Best practice: decouple individual containers to separate pods
  - *What is the best container composition into pods?* (1:1, 1:many)
- **Scaling**
  - Pods are the unit of scaling
  - Add and remove pods to scale up/down
  - Do not add containers to a pod, add pod instances
  - Pod instances can be scheduled on the same or different host
- **Atomic Operation**
  - Pods are either fully up and running their service (i.e. port open/exposed), or pods are down / offline

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.107

107

PODS - 4

- **Pod Lifecycle**
  - An application should not be tightly bound or dependent on a specific Pod instance
  - Pods are designed to fail and be replaced
  - Use of **service objects** in Kubernetes help decouple pods to offer resiliency upon failure
- **Deployments**
  - Higher level controllers often used to deploy pods
  - Controllers implement a controller and watch loop:
  - "Deployments" – offer scalability & rolling updates
  - "DaemonSets" – run instance of service on every cluster node
  - "StatefulSets" – used for stateful components
  - "CronJobs" – for short lived tasks that need to run at specified times

November 17, 2022	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2022] School of Engineering and Technology, University of Washington - Tacoma	L15.108

108

CORE KUBERNETES COMPONENTS

- Kubernetes DNS
- Pods
- Services

109

KUBERNETES "SERVICES"

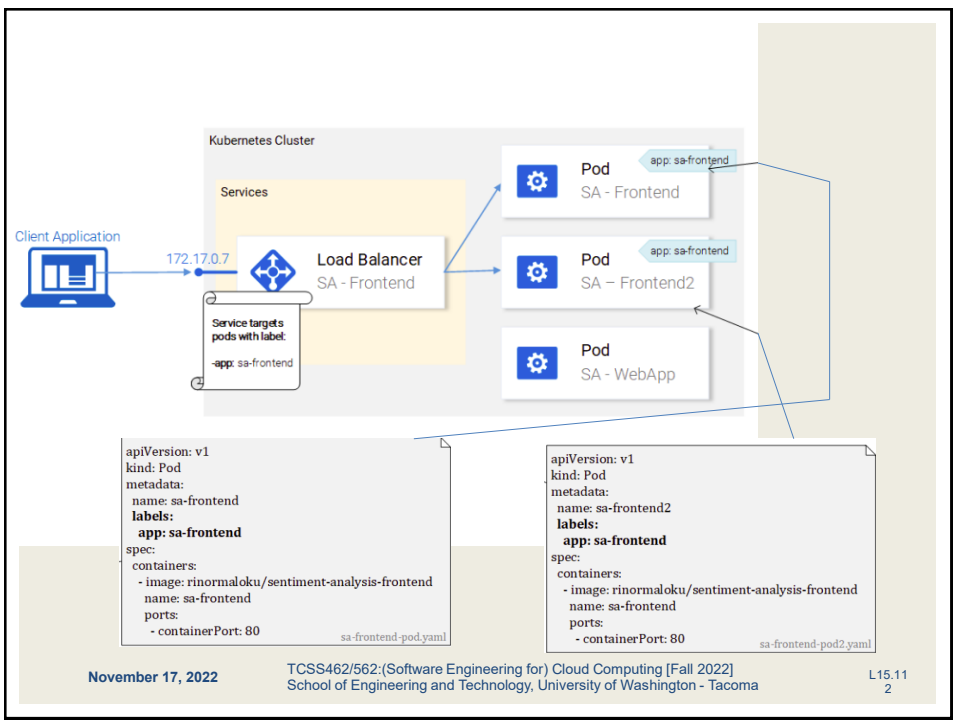
- Pods managed with "Deployments" or "DameonSets" controllers are automatically replaced when they die
  - This provides resiliency for the application
- **KEY IDEA**: Pods are unreliable
- **Services** provide reliability by acting as a "GATEWAY" to pods that implement the services
  - They underlying pods can change over time
  - The services endpoints remain and are always available
- Service objects provide an abstraction layer w/ a reliable name and load balancing of requests to a set of pods

110

SERVICES

- Provide reliable front-end with:
  - Stable DNS name
  - IP Address
  - Port
- Services do not posses application intelligence
- No support for application-layer host and path routing
- Services have a "label selector" which is a set of lables
- Requests/traffic is only sent to Pods with matching labels
- Services only send traffic to healthy Pods
- **KEY IDEA**: Services bring stable IP addresses and DNS names to unstable Pods

111



112

QUESTIONS

November 17, 2022

TCSS462/562:(Software Engineering for) Cloud Computing [Fall 202
School of Engineering and Technology, University of Washington - Tacoma

L15.11
3

113