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

Containerization

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

1

OFFICE HOURS – FALL 2024

- **THIS WEEK**
- **Tuesday:**
  - 2:30 to 3:30 pm - CP 229
- **Friday *:**
  - 1:30 pm to 2:30 pm –via Zoom*
- Or email for appointment

> Office Hours set based on Student Demographics survey feedback

2

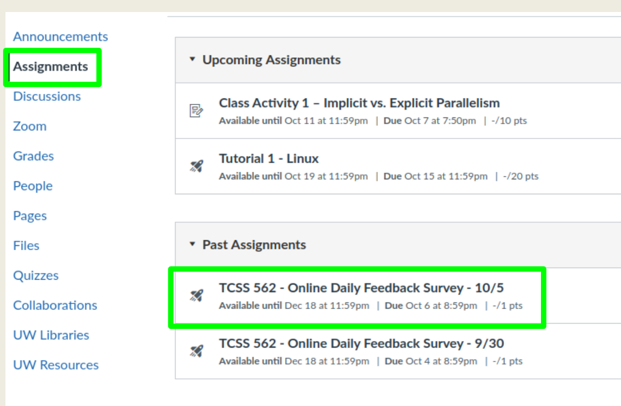
OBJECTIVES – 11/19

- Questions from 11/14
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Tutorial 8: AWS Step Functions, AWS SQS
- Containerization
- Kubernetes

3

ONLINE DAILY FEEDBACK SURVEY

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



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	

November 19, 2024 — TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024]
School of Engineering and Technology, University of Washington - Tacoma — L16.5

5

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (**42** respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 5.31 (↓ - *previous 5.60*)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.10 (↓ - previous 5.42)**

- **Response rates:**
- TCSS 462: 28/42 – 66.6%
- TCSS 562: 14/20 – 70.0%

November 19, 2024 — TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024]
School of Engineering and Technology, University of Washington - Tacoma — L16.6

6

FEEDBACK FROM 11/14

7

FEEDBACK FROM 11/16

- ***Why is it advantageous for containers to be run on top of VMs?***

- ***Why is it advantageous for containers to be run on top of bare metal?***

8

AWS CLOUD CREDITS UPDATE

- AWS CLOUD CREDITS ARE NOW AVAILABLE FOR TCSS 462/562
- Credits provided on request
- Credit codes must be securely exchanged
- Request codes by sending an email with the subject "**AWS CREDIT REQUEST**" to wlloyd@uw.edu
- Codes can also be obtained in person (or zoom), in the class, during the breaks, after class, during office hours, by appt
  - 57 credit requests fulfilled as of Nov 18 @ 11:59p
- Codes not provided using discord

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

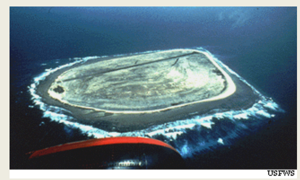
Tutorial 3 spot instance:
c5d.large instance @ ~3.2 cents / hour

$0.78 / day
$5.48 / week
$23.78 / month
$285.42 / year

AWS CREDITS → → → → → → → →

TUTORIAL SUBMISSION TIME

- Tutorials can now be submitted on the due date until the very last minute of the day **Anywhere-on-Earth (AOE)**
  - **Equivalent to 4:59 AM Pacific Standard Time (PST)**
- Anywhere-on-Earth timezone: **Baker Island, Pacific Ocean**
- https://www.timeanddate.com/time/zones/aoe
- Uninhabited island in Pacific Ocean
- Coordinates 0°11′45″N 176°28′45″W
- Area 2.1 km2 (0.81 sq mi)
- Length 1.81 km (1.125 mi)
- Width 1.13 km (0.702 mi)
- Coastline 4.8 km (2.98 mi)
- Highest elevation 8 m (26 ft)
- Population 0 (2000)



11

OBJECTIVES – 11/19

- Questions from 11/14
- **Tutorials Questions**
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Tutorial 8: AWS Step Functions, AWS SQS
- Containerization
- Kubernetes

12

TUTORIAL 5 – DUE NOV 14, LATE SUBMISSIONS UNTIL NOV 19

- Introduction to Lambda II: Working with Files in S3 and CloudWatch Events
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2024_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

13

TUTORIAL 6 – NOV 23

- Introduction to Lambda III: Serverless Databases
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2024_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

14

TUTORIAL 7 – DEC 1

- Introduction to Docker
- https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2024_tutorial_7.pdf
- Complete tutorial using Ubuntu 24.04 (for cgroups v2)
- Complete using **c6i.large ec2 instance** (for consistency)
- 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

OBJECTIVES – 11/19

- Questions from 11/14
- Tutorials Questions
- **Class Presentations Schedule - Cloud Technology or Research Paper Review**
- Tutorial 8: AWS Step Functions, AWS SQS
- Containerization
- Kubernetes

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 26
  - Tuesday December 3, Thursday December 5
- **Peer Reviews**
  - Word DOCX form will be provided, fill out, submit PDF on Canvas
  - Feedback shared with groups
  - TCSS 462: submit 4 total peer reviews in lieu of a group presentation

GROUP PRESENTATIONS

- 9 Presentation Teams
- 3 Cloud Technology Talks
- 6 Cloud Research Paper Presentations
- 2 one-person teams
- 4 two-person teams
- 3 three-person teams

- Thank you for the submissions

PRESENTATION SCHEDULE

- **<Tuesday November 26>**

1. Soumith Kondubhotla, Siva Srinivasa Aditya, Sri Mylavarapu
Research paper: ***Sandboxing Functions for Efficient and Secure Multi-tenant Serverless Deployments***

2. Mingzhi Ma, Derry Cheng, Aaron Chen
Research paper: ***Serverless? RISC more!***

3. Ishwarya Narayana Subramanian, Thanvi Yadav Sirla
Cloud Technology: ***Azure Kubernetes Service***

4. Steven Golob
Research paper: ***Tiny Autoscalers for Tiny Workloads: Dynamic CPU Allocation for Serverless Functions***

- **<Tuesday December 3>**

1. Andrew Nguyen, Pavel Braginskiy
Cloud Technology: ***AWS Amplify***

PRESENTATION SCHEDULE - 2

- **<Thursday December 5>**

1. Viktoria Dolojan and Carla Peterson
Research paper: ***FootPrinter: Quantifying Data Center Carbon Footprint***

2. Andrew Jang, Shrey Srivastava, Naga
Cloud Technology: ***SageMaker: training configurations***

3. Roark Zhang
Research paper: ***Process-as-a-Service: Unifying Elastic and Stateful Clouds with Serverless Processes***

4. Sanya Sinha, Jackson Davis
Research paper: ***Goldfish: Serverless Actors with Short-Term Memory State for the Edge-Cloud Continuum***

OBJECTIVES – 11/19

- Questions from 11/14
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- **Tutorial 8: AWS Step Functions, AWS SQS**
- Containerization
- Kubernetes

November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.21

21

TUTORIAL 8 – TO BE POSTED

- Introduction to AWS Step Functions and Amazon Simple Queue Service (SQS)
- Not Required, available for extra credit (scored out of 0)
  - adds points to overall tutorials score
- Tasks
  - Adapt Caesar Cipher Lambda functions for use with AWS Step Functions
  - Create AWS Step Functions State Machine
  - Create a BASH client to invoke the AWS Step Function
  - Create Simple Queue Service Queue for messages
  - Add message to SQS queue from AWS Lambda function
  - Modify AWS Step Function Bash client script to retrieve AWS Step Function result from SQS queue

November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.22

22

OBJECTIVES – 11/19

- Questions from 11/14
- Tutorials Questions
- Class Presentations Schedule -
  Cloud Technology or Research Paper Review
- Tutorial 8: AWS Step Functions, AWS SQS
- **Containerization**
- Kubernetes

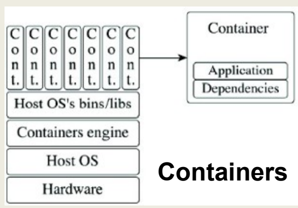
23

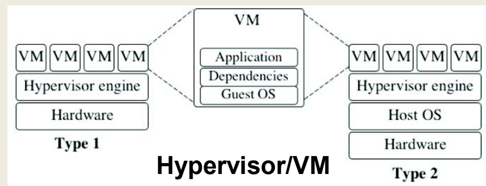
CONTAINERIZATION

24

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





25

CONTAINER PERFORMANCE – LU FACTORIZATION PERFORMANCE

- Solve linear equations – matrix algebra

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

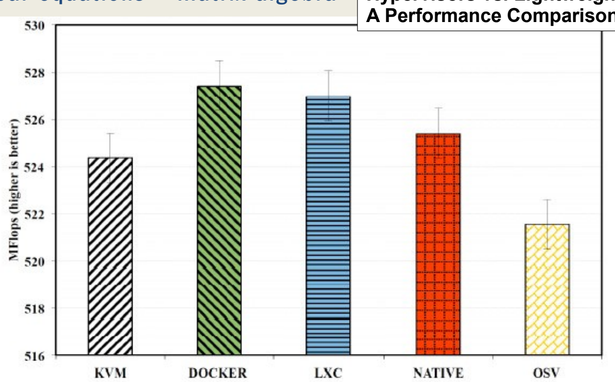
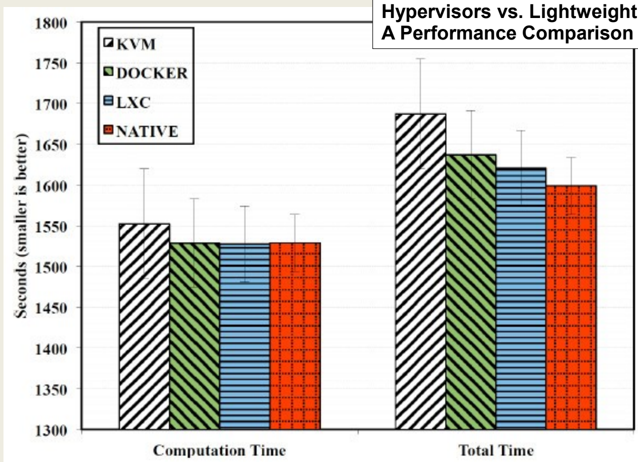


Fig. 4. The value of Linpack results on each platform over 15 runs. This is the particular case of N=1000.

26

CONTAINER PERFORMANCE – Y-CRUNCHER: PI CALCULATOR

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



November 19, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024]
School of Engineering and Technology, University of Washington - Tacoma | L16.27

27

CONTAINER PERFORMANCE – BONNIE++

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

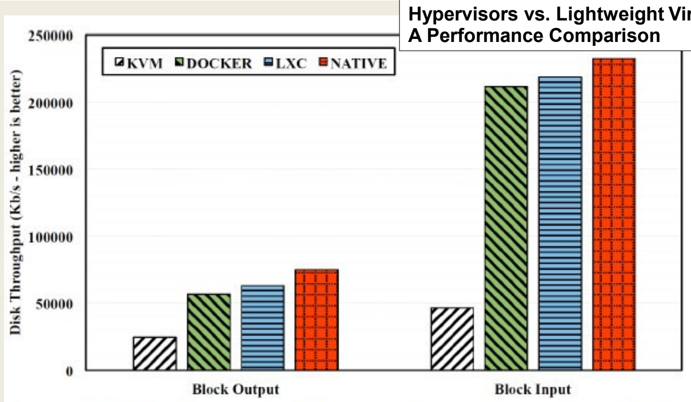


Fig. 6. Disk Throughput achieved by running Bonnie++ (test file of 25 GiB). Results for sequential writes and sequential read are shown.

November 19, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024]
School of Engineering and Technology, University of Washington - Tacoma | L16.28

28

WHAT IS A CONTAINER?

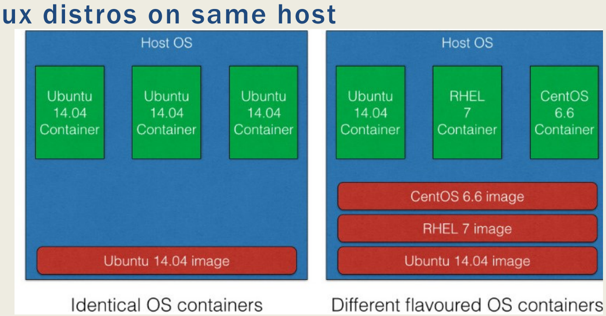
According to NIST (National Institute of Standards Technology)

- **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

29

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/

30

APPLICATION CONTAINERS

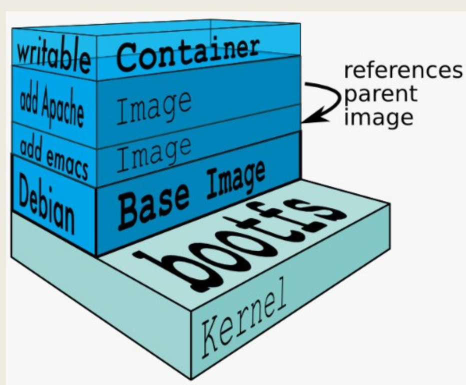
- Designed to package and run a single service
- All containers share host kernel
- Subtle differences from operating system containers
- Examples: Docker, Rocket
- Docker: runs a single process on creation
- OS containers: run many OS services, for an entire OS
- Create application containers for each 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 19, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.31

31

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

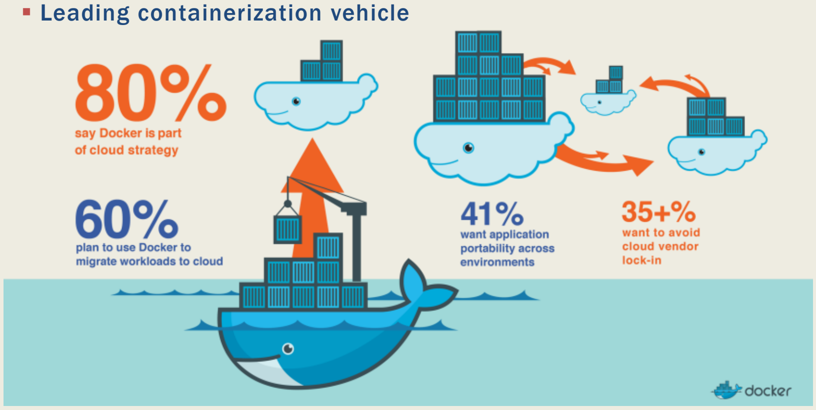


November 19, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.32

32

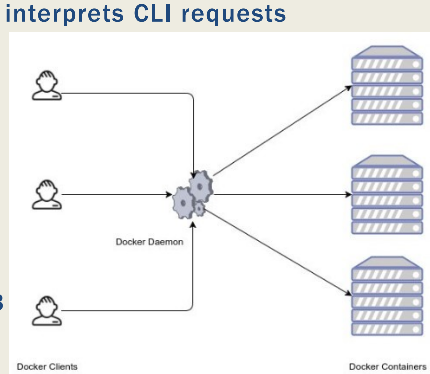
2016 DOCKER SURVEY

- 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

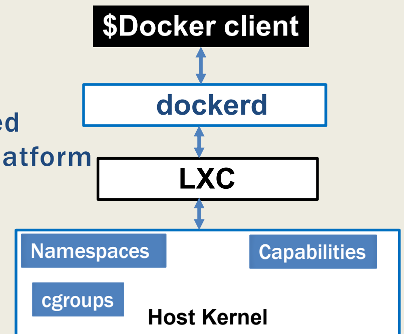


Docker Client-Server Architecture

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

ORIGINAL DOCKER ENGINE IMPLEMENTATION

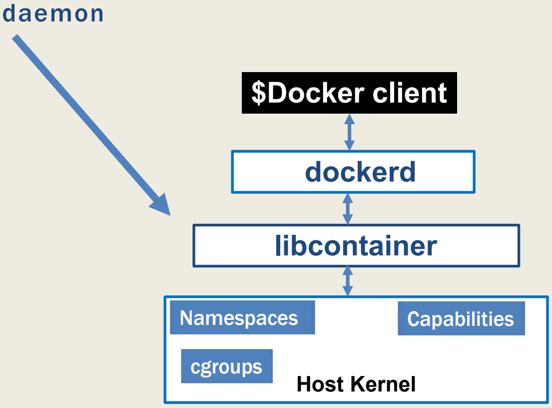
- (1) Original Docker engine relied on LXC
  - LXC itself is a containerization tool predating Docker
  - Original Docker API just called it
  - LXC originally provided access to Linux kernel features: namespaces and cgroups
  - LXC was Linux specific – caused issues if wanting to be multi-platform
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



35

INTRODUCTION OF LIBCONTAINER

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



36

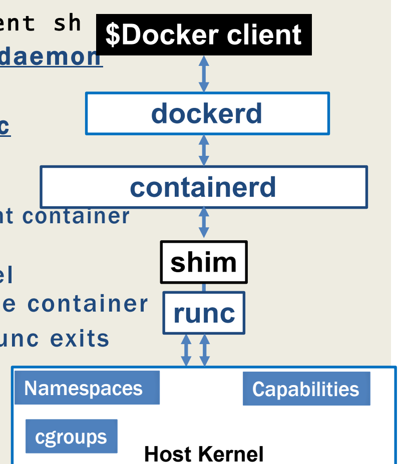
OPEN CONTAINER INITIATIVE (OCI)

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

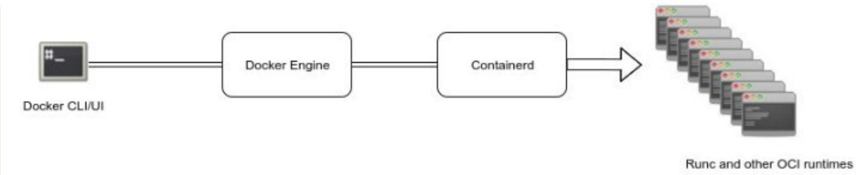
CREATING A CONTAINER

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

- Docker CLI posts request to Docker daemon
- Daemon calls containerd
- Containerd passes of request to runc
  - Containerd converts docker image into OCI compliant bundle
  - This step would allow any OCI compliant container to be plugged into the back-end
- Runc interfaces with the Linux kernel (namespaces, cgroups, etc.) to create container
- Shim: once a container is created, runc exits
  - Shim remains as a daemonless stub to implement the container
  - Allows Docker to be upgraded w/o stopping the container !!!



CREATING A CONTAINER - 2

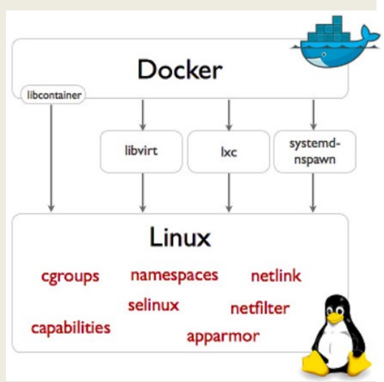


Containerd Integration Architecture

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

SUPPORT FOR ALTERNATE CONTAINER RUNTIMES

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

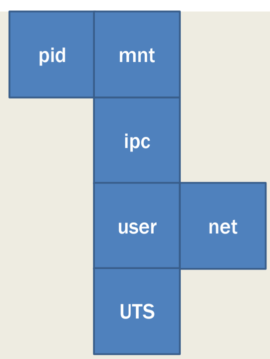


WE WILL RETURN AT ~4:50 PM

41

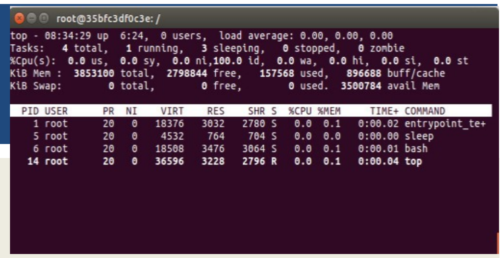
LINUX KERNEL NAMESPACES

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



42

NAMESPACES - 2

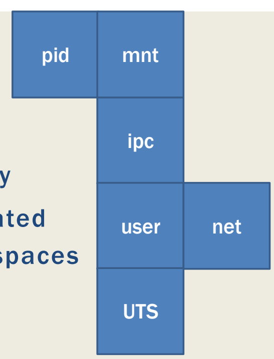


- **Provides isolation of OS entities for containers**
- **mnt**: separate filesystems
- **pid**: independent PIDs; first process in container is PID 1
- **ipc**: prevents processes in different IPC namespaces from being able to establish shared memory. Enables processes in different containers to reuse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

43

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



44

CONTROL GROUPS (CGROUPS)

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

CGROUPS - 2

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

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

OVERLAY FILE SYSTEMS

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

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

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

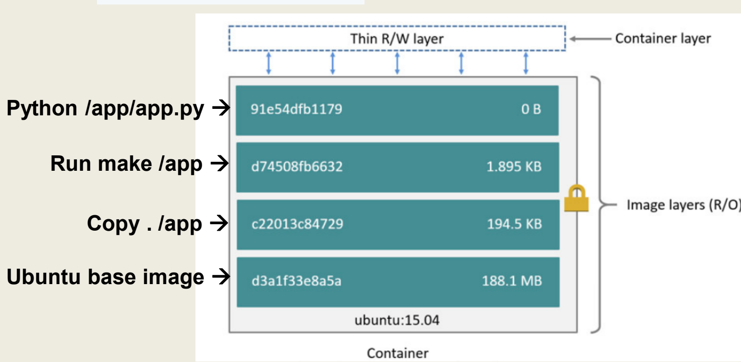
- https://www.slideshare.net/jpetazzo/scale11x-lxc-talk-1/

47

LAYERED FS: BUILDING A CONTAINER

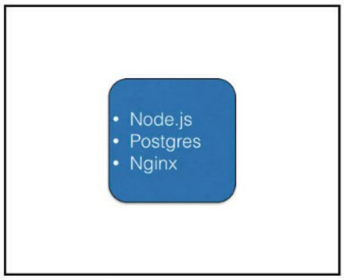
- Dockerfile:

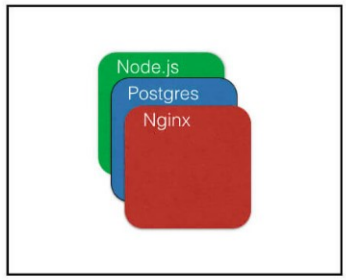
```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



48

THREE-TIER ARCHITECTURE





OS containers

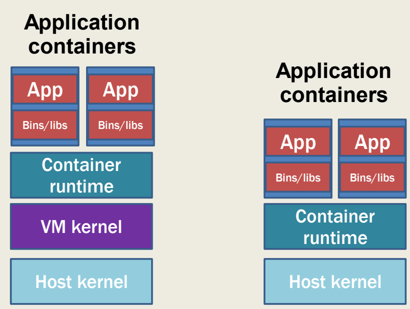
- Meant to used as an OS - run multiple services
- No layered filesystems by default
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, OpenVZ, Linux VServer, BSD Jails, Solaris Zones

App containers

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

CONTAINER ISOLATION

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



LXC (LINUX CONTAINERS)

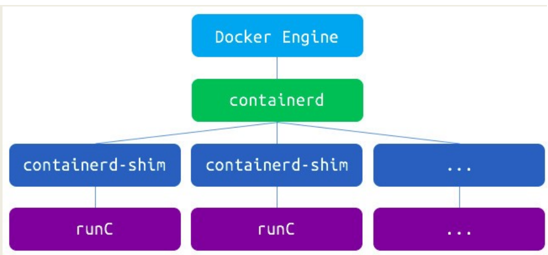
- Operating system level virtualization
- Run multiple isolated Linux systems on a host using a single Linux kernel
- Control groups(cgroups)
  - Including in Linux kernels => 2.6.24
  - Limit and prioritize sharing of CPU, memory, block/network I/O
- Linux namespaces
- Docker initially based on LXC

November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.51

51

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 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.52

52

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

53

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

54

CONTAINER ORCHESTRATION FRAMEWORKS - 2

- Docker swarm
- Apache mesos/marathon
- Kubernetes
  - Many public clouds now offer managed services to host Kubernetes clusters
  - Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), Google Kubernetes Engine (GKE)
- Amazon elastic container service (ECS)
- Apache aurora (retired project based on Mesos)

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

OBJECTIVES – 11/19

- Questions from 11/14
- Tutorials Questions
- Class Presentations Schedule - Cloud Technology or Research Paper Review
- Tutorial 8: AWS Step Functions, AWS SQS
- Containerization
- **Kubernetes**

KUBERNETES

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

L16.57

57

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 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.58

58

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

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

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 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.61

61

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 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.62

62

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

November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.63

63

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 the goal of keeping it running smoothly

November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.64

64

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 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.65

65

LOOK AHEAD: 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**

docker

November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.66

66

DECLARATIVE SERVICE APPROACH

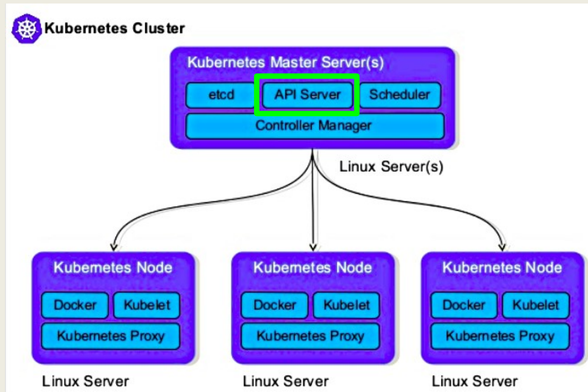
- **Imperative definition**: sets of commands and operations
  - Example: BASH script, Dockerfile
- **Declarative definition**: specification of a service's properties
  - What level of service it should sustain, etc.
  - Example: Kubernetes YAML files
- Kubernetes manages resources **declaratively**
- 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*

KUBERNETES MASTERS

- Provide system services to host the control plane
- Simplest clusters use only 1 master – (i.e. 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

MASTER SERVICES

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



November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.69

69

API SERVER

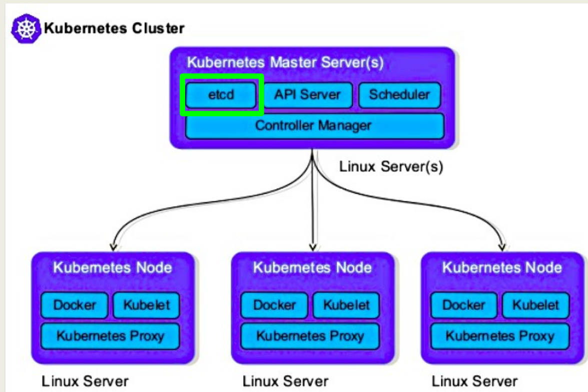
- Can run on 1-node for lab, test/dev environments
- Default port is 443
- Exposes a RESTful API where YAML configuration files are POST(ed) to
- YAML files (manifests) describe desired state of an application
  - Which container image(s) to use
  - Which ports to expose
  - How many POD replicas to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

November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.70

70

MASTER SERVICES

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

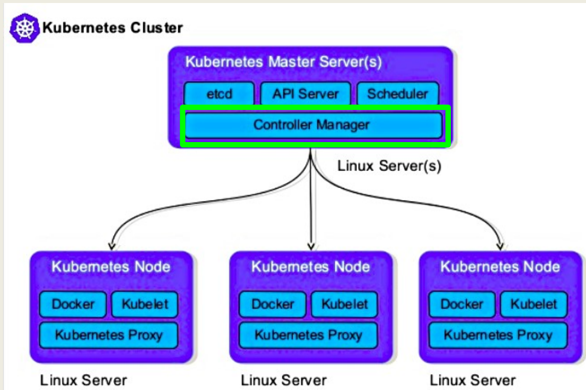


CLUSTER STORE

- Used to persist Kubernetes cluster state information
- 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

MASTER SERVICES

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



73

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:

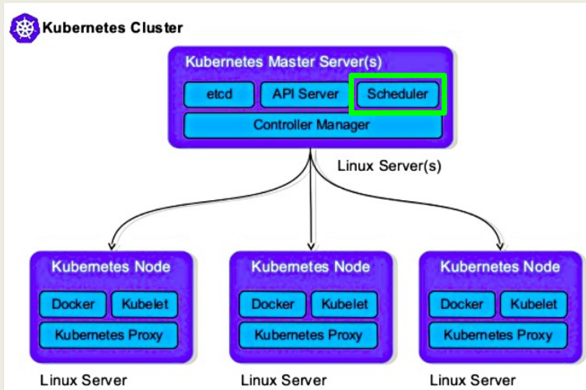
1. Obtain desired state (defined in manifest YAMLs)
2. Observe the current state
3. Determine differences
4. Reconcile differences

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

74

MASTER SERVICES

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



75

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

76

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

77

RANKING NODES

- Remaining nodes are ranked based on for example:

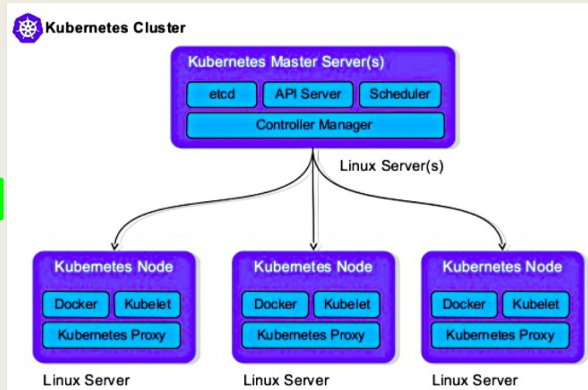
1. Does the node have the required images?
   - Cached images will lead to faster deployment time
2. How much free capacity (CPU, memory) does the node have?
3. How many tasks is the node already running?

- 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*

78

MASTER SERVICES

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

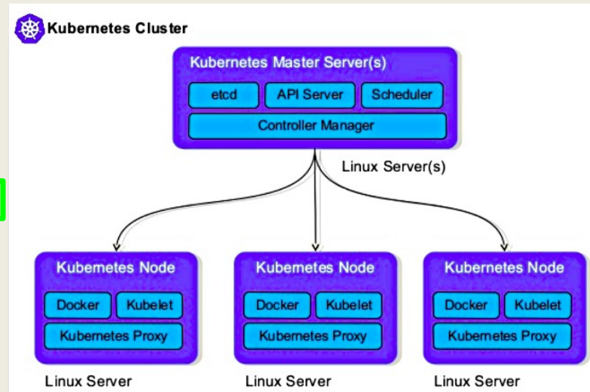


CLOUD CONTROLLER MANAGER

- 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.

MASTER SERVICES

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



November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.81

81

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.

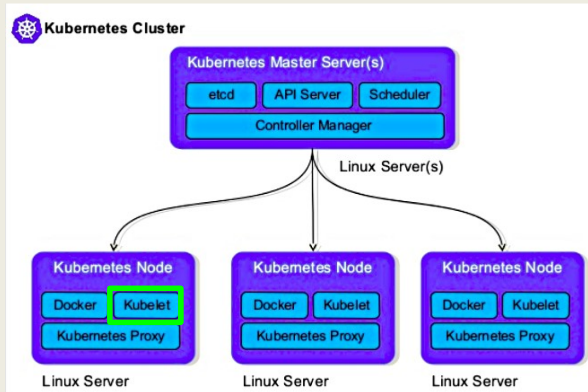
- Nodes are considerably simpler than masters

November 19, 2024	TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma	L16.82

82

WORKER NODES

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

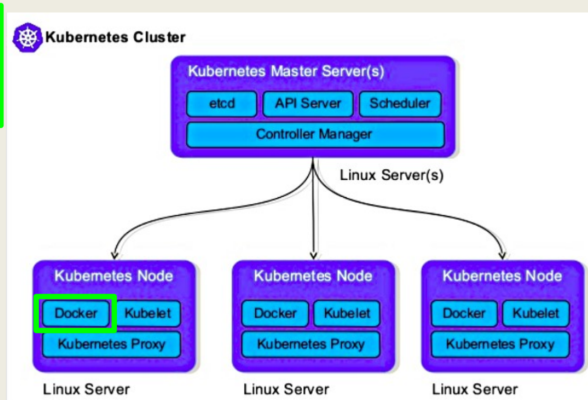


KUBELET

- 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

WORKER NODES

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



85

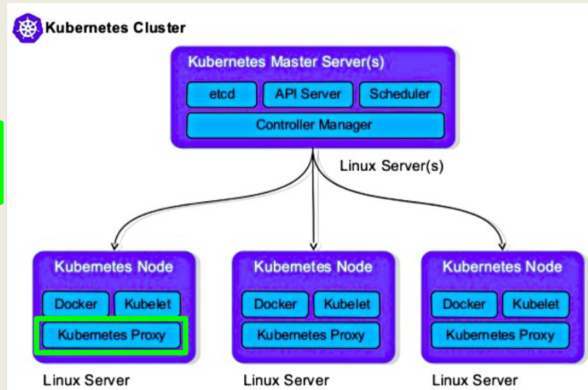
CONTAINER RUNTIME(S)

- 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

86

WORKER NODES

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



87

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

88

CORE KUBERNETES COMPONENTS

- Kubernetes DNS
- Pods
- Services

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)

CORE KUBERNETES COMPONENTS

- Kubernetes DNS
- Pods
- Services

November 19, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.91

91

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

November 19, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.92

92

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

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

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

CORE KUBERNETES COMPONENTS

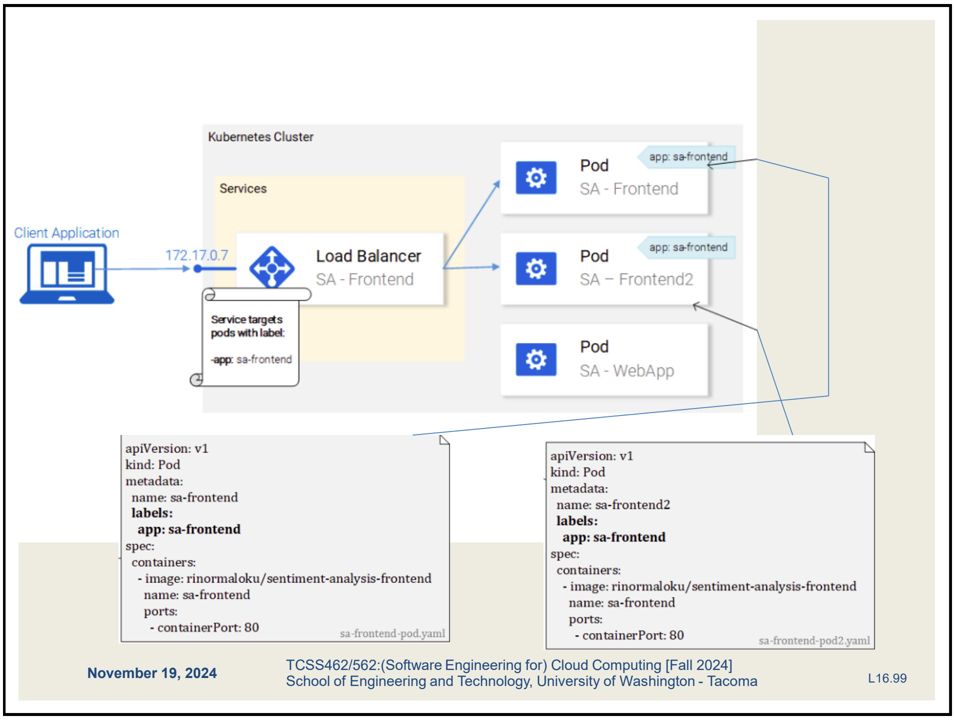
- Kubernetes DNS
- Pods
- Services

KUBERNETES "SERVICES"

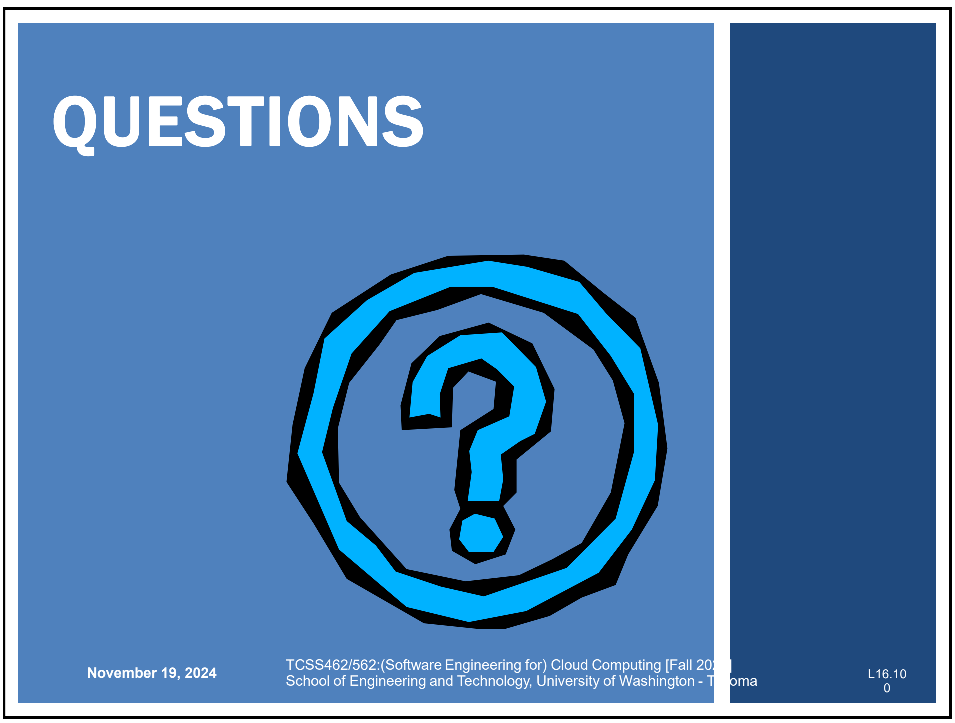
- Pods managed with "Deployments" or "DameonSets" controllers are automatically replaced when they die
  - This provides resiliency for the application
- **KEY IDEA**: Pods are unreliable
- **Services** provide reliability by acting as a "GATEWAY" to pods that implement the services
  - They underlying 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

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


Kubernetes Cluster
Services
Client Application
172.17.0.7
Load Balancer
SA - Frontend
Service targets pods with label:
-app: sa-frontend
Pod
SA - Frontend
app: sa-frontend
Pod
SA – Frontend2
app: sa-frontend
Pod
SA - WebApp
apiVersion: v1
kind: Pod
metadata:
name: sa-frontend
labels:
app: sa-frontend
spec:
containers:
- image: rinormaloku/sentiment-analysis-frontend
name: sa-frontend
ports:
- containerPort: 80
sa-frontend-pod.yaml
apiVersion: v1
kind: Pod
metadata:
name: sa-frontend2
labels:
app: sa-frontend
spec:
containers:
- image: rinormaloku/sentiment-analysis-frontend
name: sa-frontend
ports:
- containerPort: 80
sa-frontend-pod2.yaml
November 19, 2024
TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024]
School of Engineering and Technology, University of Washington - Tacoma
L16.99

99


QUESTIONS
November 19, 2024
TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024]
School of Engineering and Technology, University of Washington - Tacoma
L16.100

100