TCSS 562: SOFTWARE ENGINEERING FOR CLOUD COMPUTING

Cloud Enabling Technology, Containerization

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

MW 5:50-7:50 PM

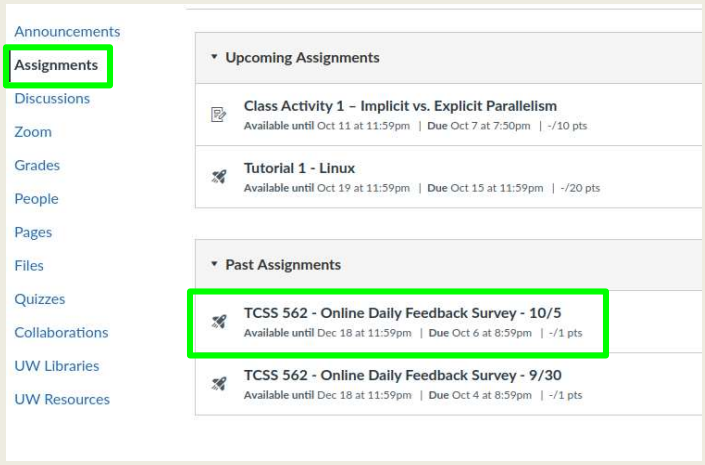
OBJECTIVES – 11/16

- Questions from 11/9
- Quiz 2 – to be posted this week
- Group Presentations for 11/30 – 12/9
- From: Cloud Computing Concepts, Technology & Architecture: Chapter 5 - Cloud Enabling Technology

- 2nd hour:
- Introduction to Containerization
- Tutorial questions (4, 5, 6)
- Team planning

ONLINE DAILY FEEDBACK SURVEY

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



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

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (22 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.55** (↑ - *previous 6.30*)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.41** (↑ - *previous 5.40*)

TUTORIAL QUESTIONS

- I am receiving ***many*** emails regarding the tutorials
- Many emails are regarding computer issues *(setting up required software dependencies, etc.)*
- Other emails are regarding clarifications on what the assignments require
  - When confusing points are found, I make every effort to capture valuable feedback, and post a revision to the tutorial
  - Tutorials are living documents – your feedback and participation actively makes them better - **THANK YOU** !!!
  - AWS is continuously changing
    - For example – location of where a function handler is configured in the AWS Lambda GUI changed between posting of Tutorial 4 and 5
  - Ubuntu and our working environments are continuously changing

FEEDBACK FROM 11/9

- ***Can you elaborate on the differences between AWS Lambda VPC and NO VPC function deployments mentioned in tutorial 4?***
- **Virtual Private Clouds** (VPCs) enable users to customize network settings and create virtual networks with unique routing rules
- There are two aspects of networking:
  - **Security groups** – define firewall rules which describe which types of network traffic is allowed to pass across the connection
  - **Routing rules** – defines virtual networking paths that enable traffic to transit in various ways

Is a security group (firewall) rule necessary in AWS to prevent undesired traffic between two cloud services where a network route has NOT been established and the cloud services are deployed on different subnetworks?

- YES, always
- NO, never
- Rules only needed for inbound traffic
- Rules only needed for outbound traffic

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

FEEDBACK - 2

- **Can you talk about why you didn't want us to call the Lambda function using AWS CLI for Caesar cipher ?**
- For testing purposes, invoking a Lambda function using the AWS CLI requires access credentials from tutorial 0
- These are the **access_key** and **secret_key**
- For the instructor to test your encryption pipeline using the AWS CLI, you would need to provide access credentials to a user defined in your AWS account with permission to access the Lambda functions
- ***Setting up an IAM User & credentials requires additional effort***
- ***Distributing keys securely is arduous and fault prone***

FEEDBACK - 3

- For Tutorial 4 - ***Security through Obscurity***:
  - API gateway endpoints are unlikely to be attacked as the URIs are quite cryptic
- ***PLEASE DO DELETE API GATEWAY ENDPOINTS ONCE TUTORIAL RECEIVES A GRADE***

FEEDBACK - 4

- It would be great to have a "mind map" of the tutorials we do, since sometimes it is hard to see the broader picture after reading and doing it through.
- What I mean is nothing detailed, just a general connection between the tutorials and bullet points within a tutorial. Thank you!

TUTORIALS

- Tutorials 1-7 graded and required
- Tutorials 8+ optional
- Up to two optional tutorials can be completed to replace the grade for tutorials 1-7

TUTORIAL 4 OBJECTIVES

- Provides introduction to:
- AWS Lambda
- API Gateway for HTTP/REST endpoints
- AWS CLI
- Serverless Application Analytics Framework (SAAF)
  - Deploy script (***optional***)
  - FaaS Runner Introduction: supports running experiments

TUTORIAL 5 OBJECTIVES

- Provides introduction to:
- Simple Storage Service
- Including dependencies in MAVEN projects (AWS libraries, etc.)
- Security: use of roles and policies to manage fine-grained access between AWS services
- CloudTrail: key idea is to expose events to CloudWatch
- CloudWatch: Rules to invoke targets when events occur
- CloudWatch: Viewing log files for AWS Lambda functions
- Providing event data to Lambda so that function can obtain metadata about the triggering event (***optional***)

TUTORIAL 6 OBJECTIVES

- SQLite databases & command-line client
- Using SQLite from Lambda
- Lambda function persistent data (static variables in Java)
- Concurrent Lambda function calls and function instances (newcontainer attribute in SAAF)
- AWS Aurora MySQL Serverless databases
- Mysql command-line client
- Lambda functions w/ Virtual Private Clouds (VPCs)
- Using Aurora MySQL from Lambda
- Serverless freeze/thaw lifecycles

UPCOMING TUTORIALS

- Tutorial 7 – Introduction to Docker Containerization
- Going further - optional tutorials:
  - Ungraded or substitute
- Tutorial 8 – Introduction to FaaS IV: Step Functions and SQS
- Tutorial 9 – Asynchronous Function Profiling with SAAF
- Tutorial 10 – Automating Experiments with SAAF & FaaS Runner
- Tutorial 11 – Scaling beyond a single client – concurrent webservice benchmarking with multiple EC2 instances

OBJECTIVES – 11/16

- Questions from 11/9
- Quiz 2 – to be posted this week
- Group Presentations for 11/30 – 12/9
- From: Cloud … Architecture: Chapter 5 -
- 2nd hour:
- Introduction
- Tutorial que
- Team planning

Quiz 2 Coverage:

- Focus on lectures 7 – 12
- AWS
- Tutorials

OBJECTIVES – 11/16

- Questions from 11/9
- Quiz 2 – to be posted this week
- Group Presentations for 11/30 – 12/9
- From: Cloud Computing Concepts, Technology & Architecture: Chapter 5 - Cloud Enabling Technology
- 2nd hour:
- Introduction to Containerization
- Tutorial questions (4, 5, 6)
- Team planning

GROUP PRESENTATION

- TWO OPTIONS:
- *Cloud technology presentation*
- *Cloud research paper presentation*
  - Recent & suggested papers will be posted at: http://faculty.washington.edu/wlloyd/courses/tcss562/papers/
- Submit presentation type and topics (paper or technology) with desired dates of presentation via Canvas by Monday November 23rd @ 11:59pm
- Presentation dates:
  - Monday November 30, Wednesday December 2
  - Monday December 7, Wednesday December 9

OBJECTIVES – 11/16

- Questions from 11/9
- Quiz 2 – to be posted this week
- Group Presentations for 11/30 – 12/9
- From: Cloud Computing Concepts, Technology & Architecture: Chapter 5 - Cloud Enabling Technology
- 2nd hour:
- Introduction to Containerization
- Tutorial questions (4, 5, 6)
- Team planning

CLOUD ENABLING TECHNOLOGY

November 16, 2020

TCSS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L13.21

VIRTUALIZATION MANAGEMENT

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

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

November 16, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L13.22

VIRTUAL INFRASTRUCTURE MANAGEMENT (VIM)

- Middleware to manage virtual machines and infrastructure of IaaS "clouds"
- Examples
  - OpenNebula
  - Nimbus
  - Eucalyptus
  - OpenStack

VIM FEATURES

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

VIM FEATURES - 2

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

VIM FEATURES - 3

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

CONTAINER ORCHESTRATION FRAMEWORKS

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

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

- **Proprietary**
- Amazon Elastic Container Service

CONTAINER SERVICES

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

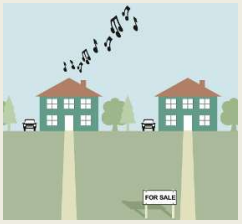
- **Container-as-a-Service**
- Azure Container Instances (ACI – April 2018)
- AWS Fargate (November 2017)
- Google Kubernetes Engine Serverless Add-on (alpha-July 2018)

CLOUD ENABLING TECHNOLOGY

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

4. MULTITENANT APPLICATIONS

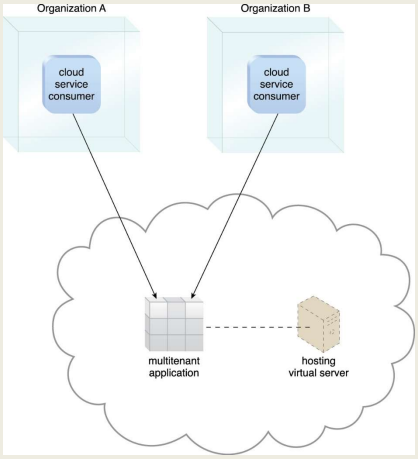
- Each tenant (like in an apartment) has their own view of the application
- Tenants are unaware of their neighbors
- Tenants can only access their data, no access to data and configuration that is not their own



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

MULTITENANT APPS - 2

- Forms the basis for SaaS (applications)



CLOUD ENABLING TECHNOLOGY

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

5. WEB SERVICES/WEB

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

HYPERTEXT TRANSPORT PROTOCOL (HTTP)

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

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

REST: REPRESENTATIONAL STATE TRANSFER

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


```
// SOAP REQUEST

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

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

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

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



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

REST CLIMATE SERVICES EXAMPLE

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

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

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

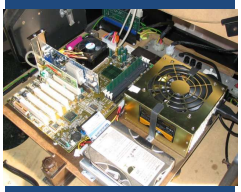

REST - 2

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

REST ARCHITECTURAL ADVANTAGES

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

WE WILL RETURN AT ~7:13PM

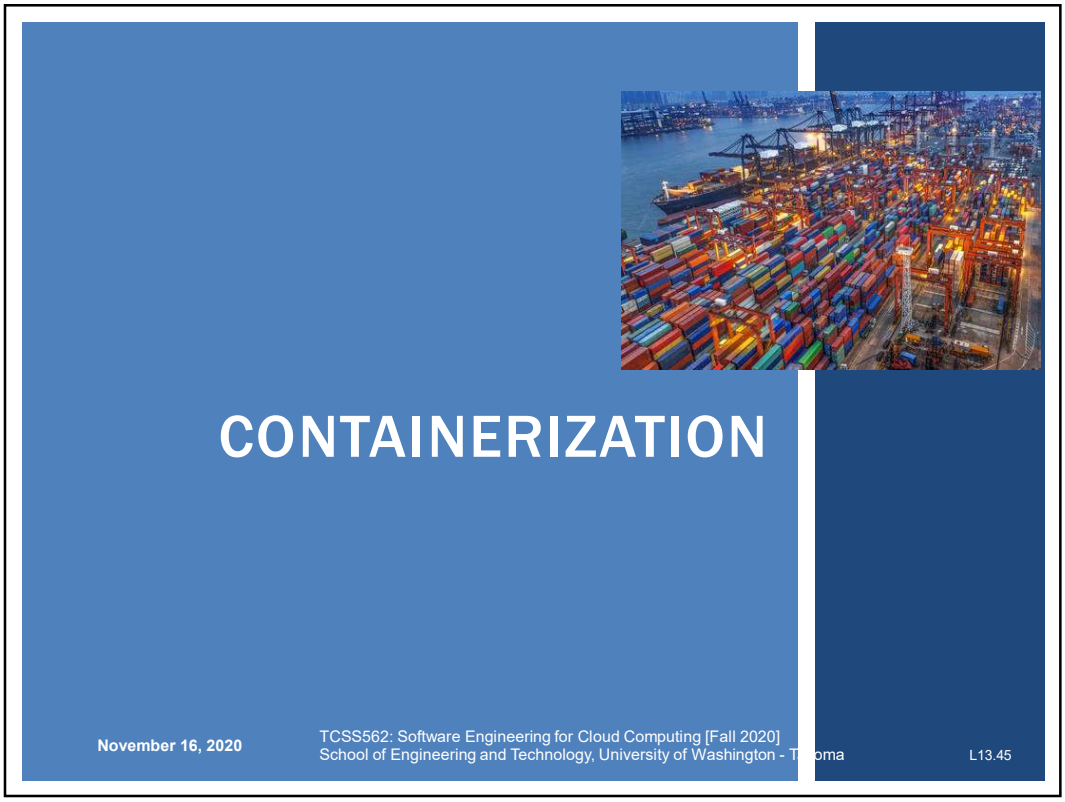


OBJECTIVES – 11/16

- Questions from 11/9
- Quiz 2 – to be posted this week
- Group Presentations for 11/30 – 12/9
- From: Cloud Computing Concepts, Technology & Architecture: Chapter 5 - Cloud Enabling Technology
- 2nd hour:
- Introduction to Containerization
- Tutorial questions (4, 5, 6)
- Team planning

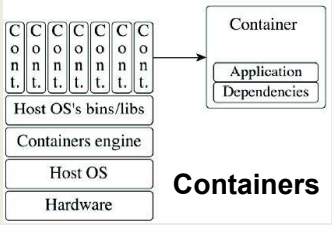
OBJECTIVES – 11/16

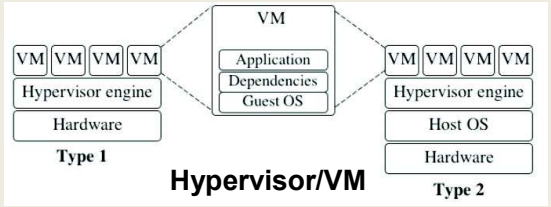
- Questions from 11/9
- Quiz 2 – to be posted this week
- Group Presentations for 11/30 – 12/9
- From: Cloud Computing Concepts, Technology & Architecture: Chapter 5 - Cloud Enabling Technology
- 2nd hour:
- Introduction to Containerization
- Tutorial questions (4, 5, 6)
- Team planning



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





CONTAINER PERFORMANCE – LU FACTORIZATION PERFORMANCE

- Solve linear equations – matrix algebra

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

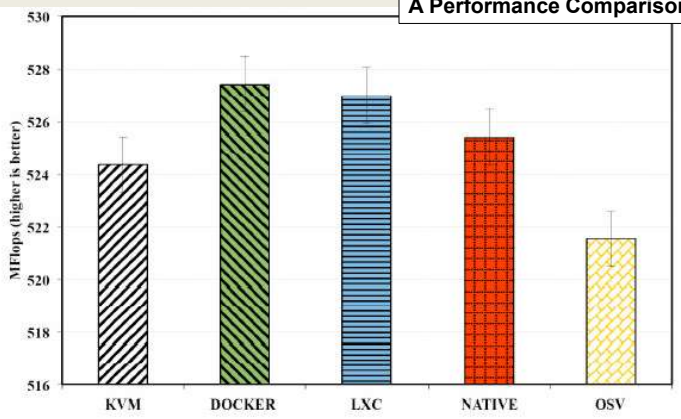
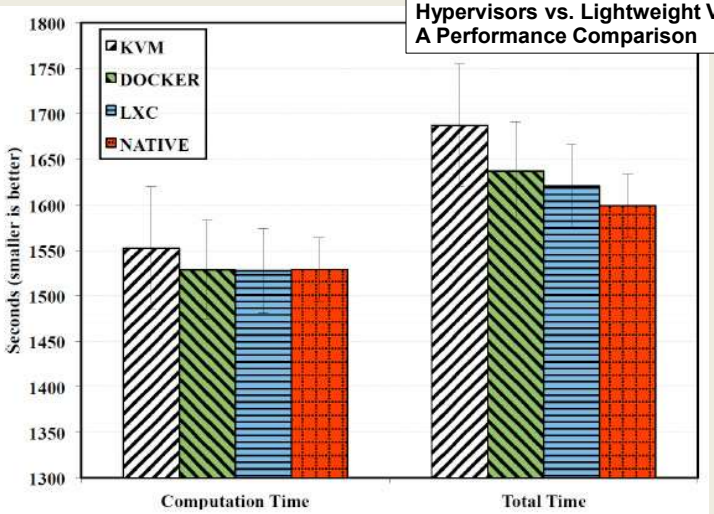


Fig. 4. The value of Linpack results on each platform over 15 runs. This is the particular case of N=1000.

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

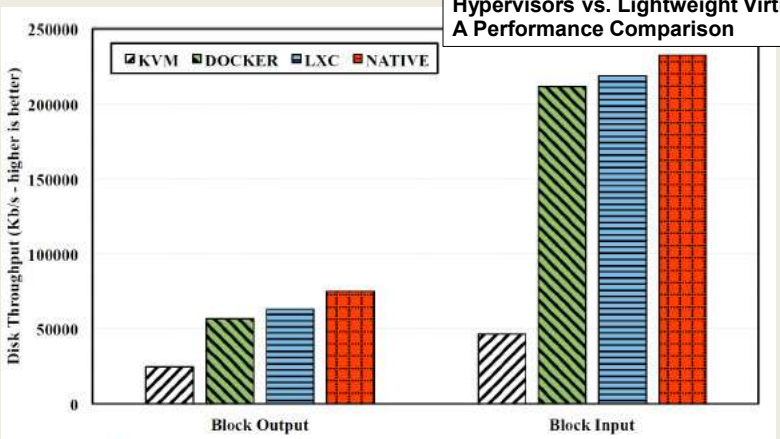


Fig. 6. Disk Throughput achieved by running Bonnie++ (test file of 25 GiB). Results for sequential writes and sequential read are shown.

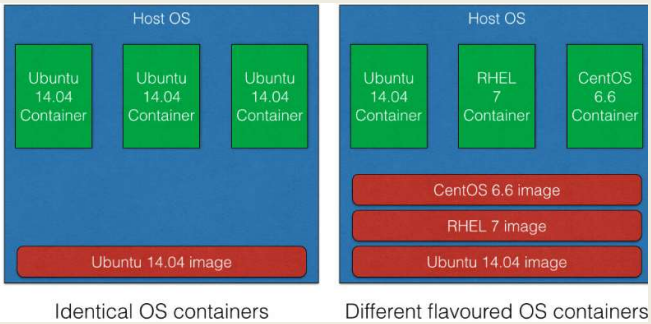
WHAT IS A CONTAINER?

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

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



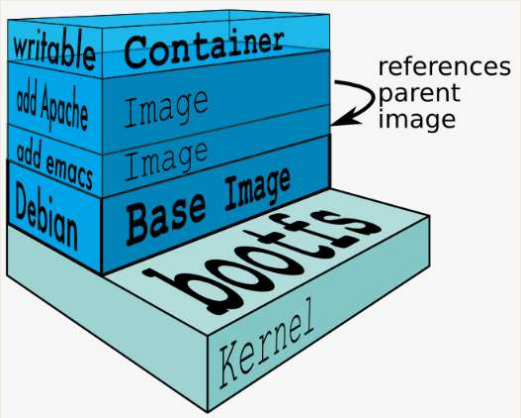
- Credit: https://blog.risingstack.com/operating-system-containers-vs-application-containers/

APPLICATION CONTAINERS

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

APPLICATION CONTAINERS - 2

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



OVERLAY FILE SYSTEMS

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

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

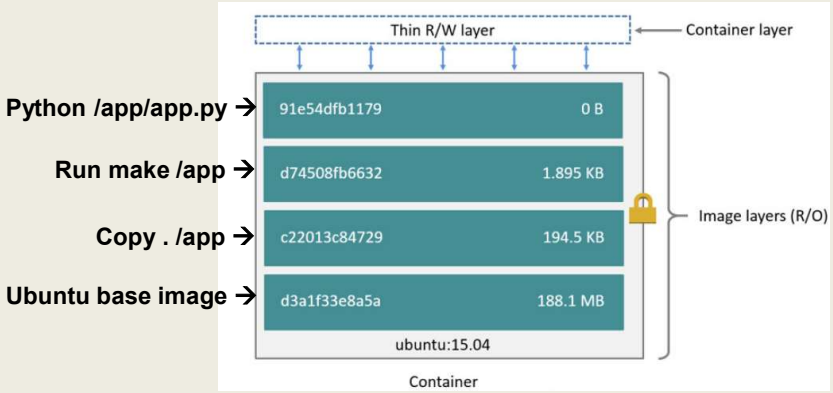
- https://medium.com/@nagarwal/docker-containers-filesystem-demystified-b6ed8112a04a

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

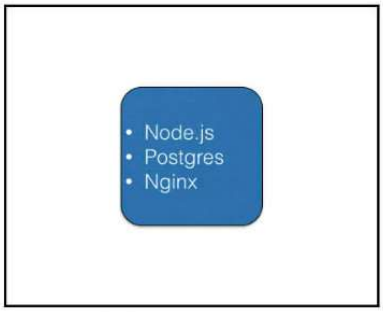
LAYERED FS: BUILDING A CONTAINER

- **Dockerfile:**

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

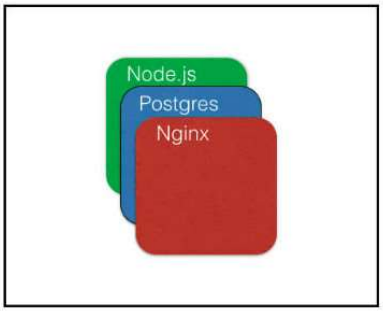


THREE-TIER ARCHITECTURE



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

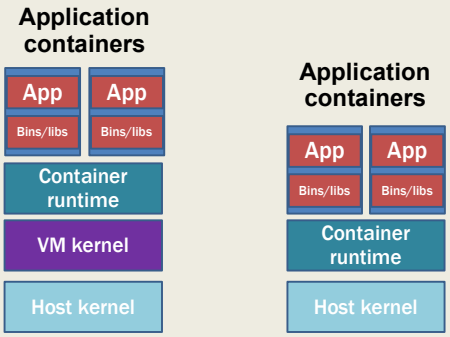


App containers

- 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?

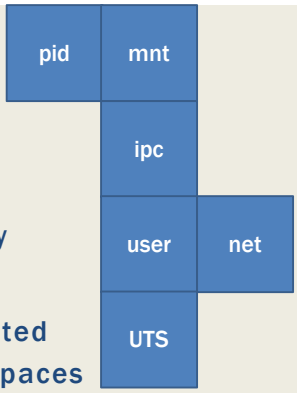


LXC (LINUX CONTAINERS)

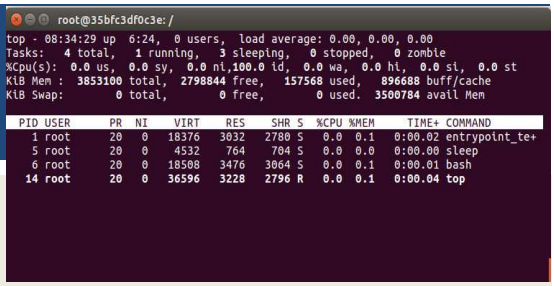
- Operating system level virtualization
- Run multiple isolated Linux systems on a host using a single Linux kernel
- Control groups(cgroups)
  - Including in Linux kernels => 2.6.24
  - Limit and prioritize sharing of CPU, memory, block/network I/O
- Linux namespaces
- Docker initially based on LXC

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



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

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

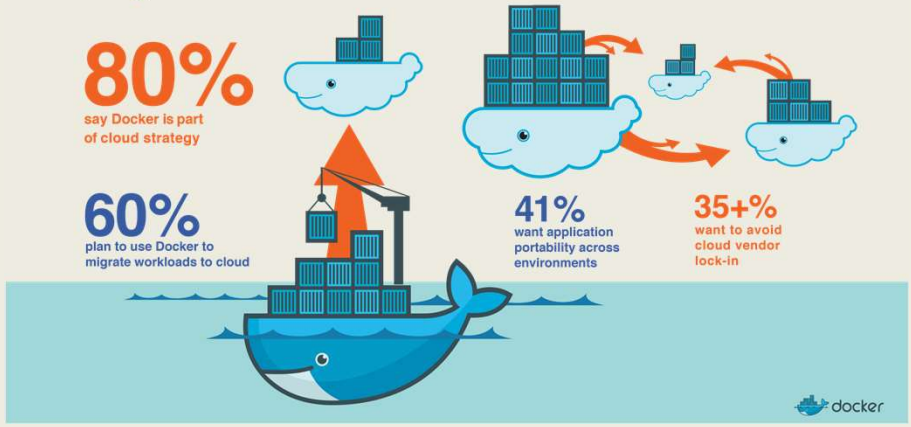
CGROUPS - 2

- 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

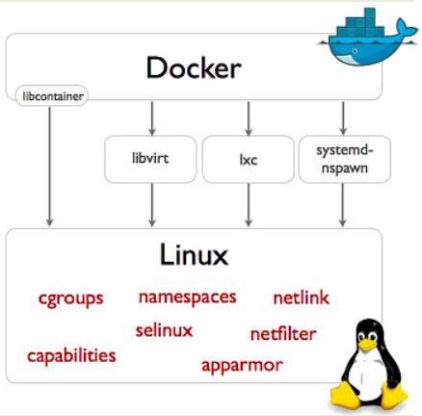
2016 DOCKER SURVEY

- Docker application containers
  - Leading containerization vehicle



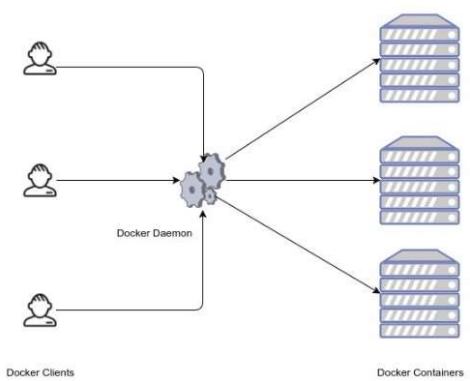
DOCKER EXECUTION ENVIRONMENTS

- (1) Original default Docker execution enviornment: LXC
- (2) Docker v0.9: libcontainer introduced (~2014)
- (3) Now runc (2015)
- Provides Docker access to Linux container APIs
- Execution drivers concept:
- Enable docker to leverage many OS containers as the exec environment
- OpenVZ, system-nspawn, libvirt-lxc, libvirt-sandbox, qemu/kvm, BSD Jails, Solaris Zones, and chroot



DOCKER

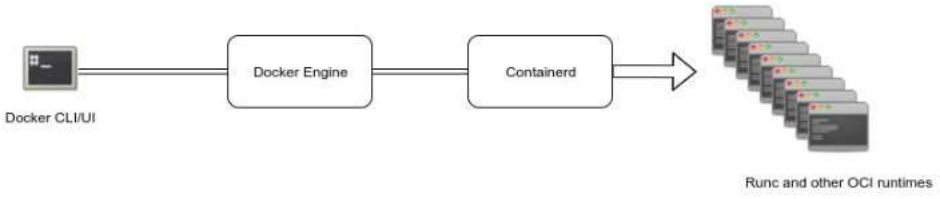
- Docker daemon "dockerd"
  - Provides docker services to Linux
- Docker 1.11+
- Open Container Initiative
- June 2015: Industry standard for container runtimes and formats
- Ensure containers are portable among different execution environments (engines)



Docker Client-Server Architecture

- Credit: https://hackernoon.com/docker-containerd-standalone-runtimes-heres-what-you-should-know-b834ef155426

DOCKER - 2

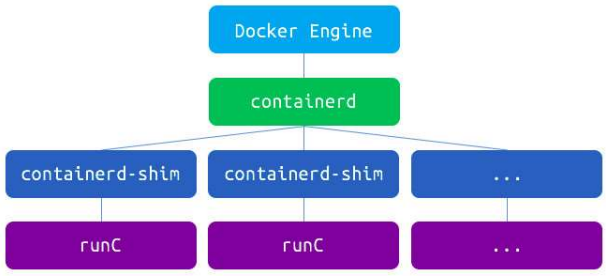


Containerd Integration Architecture

- 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

DOCKER - 3

- Docker architecture:



- 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

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

KEY ORCHESTRATION FEATURES

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

CONTAINER ORCHESTRATION FRAMEWORKS - 2

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

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

TUTORIAL #7

DOCKER, CGROUPS, RESOURCE ISOLATION

November 16, 2020

TCSS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L13.71

DOCKER CLI

- Docker CLI → Docker Enginer (dockerd) → containerd → runc
- Docker installation
- Docker file
- Docker run
- Docker ps
- Docker exec -it
- Docker stop

November 16, 2020	TCSS562: Software Engineering for Cloud Computing [Fall 2020] School of Engineering and Technology, University of Washington - Tacoma	L13.72

```
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
```

TUTORIAL 7

- Linux performance benchmarks
- stress-ng
- 100s of CPU, memory, disk, network stress tests
- Sysbench
- Used in tutorial for memory stress test

QUESTIONS

November 16, 2020

TCSS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L13.75

QUESTIONS

November 16, 2020

TCSS562: Software Engineering for Cloud Computing [Fall 2020]
School of Engineering and Technology, University of Washington - Tacoma

L13.76

TCSS 562

OFFICE HOURS

PLEASE SAY HELLO

TCSS 562
OFFICE HOURS

HAVE STEPPED OUT

WILL RETURN SHORTLY

AREAS OF THE CLOUD

- **Area:** Serverless Computing
  - Function-as-a-Service
  - Container-as-a-Service
- Infrastructure-as-a-Service Cloud
  - Virtual Machines
  - Containers & container clusters (Kubernetes)
- **Perspective**: cloud provider vs. cloud consumer
- **Applications**: tsunami modeling, bioinformatics, environmental modeling
- **Problem:** driven by the area & perspective
  - Common problems: what is the right abstraction? → observability
  - resource contention, resource heterogeneity, provisioning variation, performance variability (delta between min/max performance)