



# Deployment and Operations for Software Engineers

Instructor slides



# Deployment and Operations for Software Engineers

Chapter 1 Virtualization -1

# Overview

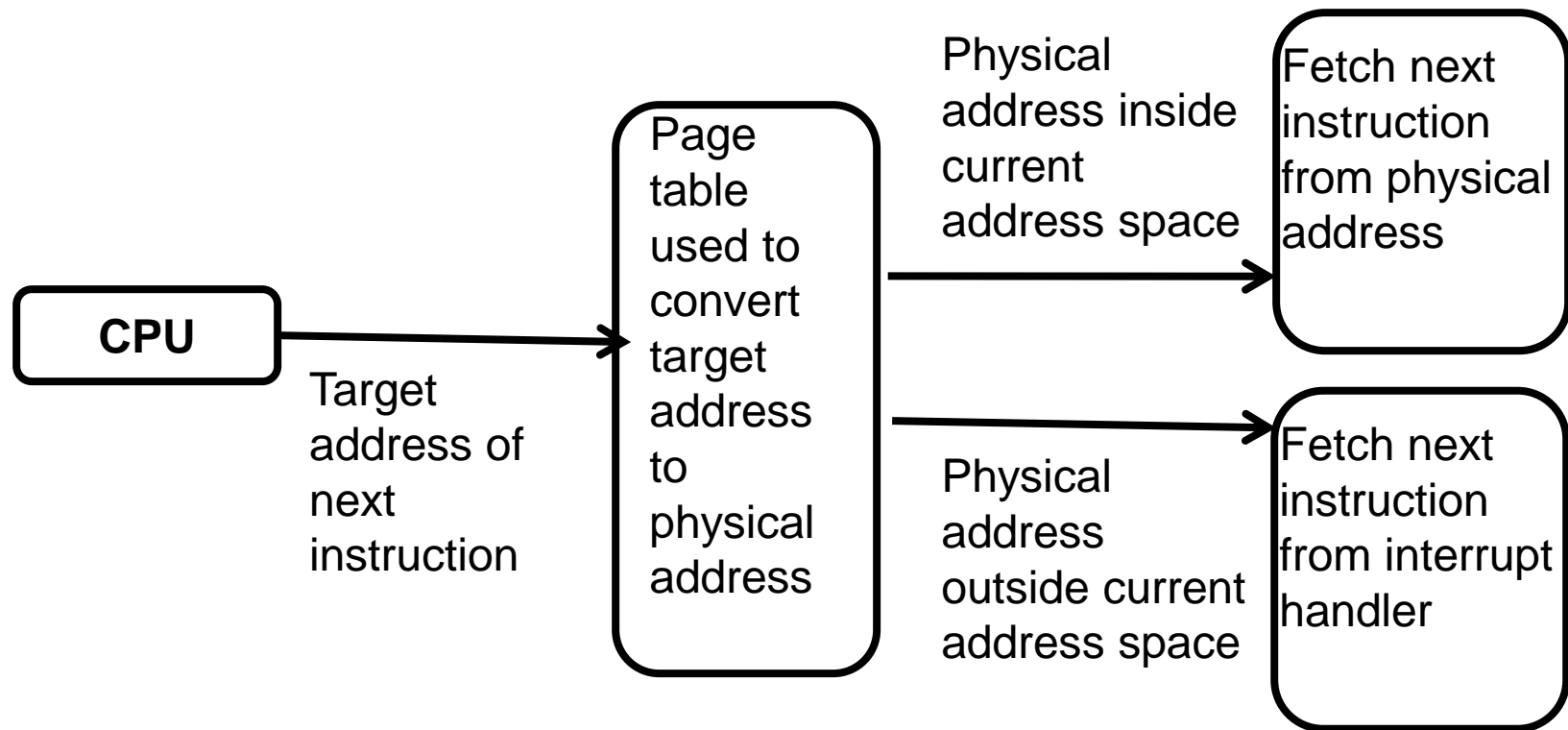
---

- **Virtual machine hardware**
- Virtual machine operating system
- Virtual machine images
- Virtual machine and networks

# What is a Virtual Machine?

- 
- From the perspective of software running on the virtual machine, a virtual machine is a “bare metal” computer. I.e. there is no difference
  - Making a virtual machine look like a bare metal computer is accomplished through a combination of
    - Hardware – primarily address translation
    - Software – a specialized operating system called a “hypervisor”.
-

# Virtual Memory address translation

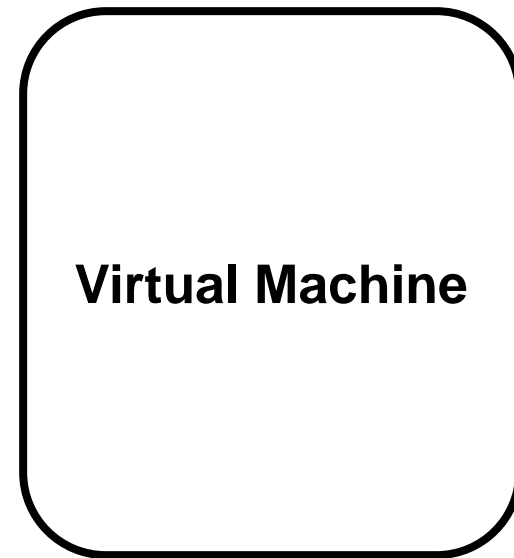


- Hardware enables trapping instructions that are outside of current address space.

# Virtual Machine

---

- Computer with bare hardware
- Instruction set is the same as the host computer
- Address space is guaranteed private from other virtual machines (through the addressing mechanism)
- Available memory may be less than that in the host machine
- Processor is shared across all virtual machines on a single host machine.

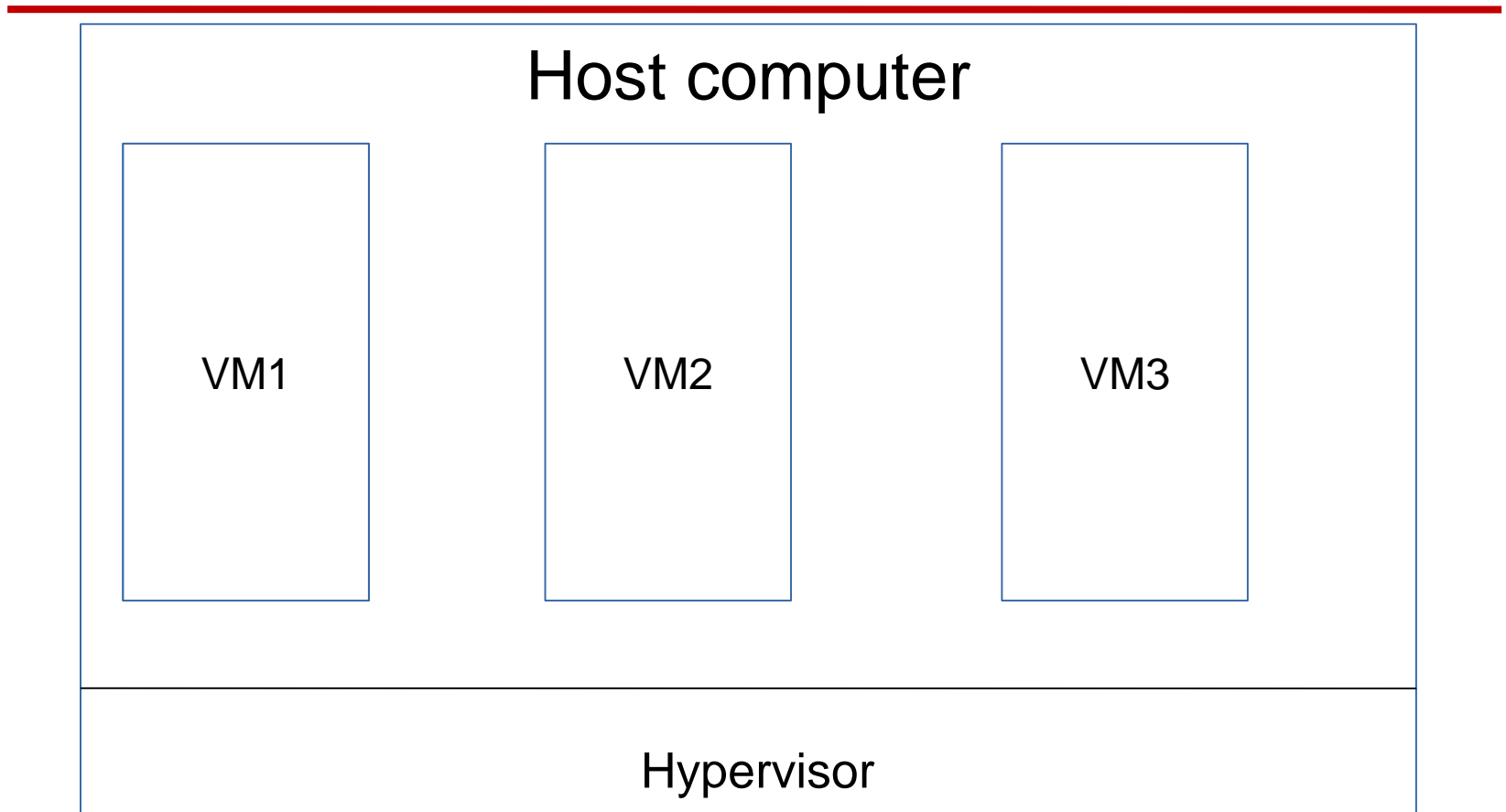


# Overview

---

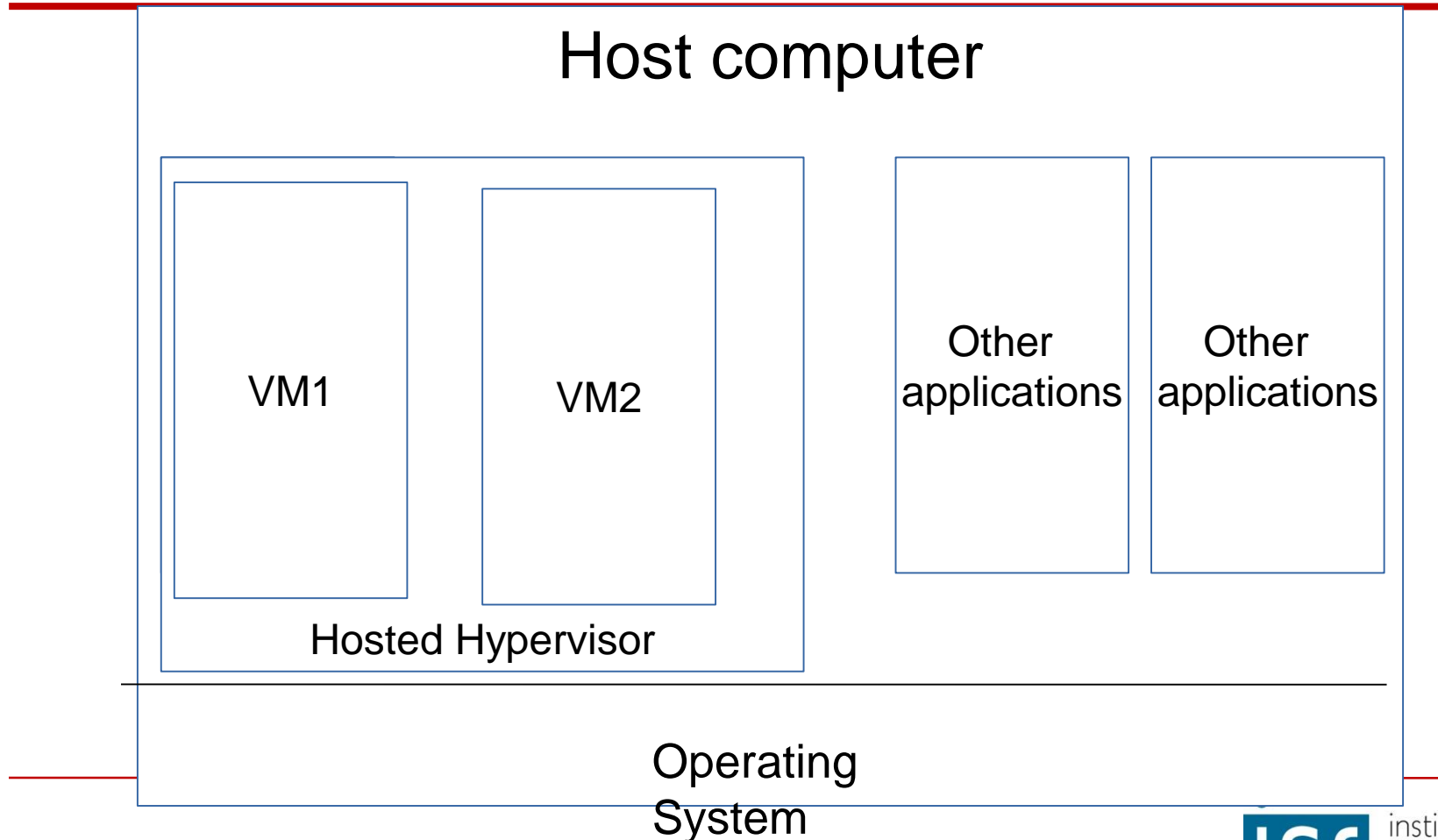
- Virtual machine hardware
- **Virtual machine operating system**
- Virtual machine images
- Virtual machine and networks

# Type 1 Hypervisor on Bare Metal





# Hosted Hypervisor (Type 2)



# File System

---

- Host Computer manages file system for VMs
- Each VM thinks it has its own disk
- File system is an arrangement on that disk

# Network

---

- Each VM has access to the network managed by the Host Computer
- Outgoing messages are tagged with a VM identifier (we discuss IP addresses in Networking section)
- Incoming messages are routed by Hypervisor to correct VM

# Overview

---

- Virtual machine hardware
- Virtual machine operating system
- **Virtual machine images**
- Virtual machine and networks

# Virtual machine images

---

- Bare (virtual) hardware may be all that is necessary for some uses. E.g. operating system revisions.
  - For other uses it is useful to have an operating system and possibly some applications. Application licensing is, typically, by virtual machine.
  - A virtual machine image is set of bits that reside on a disk file and are loaded when the VM is created.
-

# Where does a VM image come from?

---

- A VM image is created by writing the contents of an existing VM to a disk file.
- An initial executable image is created through the build process
- Then software is loaded to create a desired image

# What images exist?

---

- Images can come from
    - A library
    - Something created by an application previously.
  - Sample images might be LAMP – Linux, Apache Server, MySQL, PHP or a version of Ubuntu
  - Why would an application create an image?
    - For deployment to a production environment
    - For restart in case of error
    - As a base to build from when creating a new version.
-

# Overview

---

- Virtual machine hardware
- Virtual machine operating system
- Virtual machine images
- **Virtual machine and networks**



# Loading of VM images

---

- OS executable is ~ 1GB(yte)
- Fast network is ~ 1Gb(it) rated
- Since there are 8 bits per byte, transferring an OS should take 8 seconds.
- But a 1Gb rated network is ~35Mb in practice
- This means loading an OS is >30 seconds
- Additional software adds more time.
- Loading a VM image across a network takes minutes

\*<http://www.tomshardware.com/reviews/gigabit-ethernet-bandwidth,2321-3.html>

---

# Summary

- 
- A virtual machine can be treated as a bare metal computer from a user's perspective
  - A virtual machine is managed by a specialized operating system (hypervisor) and supporting hardware.
  - Loading a VM image across a network takes minutes
-



# Deployment and Operations for Software Engineers

Chapter 1 Virtualization - 2

# Overview

---

- **Container definition**
- Layers and deployment

# Goal

- 
- Want to package executable such that it
    - Is isolated from other executables
    - Is lightweight in terms of loading and transferring
  - Containers provide such a packaging
  - Containers require run time engine which provides many services

# Containers

---



## Process

- Isolate address space
- No isolation for files or networks
- Lightweight



## Container

- Isolate address space
- isolate files and networks
- Lightweight



## Virtual Machine

- Isolate address space
- isolate files and networks
- Heavyweight

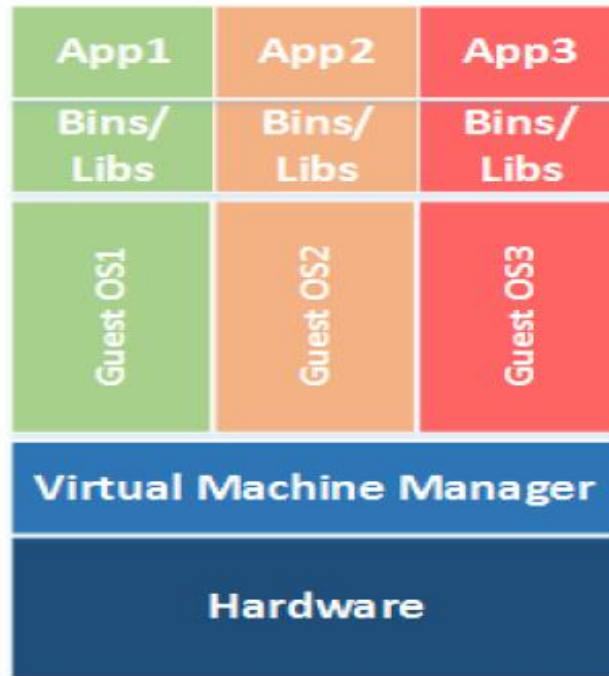
# Container images

---

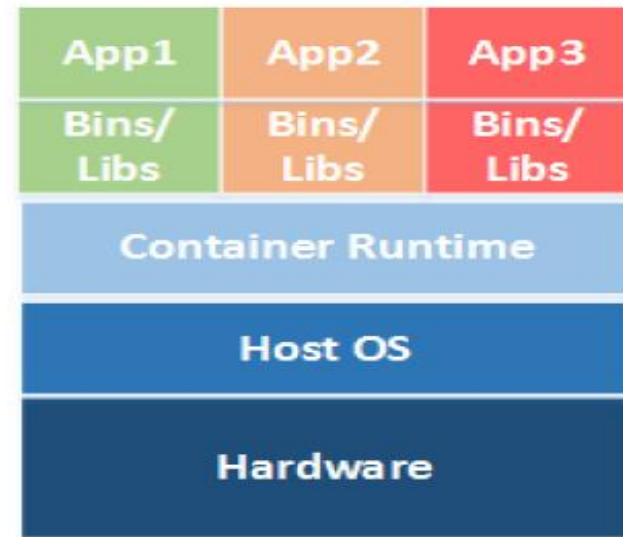
- As with VMs,
  - A container image is a set of bits on a disk.
  - A container is an executing entity
- Terminology is not always followed.
  - “Container” is used to refer both to images and executing entities

# VMs and Containers

VMs



Containers on Bare Metal





# VMs vs containers

- 
- VMs are virtualized hardware.
    - Each VM can run its own OS.
    - All OSs and apps use same instruction set
    - VM manager is hypervisor
    - Each VM has its own IP address
  - Containers are virtualized operating system.
    - Each container can use its own binaries and libraries
    - All apps use same OS
    - Container manager is, for example, “Docker Engine”
    - Each container has its own IP address
    - Containers have limitations in network and file capabilities

# Persistence

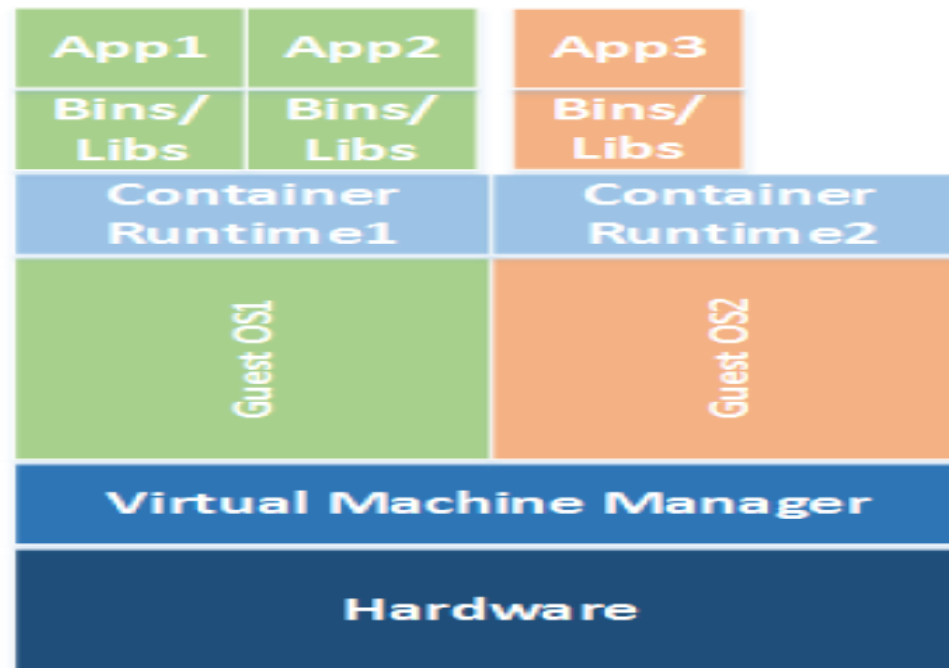
- 
- VMs, depending on the Virtual Machine Manager, will generally persist until user takes explicit action or VM fails.
  - Some container runtimes, e.g. Lambda, may not persist their containers.
    - Container will not be deleted in the middle of processing a message
    - State should not be stored in container if it will be deleted
    - IP addresses should not be shared if container will be deleted
-

# Portability

- 
- Containers are portable across platforms as long as the container runtime supports the same format.
  - The Open Container Initiative has standardized the format of containers. Consequently, runtimes from one vendor will support containers constructed from another vendor
  - This allows a container to be moved from one environment to another.
    - Same software is moved
    - Network connections must be established for each environment
-

# VMs and containers can be combined

Containers in VMs



# Overview

---

- Container definition
- **Layers and deployment**

# Layers

- 
- A container image is structured in terms of “layers”.
  - Process for building image
    - Start with base image
    - Load software desired
    - Commit base image+software to form new image
    - New image can then be base for more software
  - Image is what is transferred

# Exploiting layers

- 
- When an image is updated, only update new layers
  - Unchanged layers do not need to be updated
  - Consequently, less software is transferred and an update is faster.

# Speeding up loading across a network

---

- Recall that loading a VM across a network takes minutes
- Suppose your cloud provider kept your container images local to the physical machine where they are run.
- Then when you invoke the container, it will load in milliseconds, not minutes.



# Trade offs - 1

- 
- Virtual machine gives you all the freedom you have with bare metal
    - Choice of operating system
    - Total control over networking arrangement and file structures
    - Time consuming to load over network

# Trade offs - 2

---

- Container is constrained in terms of operating systems available
  - Linux, Windows, and OSX
  - Provides limited networking options
  - Provides limited file structuring options
  - Fast to load over network

# Summary

---

- A container is a lightweight virtual machine that provides address space, network, file isolation
- Docker allows building images in layers and deployment of a new version just requires deploying layers that have changed.

# **Deployment and Operations for Software Engineers**

## **Chapter 2 Networking - 1**

# Overview

- **IP addresses**
- Domain Name System/Time to Live

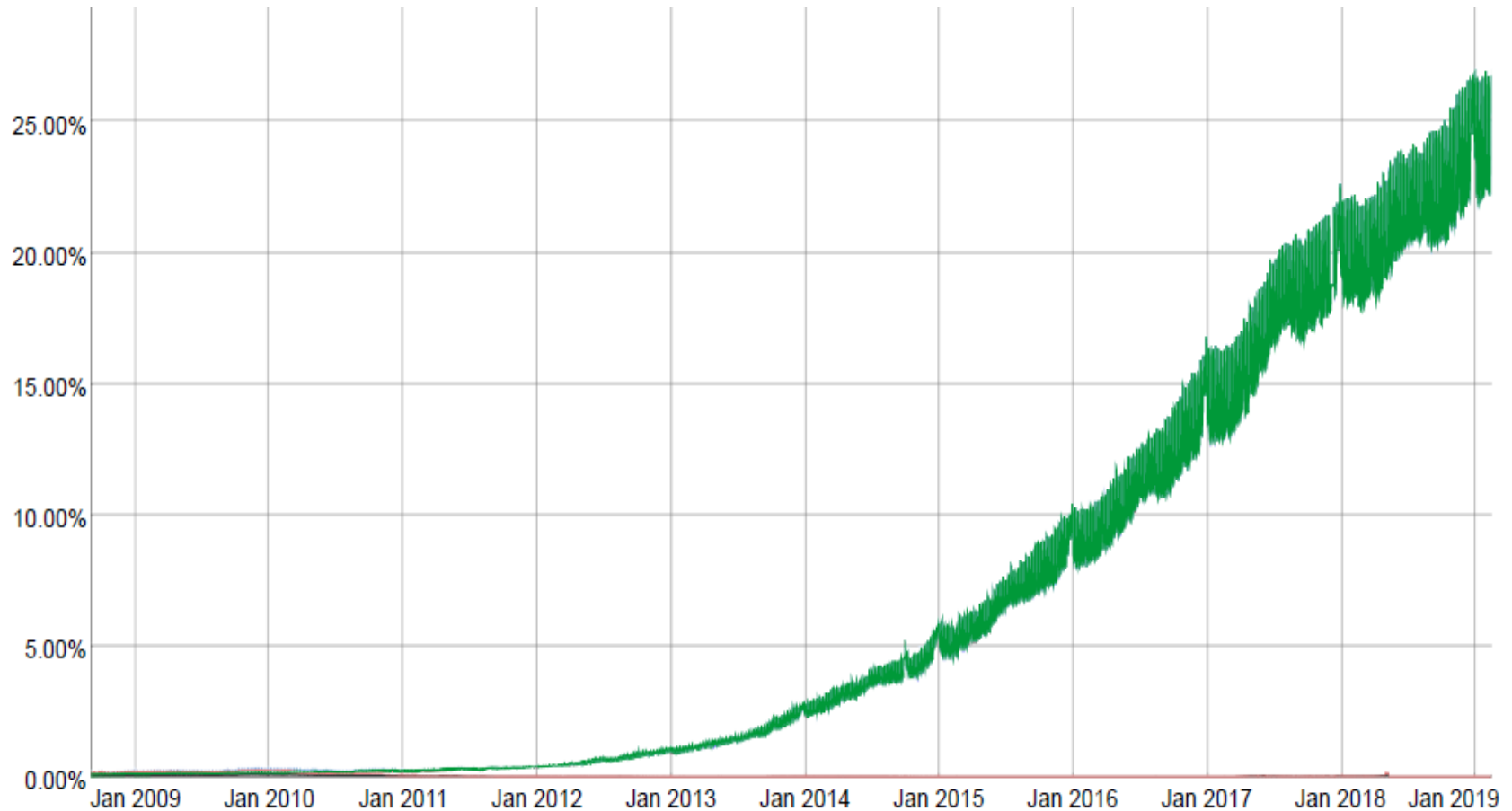
# IPv4 and IPv6

- An IP (Internet Protocol) address is a numerical label that identifies a “device” on the internet.
- IPv4 is 32 bits long and gives a four number sequence - xxx.xxx.xxx.xxx
- 32 bits is insufficient and so IPv6 was created in 1995 and it has 128 bits.

# Adoption

- For legacy reasons, IPv6 has had a very slow adoption. IPv4 numbers have been exhausted. This is causing more conversion to IPv6. June 8, 2011 was designated as world IPv6 day where top websites and internet providers provided a 24 hour test of IPv6 infrastructure. This test was successful.

## Percentage of Users that access Google over IPv6.





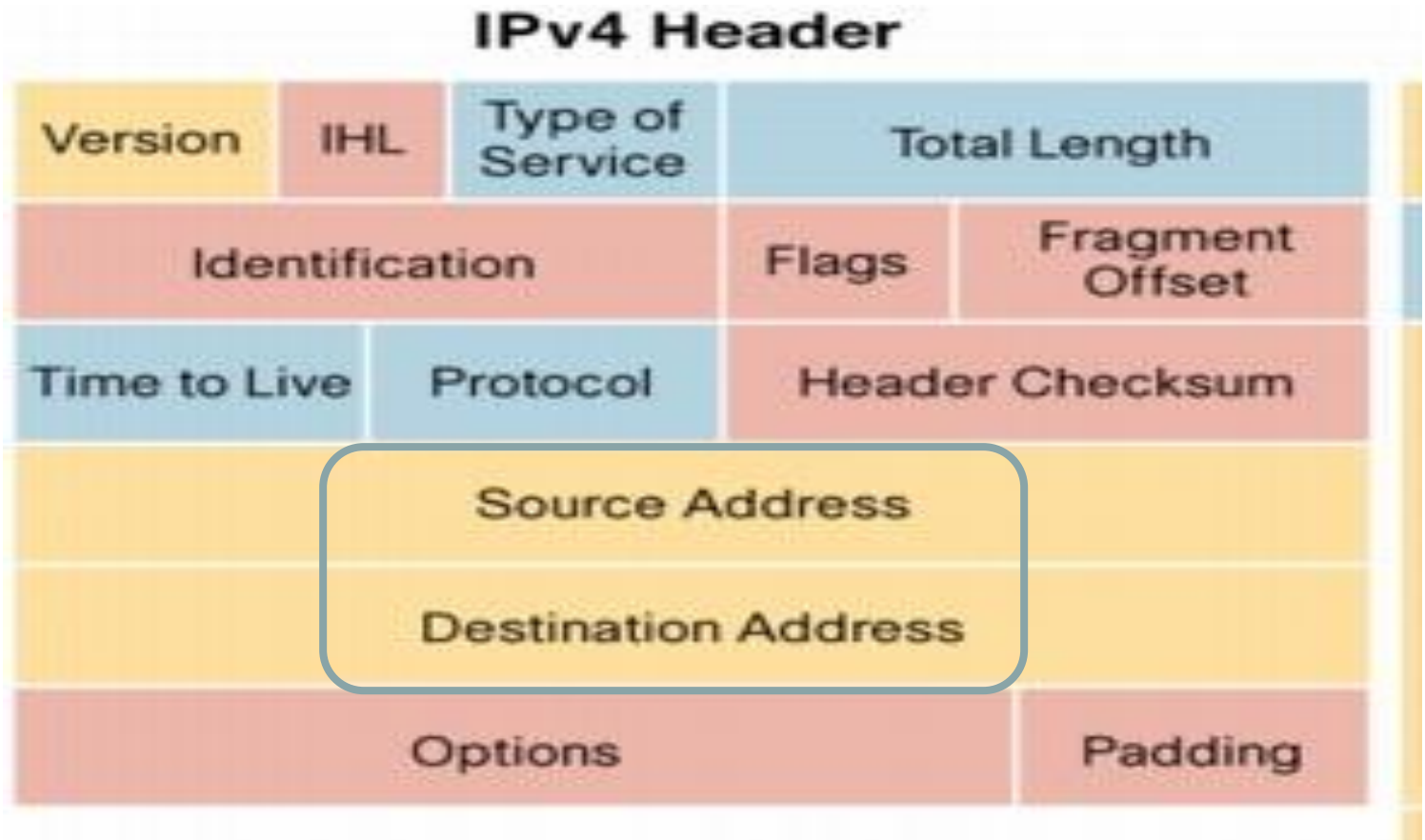
# Assigning IP addresses

- Every “device” on the internet includes virtual machines in a cloud or in an application such as VirtualBox.
  - Every VM or container gets an IP address when it is created. This IP address can be
    - Private and not seen outside of the cloud or the application..
    - Public and directly addressable from outside of the cloud.
- Conventional IP addresses
  - 127.0.0.1 – local host – current physical machine.
  - 192.xx.xx.xx – private network.

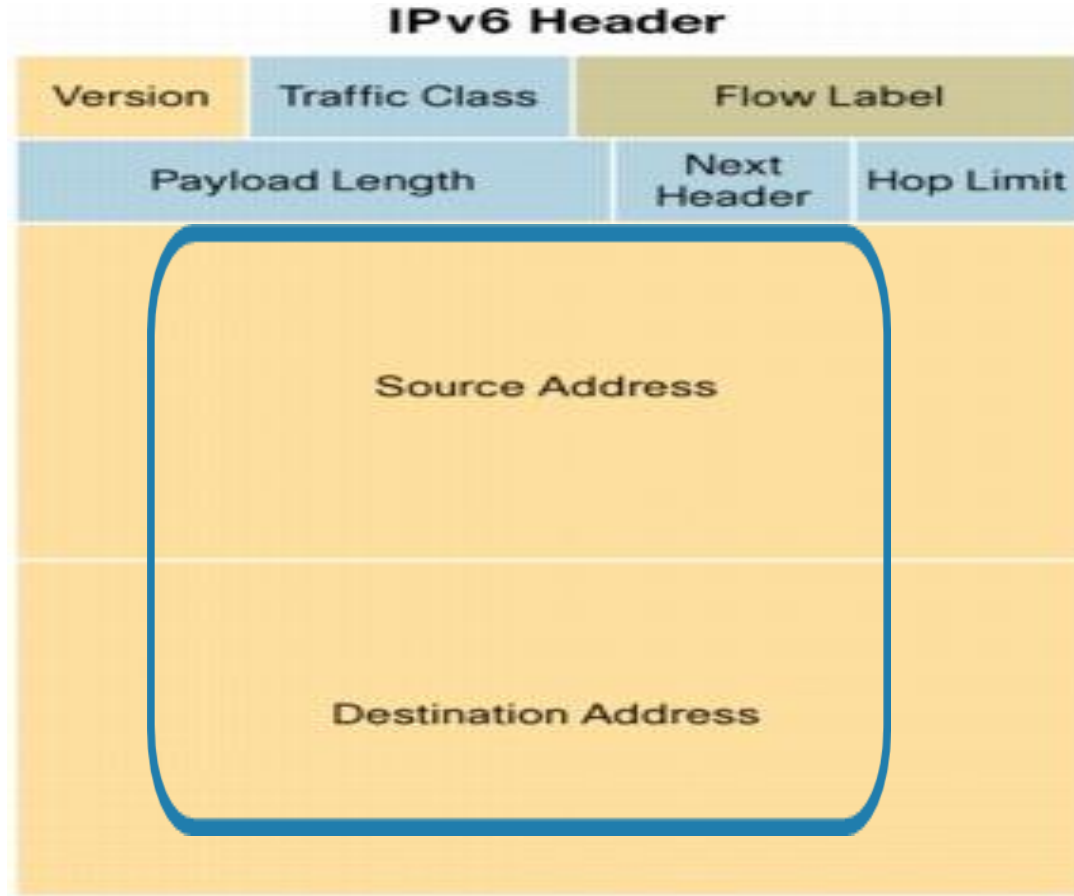
# IP message format

- An IP message has a header and a payload. The header includes
  - IP address of the source
  - IP address of the destination

# Internet Protocol packet structure (V4)



# Internet Protocol packet structure (V6)



# Private

- Private IP addresses:
  - When  $IP_A$  sends message to  $IP_B$ , i.e.,  $IP_A + \text{payload} \rightarrow IP_B$ , a gateway can make it look like the message comes from the gateway. i.e.  $IP_{\text{gateway}} + \text{payload} \rightarrow IP_B$
  - In this case the gateway must maintain a table so that it can manage the response from  $IP_B$

# Public IP addresses

- The VM manager is given a range of IP addresses that it can assign to VM instances.
- An assignment only lasts as long as the instance does, then it can be re-assigned.
- Messages from the instances come from the assigned IP address and recipient can respond directly to instance.
- The VM manager is, typically, your ISP

# Merging two local networks

- Bridge – appears externally to be one network. Each machine's IP address is available from outside of the bridged network. Typically used on bare metal hypervisors and container runtime engines.
- NAT (Network Address Translation) – externally there exists only one IP address. The NAT modifies the source on outgoing messages and the destination on incoming messages to route the messages. Typically used on hosted hypervisors.

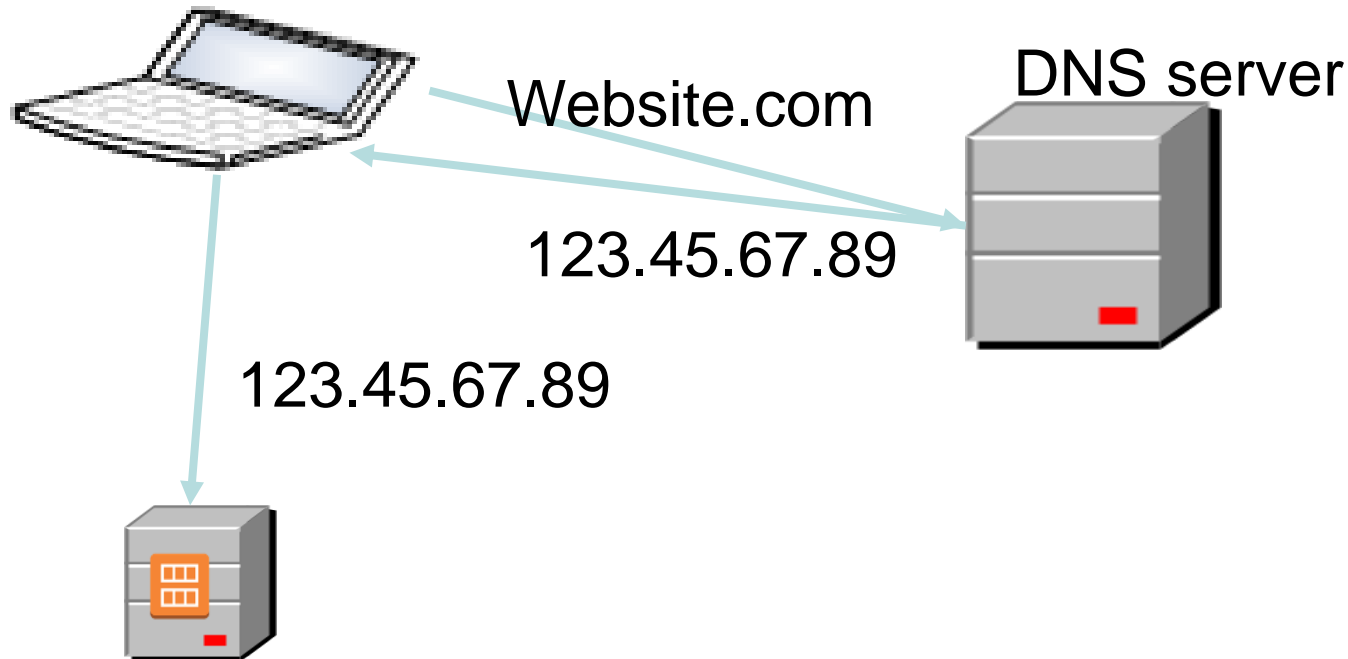
# Overview

- IP addresses
- **Domain Name System/Time to Live**



# Domain Name System(DNS)

- Client sends URL to DNS
- DNS takes as input a URL and returns an IP address
- Client uses IP address to send message to a site



# Complications

- In reality, messages being transmitted from one computer to another is more complicated.
- The picture showed a single DNS server.
  - There are multiple DNS servers
  - There is a hierarchy of DNS servers.
- The picture showed a single line from client to server.
  - There is a network for routers to transmit messages
  - Shares load

# DNS Hierarchy

- Consider URL `www.mse.isr.cmu.edu`
  - If one server held all DNS -> IP mappings, it would both get overloaded and hold over 200 million mappings.
- DNS is arranged as a hierarchy.

# Finding `www.mse.isr.cmu.edu`

- Begin with “root server”. There are ~13 root servers with known IP addresses. These are built into the router.  
(<https://www.iana.org/domains/root/servers>)
- Access root server to get IP address of the .edu DNS
- The .edu DNS has the IP of the .cmu.edu DNS and so forth.
- Eventually you get to a DNS server that is under local control
- This allows MSE to change the IP of the various local DNSs without changing anything up the hierarchy.

# Time to Live

- Clients do not access a DNS server for every request. It would generate too much internet traffic.
- Associated with each DNS entry is a Time To Live (TTL).
- This is also called a “Refresh Interval” in the DNS resource record called Start of Authority (SOA).
- The client or the ISP caches the IP addresses associated with DNS entries and these entries are valid for the TTL.
- This is distinct from the TTL listed in the IPV4 packet header.

# TTL manipulation

- The value of the TTL depends where in the hierarchy the record exists.
- High in the hierarchy (authorative servers), TTLs are set to 24 hours since the likelihood of there being a change is low.
- Records under local control can have their TTLs set low (~minutes)
- We will see applications of small TTL settings when we discuss switching between different development environments.

# Client (browser) perspective on DNS, URL, and TTL

- Client gets ip addresses of edu from root
- TTL is 24 hours
- Client gets ip addresses of cmu from one of edu
- TTL is 24 hours
- Client gets ip addresses of mse from one of cmu
- TTL is 12 hours
- Client gets ip address of www from mse
- TTL is 5 minutes

# Summary

- IP addresses tells network how to route a message to get it to a particular computer.
- Merging two local networks can be done with a bridge or with a NAT
- Domain Name System translates URLs into IP addresses



# **Deployment and Operations for Software Engineers**

## Chapter 2 Networking -2

# Overview

- **Ports**
- Firewalls and subnets

# Ports

- Hotel switchboard
  - Hotel has a phone number that rings at a switchboard
  - Switchboard operator then connects incoming call to a phone in a particular room



# Computer ports

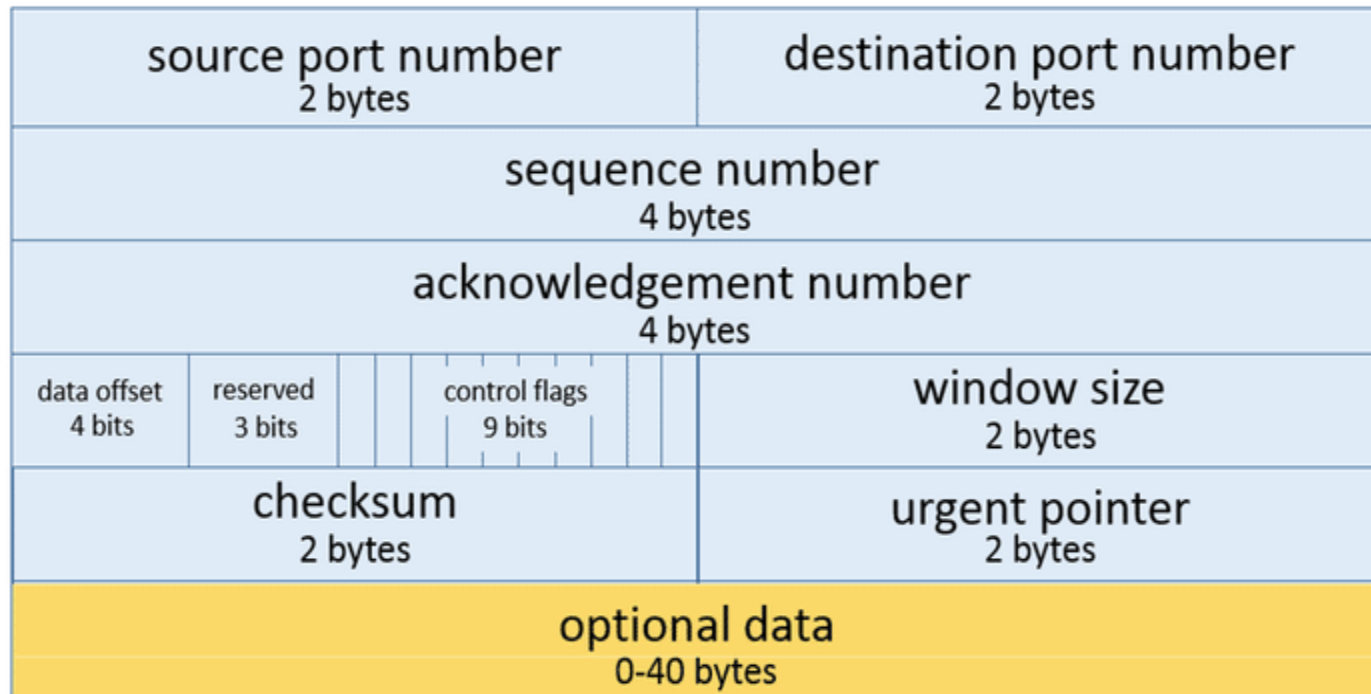
- Similar to hotel switchboard
- Application listens on a particular port
- Message comes to the Virtual Machine and is placed on a particular port.
- Application gets that message
- Known ports for standard applications. E.g. 80 for http

# Ports and IP addresses

- IP addresses say where on the internet a computer (virtual or real) lives.
  - They are machine specific and controlled by the Internet Protocol (IP)
- A port describes how to route a message to a particular application.
  - They are application specific and specified in a message by the Transmission Control Protocol (TCP)
- To send a message to a particular application on a particular machine both IP addresses and ports are used (TCP/IP)

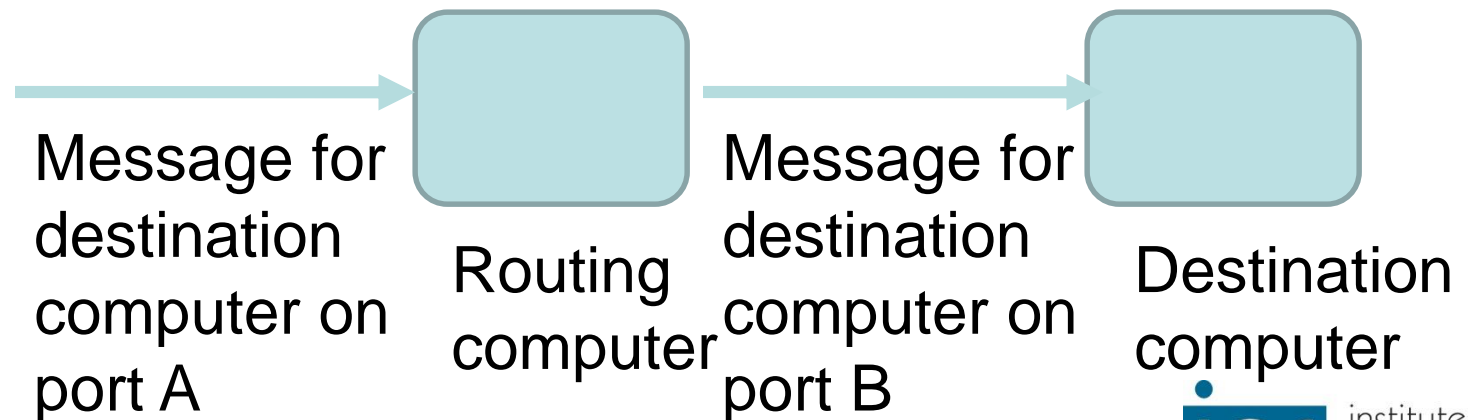
# Transmission Control Protocol (TCP)

## Transmission Control Protocol (TCP) Header 20-60 bytes



# Port forwarding

- Two applications listening on the same port will mean a message destined for one application may be sent to another.
- Port forwarding tells a routing computer that a message specified for one port should actually be delivered to another.



# Applications and ports

- Applications listen on a particular port but ...  
they may need to listen on a different port than originally specified because of contention.
- Applications may be parameterized to specify which port they listen on.
- Observation – applications specify which port they listen on but not which ip address their host has.



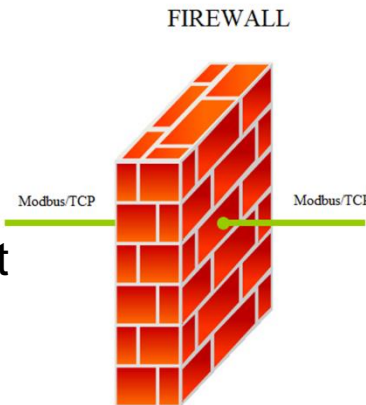
# Ports, bridged networks, VMs and Containers

- Ports are managed by the operating system. It decodes a message and routes it to the correct application.
- Each VM has its own operating system so it is possible to have two VMs on a bridged network each using the same port number.
- Containers share an operating system so two applications on a container bridged network cannot use the same port number.

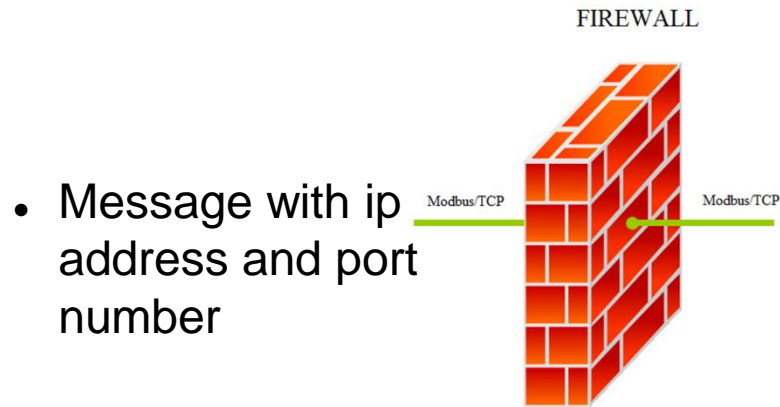
# Overview

- Ports
- **Firewalls and subnets**

# Firewall– Restrict access

- Message with ip address and port number
- 
- The diagram shows a 3D perspective of a brick wall, colored in shades of orange and red. Above the wall, the word "FIREWALL" is written in a simple, black, sans-serif font. A horizontal green line passes through the center of the wall, representing a message. On both the left and right sides of the wall, the text "Modbus/TCP" is written in a small, black, sans-serif font, indicating the protocol of the message.
- Firewalls are configured by specifying rules
    - allow message based on ip address of recipient
    - route allowed message based on port number
      - messages specifying port 80 are sent to http server
      - messages specifying port 443 are sent to https server

# Firewall– Detection



- One attack type is a “port scan”. Look for open ports
- Firewall can record requests from particular ip addresses with different ports and place that ip address on a black list
- Firewalls can also be set to act as “throttles” to limit traffic in case of a denial of service attack.
- Typically, there are multiple firewalls inside a network

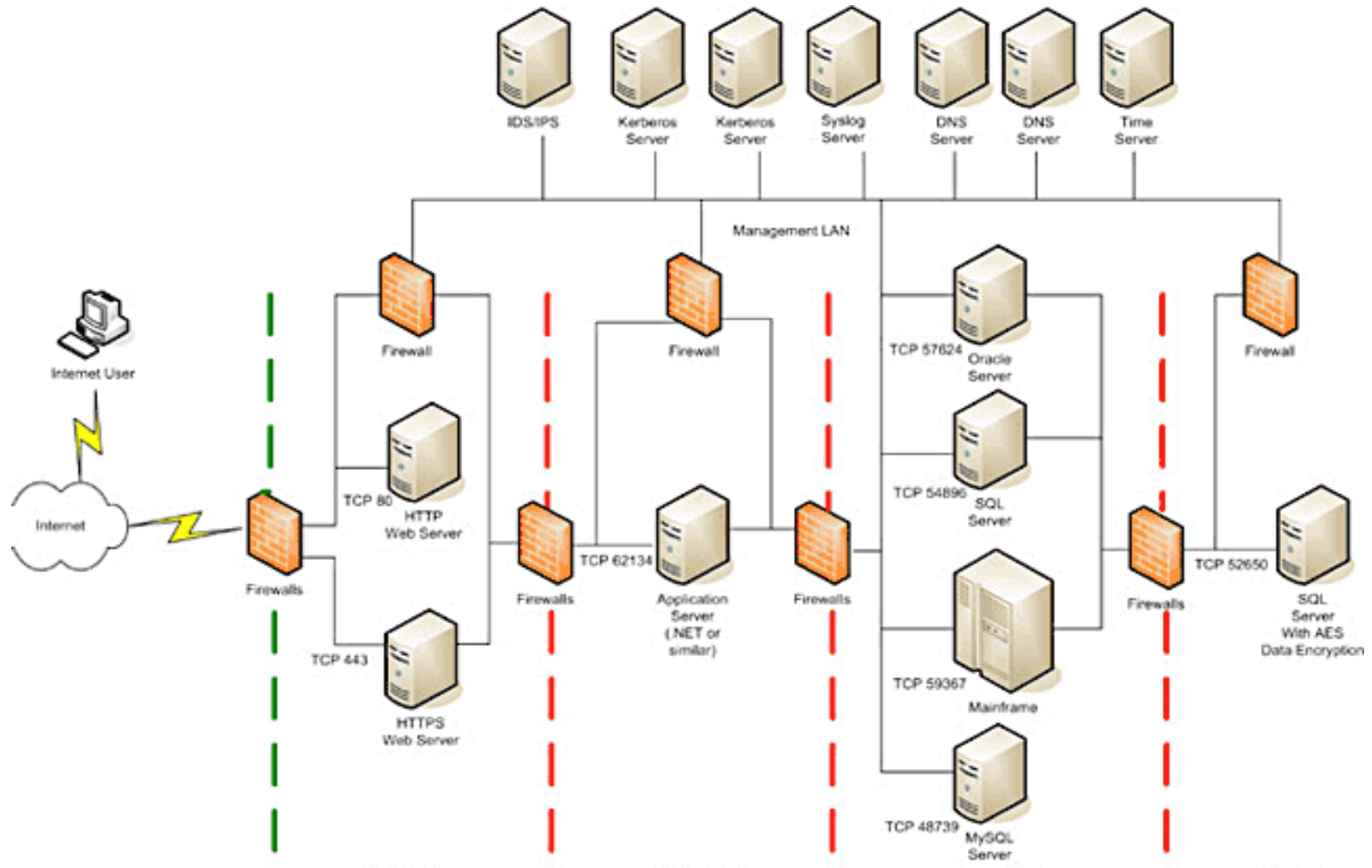
# Demilitarized Zone (DMZ)

- Typically an organization distinguishes between information publically available and that maintained within the organization.
- An intranet is a private network used by an organization
- A DMZ is an area guarded by a firewall that allows external access although not external modification
- A separate firewall allows access from the DMZ into the intranet.
- Web servers and Virtual Public Network servers are located in the DMZ

# Virtual Private Networks

- Imposes private network on top of public network
- Uses “tunneling” server located in DMZ
- You log on to tunneling server, it verifies your credentials and allows you to access the intranet.
- Your messages to the VPN are encrypted and passed through the firewall into the intranet.

# A typical Network architecture



# Networks have subnets

- A subnet is a logical portion of a network.
- It is used to divide the network into groups.
- Allows better management of traffic for network administrator
- Each group
  - Has its own access rules
  - Can be protected by a firewall



# Summary

- Ports are means for routing messages between applications.
- Firewalls restrict access both from outside and internally within a network



# Deployment and Operations for Software Engineers

Chapter 3 The Cloud - 1

# Overview

---

- **Structure**
- Failure in the Cloud
- Scaling Service Capacity

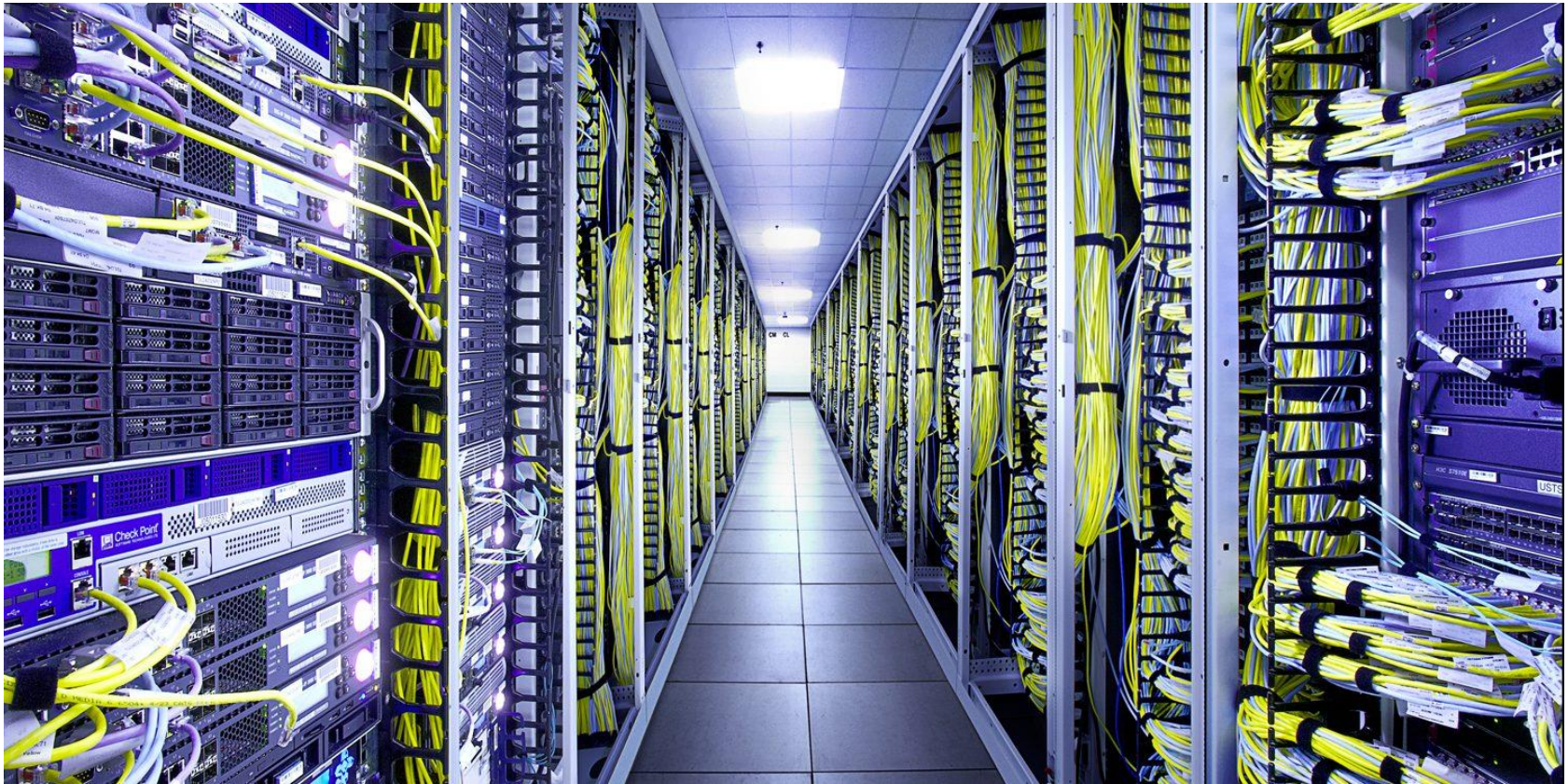
# Data centers in the cloud

---

- A cloud provider (AWS, Google, Microsoft) maintains data centers around the world.
- Each data center has ~100,000 computers.
- Limited by power and cooling considerations.

# Data Center

---



# Organization of data centers

---

- Each data center has independent power supply, independent fire control, independent security, etc
- Data centers are collected into availability zones and availability zones are collected into geographic regions.
- AWS currently has 20 regions each with several availability zones.



# Allocating a virtual machine in AWS - 1

---

- A user wishes to allocate a virtual machine in AWS
  - The user specifies
    - A region
    - Availability zone
    - Image to load into virtual machine
    - ...

# Allocating a virtual machine in AWS - 2

---

- AWS management software
    - Finds a server in that region and availability zone with spare capacity
    - Allocates a virtual machine in that server
    - Assigns IP address to that virtual machine (public)
    - Loads image into that virtual machine
  - VM can then send and receive messages
-



# Overview

---

- Structure
- **Failure in the Cloud**
- Scaling Service Capacity

---

*A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.*

Leslie Lamport

# Failures in the cloud

- 
- Cloud failures large and small
  - The Long Tail
  - Techniques for dealing with the long tail

# Sometimes the whole cloud fails

---

## Selected Cloud Outages - 2018

- Feb 15, Google Cloud Datastore
  - March 2, AWS Eastern region networking problems
  - April 6, Microsoft Office 365. Users locked out of their accounts
  - May 31, AWS Eastern Region hardware failure caused by “power event”
  - June 17, Microsoft Azure. Heatwave in Europe and malfunctioning air conditioning
  - July 17, Google. Load balancer issue affected services globally
  - July 16, Amazon Prime. Not AWS but Amazon sales
  - Sept 5. Microsoft Office 365. Botched update to Azure authentication
-

And sometimes just a part of it fails

...

# A year in the life of a Google datacenter

- 
- Typical first year for a new cluster:
    - ~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
    - ~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
    - ~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
    - ~5 racks go wonky (40-80 machines see 50% packetloss)
    - ~8 network maintenances (4 might cause ~30-minute random connectivity losses)
    - ~12 router reloads (takes out DNS and external vips for a couple minutes)
    - ~3 router failures (have to immediately pull traffic for an hour)
    - ~dozens of minor 30-second blips for dns
    - ~1000 individual machine failures
    - ~thousands of hard drive failures

---

slow disks, bad memory, misconfigured machines, flaky machines, dead horses, ...

# Amazon failure statistics

- 
- In a data center with ~64,000 servers with 2 disks each  
~5 servers and ~17 disks fail every day.

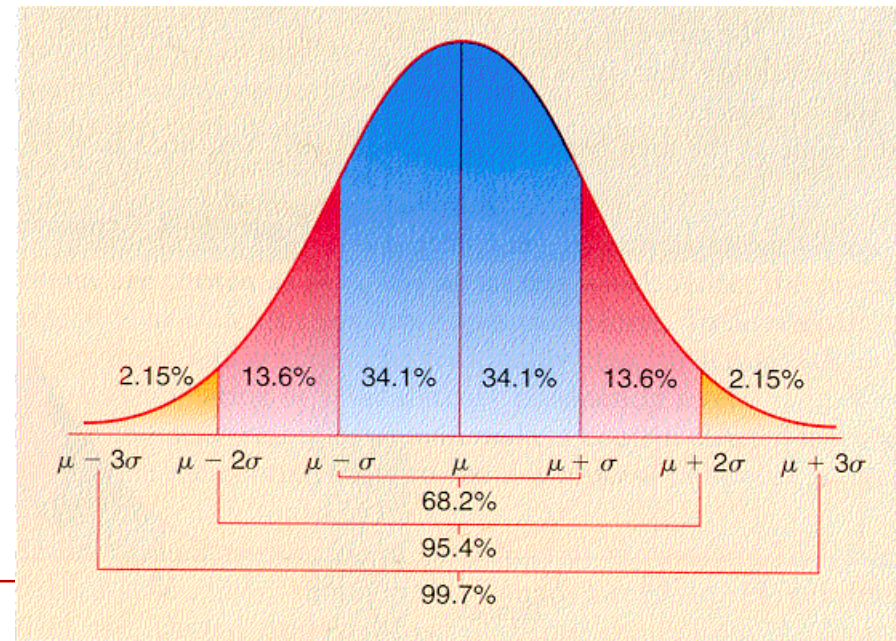
# Failures in the cloud

- 
- Cloud failures large and small
  - The Long Tail
  - Techniques for dealing with the long tail



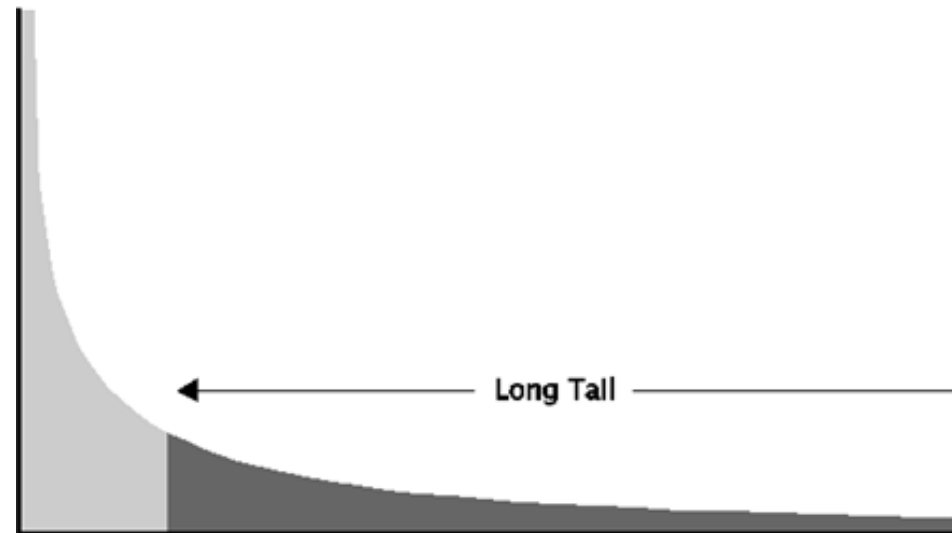
# Short digression into probability

- A distribution describes the probability that any given reading will have a particular value.
- Many phenomenon in nature are “normally distributed”.
- Most values will cluster around the mean with progressively smaller numbers of values going toward the edges.
- In a normal distribution the mean is equal to the median



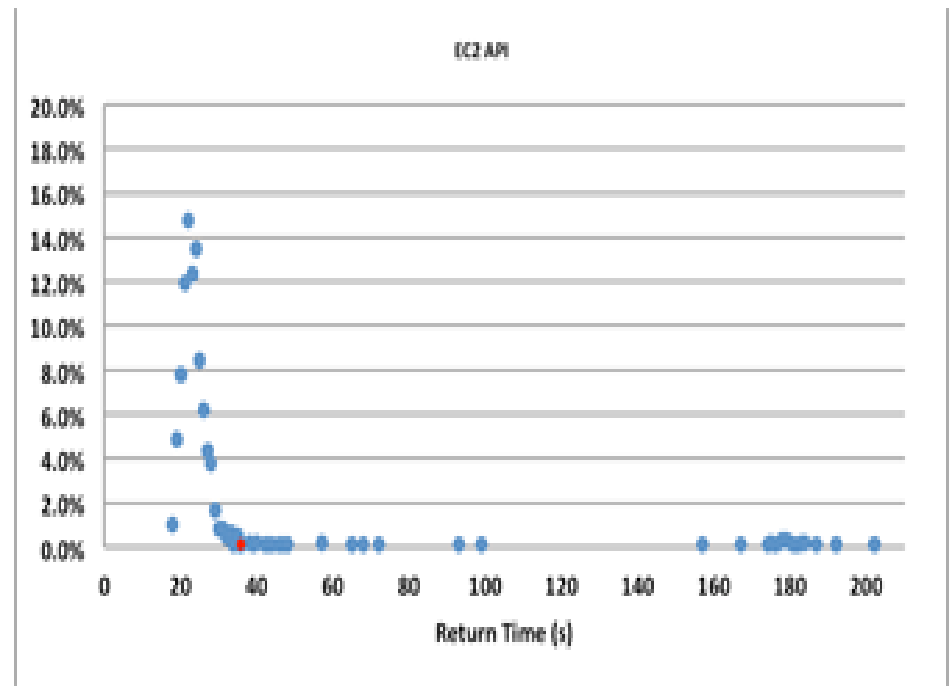
# Long Tail

- 
- In a long tail distribution, there are some values far from the mean.
  - These values are sufficient to influence the mean.
  - The mean and the median are dramatically different in a long tail distribution.



# What does this mean?

- If there is a partial failure of the cloud some activities will take a long time to complete and exhibit a long tail.
- The figure shows distribution of 1000 AWS “launch instance” calls.
- 4.5% of calls were “long tail”



# Failures in the cloud

- 
- Cloud failures large and small
  - The Long Tail
  - Techniques for dealing with the long tail

# What can you do to prevent long tail problems?

- 
- Recognize that failure has occurred
  - Recover from failure
  - Mask failure

# Recognizing failure

---

- In a distributed system, the only communication between distinct computers is through messages.
- A request (message) to another machine goes unanswered.
- The failure is recognized by the requestor making the request asynchronously and setting a timeout.

# Health Check

---

- Health check. Requires one machine to act as a monitor for other machines.
  - Machine sends a message to a monitor periodically saying “I am alive”
  - Ping/echo. Monitor sends a message to the machine being monitored and expects a reply within specified period.

# Recovering from failure

---

- Retry
- Instantiate another instance of failed service
- Graceful degradation
- Set “circuit breaker” to prevent trying to use service again.
- Netflix Open Source Hystrix library provides some useful functions



# Masking failure

- 
- “Hedged” request. Suppose you wish to launch 10 instances. Issue 11 requests. Terminate the request that has not completed when 10 are completed.
  - “Alternative” request. In the above scenario, issue 10 requests. When 8 requests have completed issue 2 more. Cancel the last 2 to respond.
  - Using these techniques reduces the time of the longest of the 1000 launch instance requests from 202 sec to 51 sec.

# Overview

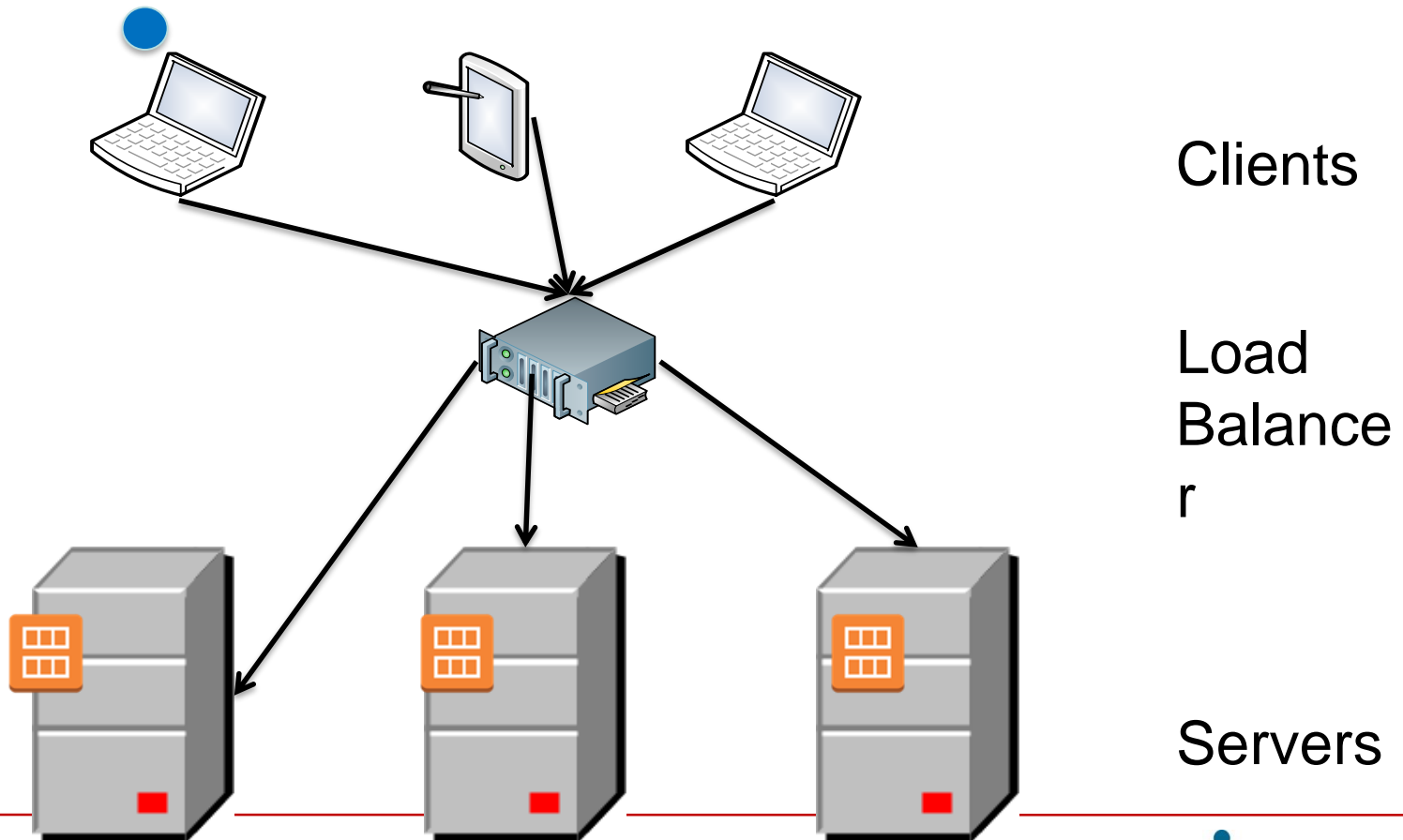
---

- Structure
- Failure in the Cloud
- **Scaling Service Capacity**

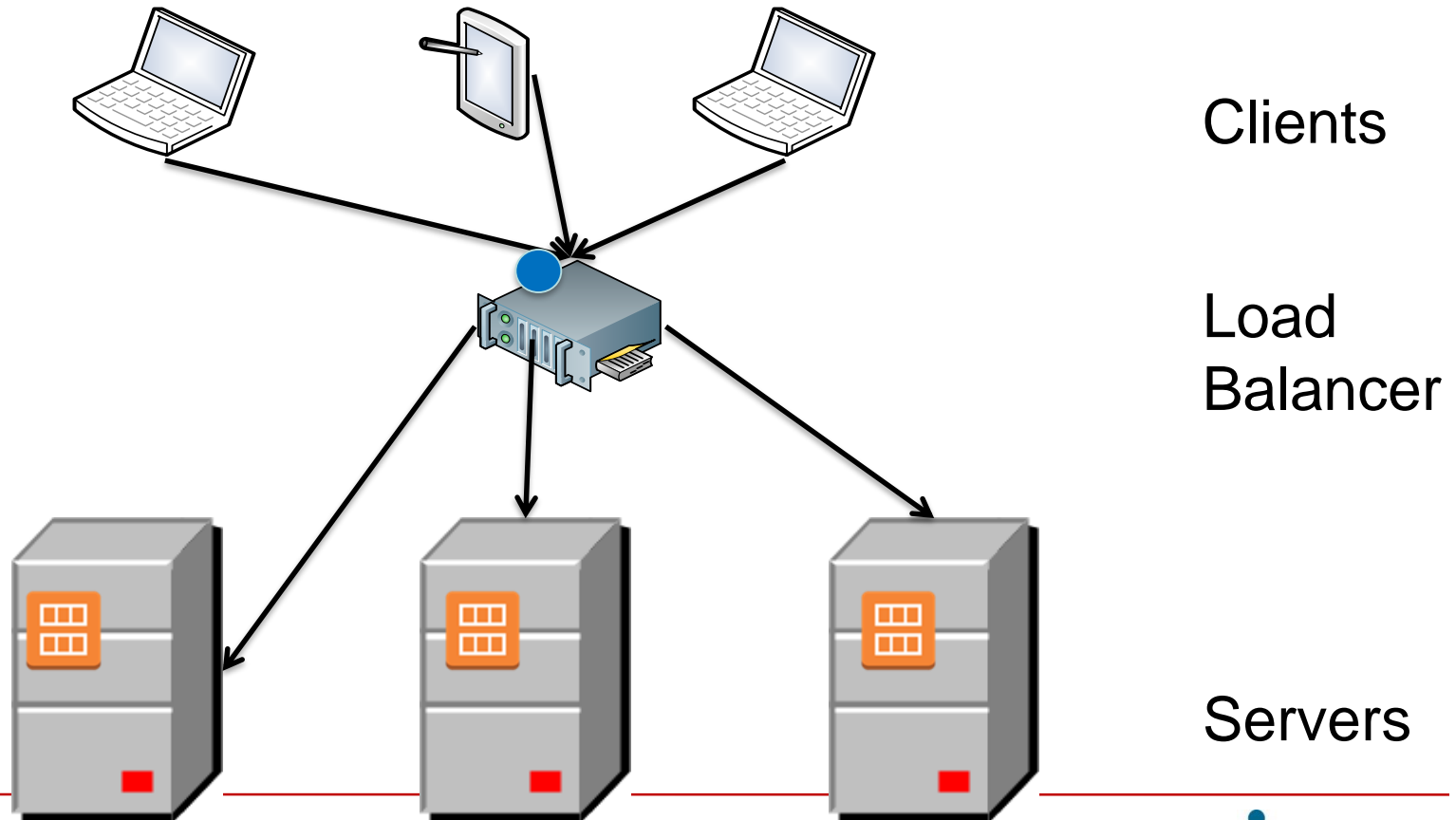
# Load Balancer

- 
- One server may not suffice for all of the requests for a given service.
    - Have multiple servers supplying the same service.
    - Use “load balancer” to distribute requests.
  - Server is registered with load balancer upon initialization
  - Load balancer monitors health of servers and knows which ones are healthy.
  - Load balancer IP is returned from DNS server when client requests URL of service.
-

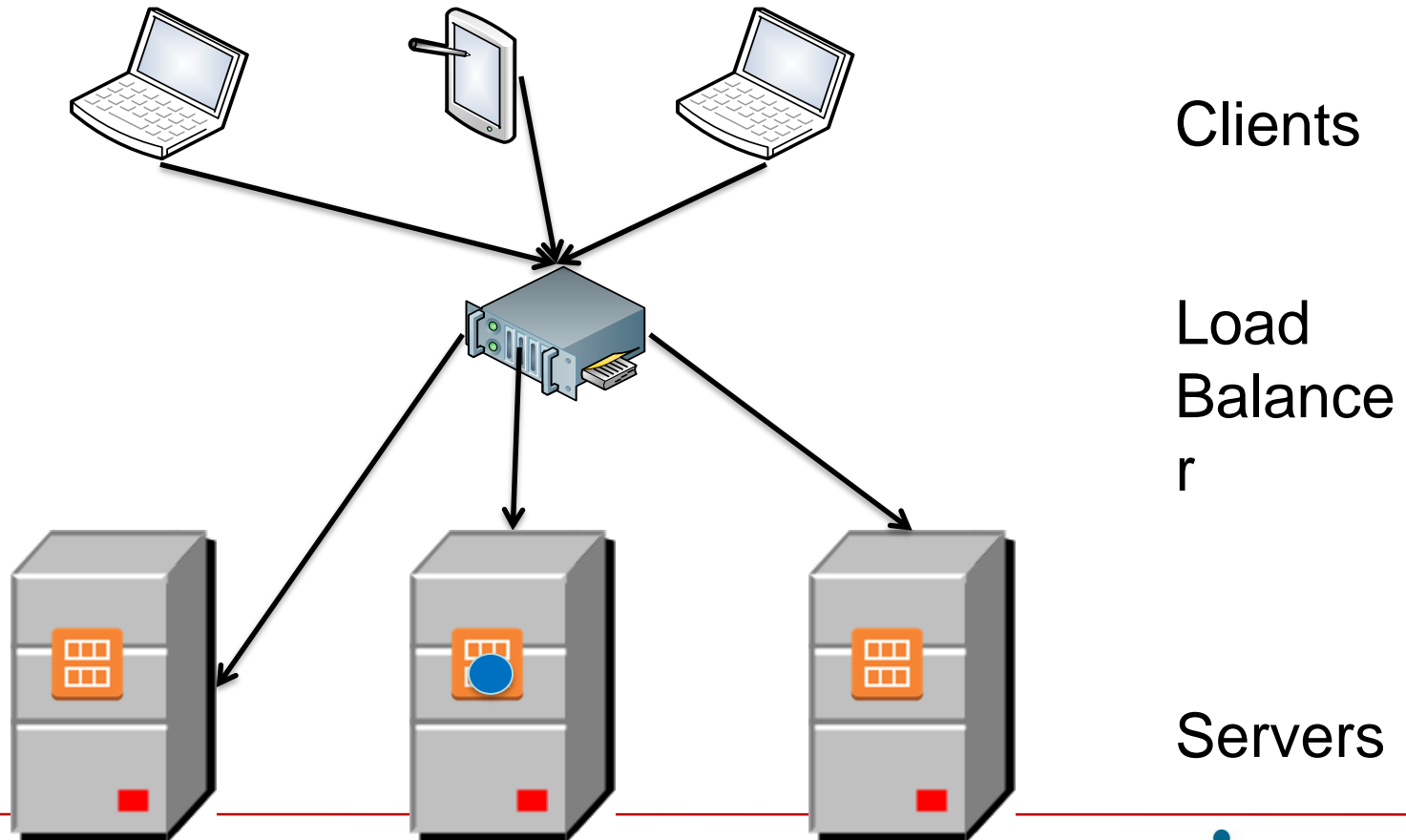
# Message sequence – client makes a request



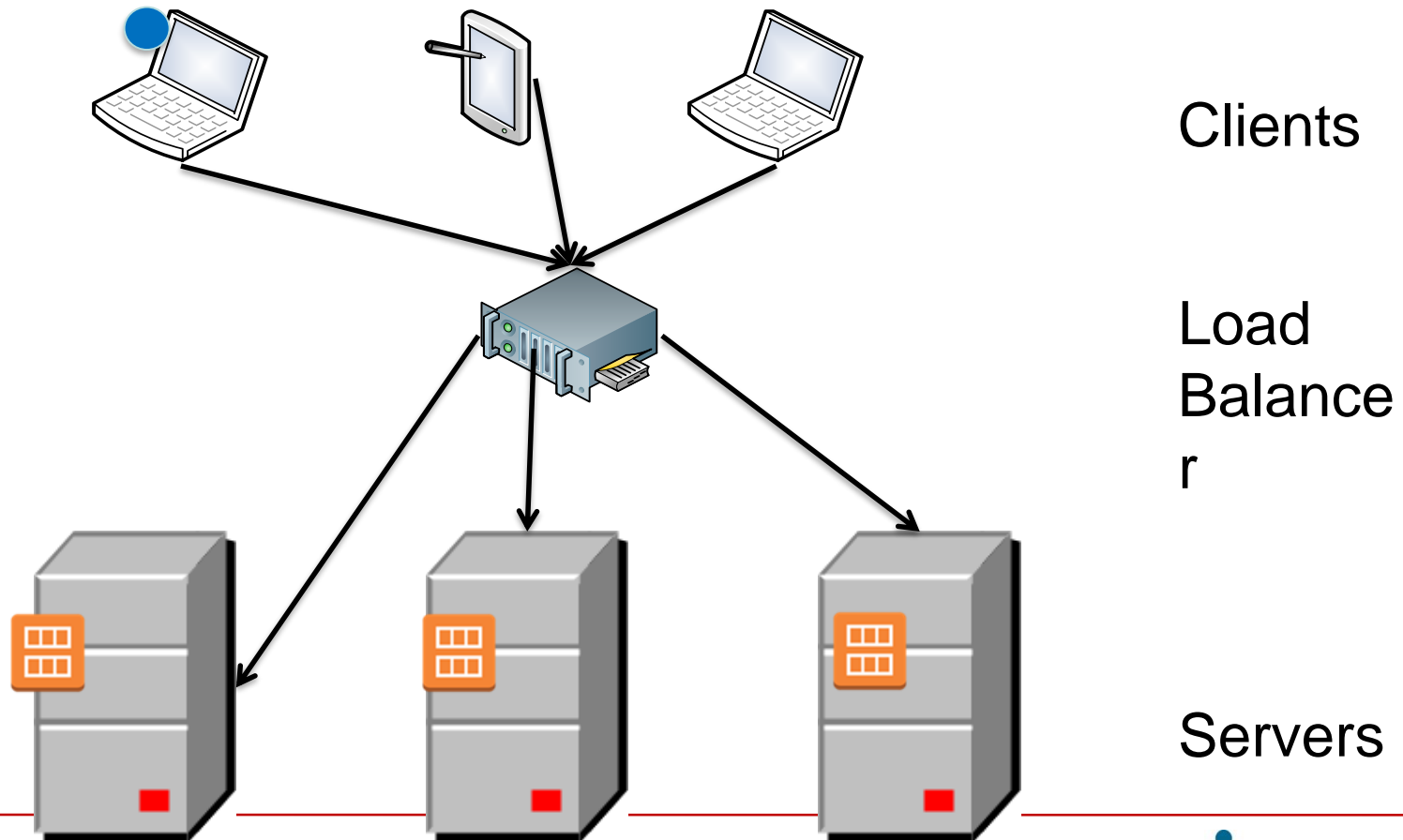
# Message sequence- request arrives at load balancer



# Message sequence – request is send to one server



# Message sequence – reply goes directly back to sender



# Note IP manipulation

- 
- Server always sends message back to what it thinks is the sender.
  - Load balancer changes destination IP but not the source. Then reply goes directly back to client
  - Load balancer (now acting as a proxy) can change origin as well. In this case, reply goes back to load balancer which must change destination (of reply) back to original client.

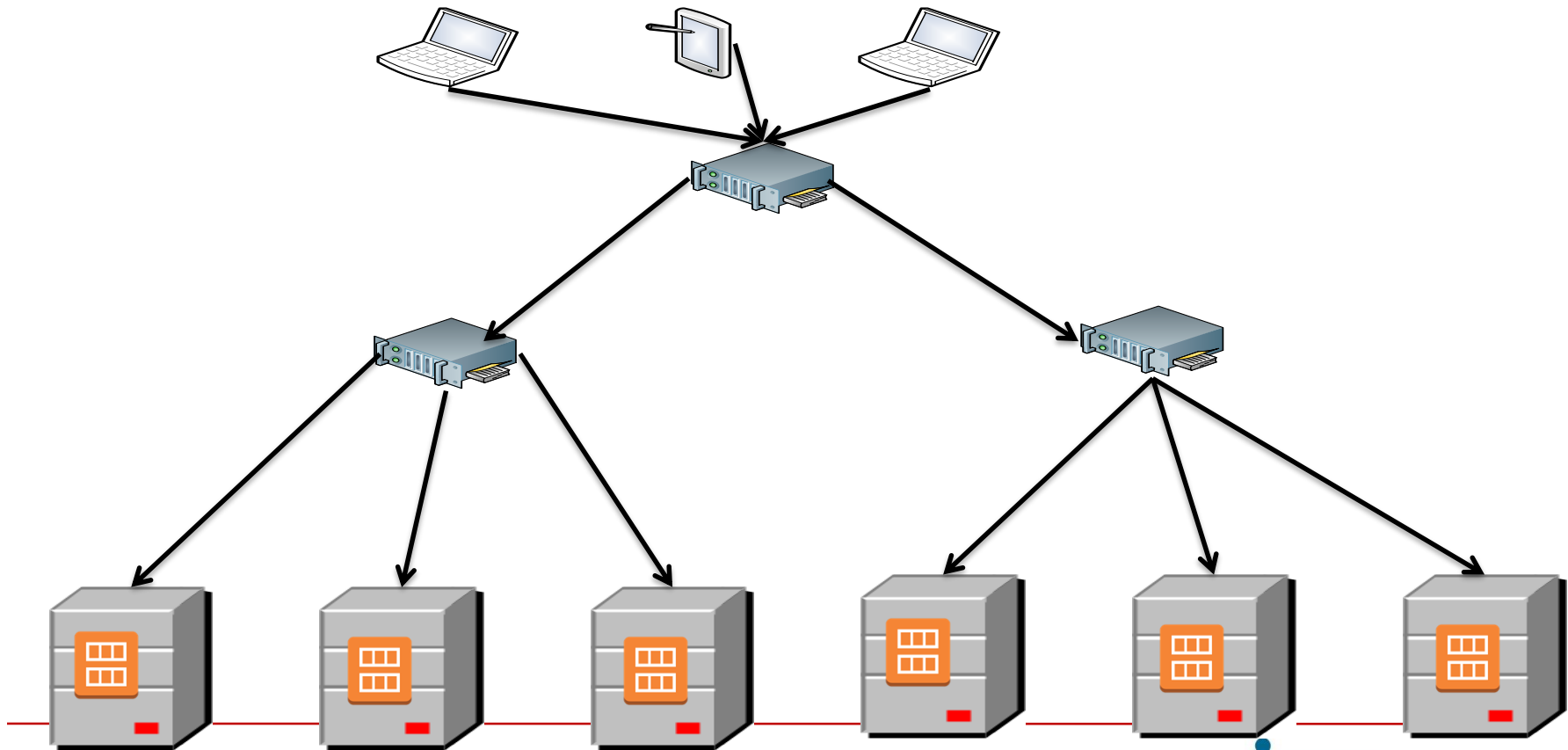


# Routing algorithms

- 
- Load balancers use variety of algorithms to choose instance for message
    - Round robin. Rotate requests evenly
    - Weighted round robin. Rotate requests according to some weighting.
    - Hashing – IP address of source to determine instance. Means that a request from a particular client always sent to same instance as long as it is still in service.
  - Note that these algorithms do not require knowledge of an instance's load.
-

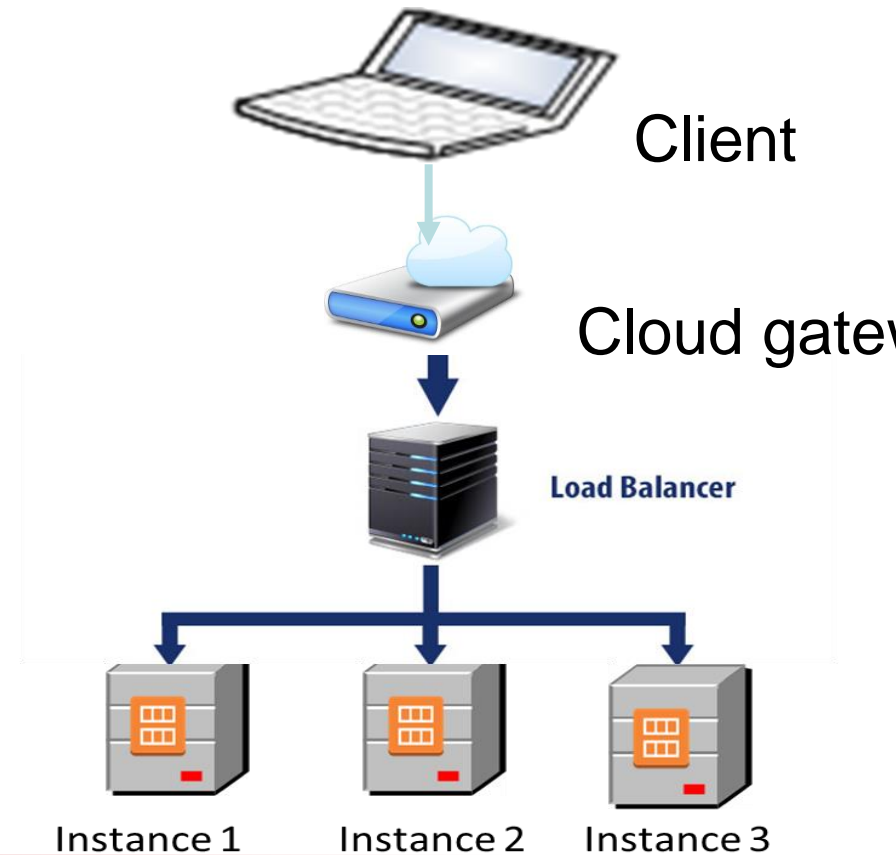
# Suppose Load Balancer Becomes Overloaded

- Load Balance the Load Balancers



# Combining pictures

- IP payload address is modified multiple times before message gets to server
- Return message from instance to client may go through the gateway (proxy) or may go directly back to the client.



# Summary

---

- The cloud consists of regions with availability zones
- A new VM is placed in an availability zone in a region
- Instances in the cloud can fail.
  - Must be detected
  - Long tail is a cloud phenomena that developers must be aware of.
- Load balancers distribute requests among identical servers.



# Deployment and Operations for Software Engineers

Chapter 3 The Cloud - 2

# Overview

---

- **Autoscaling**
- Distributed Coordination

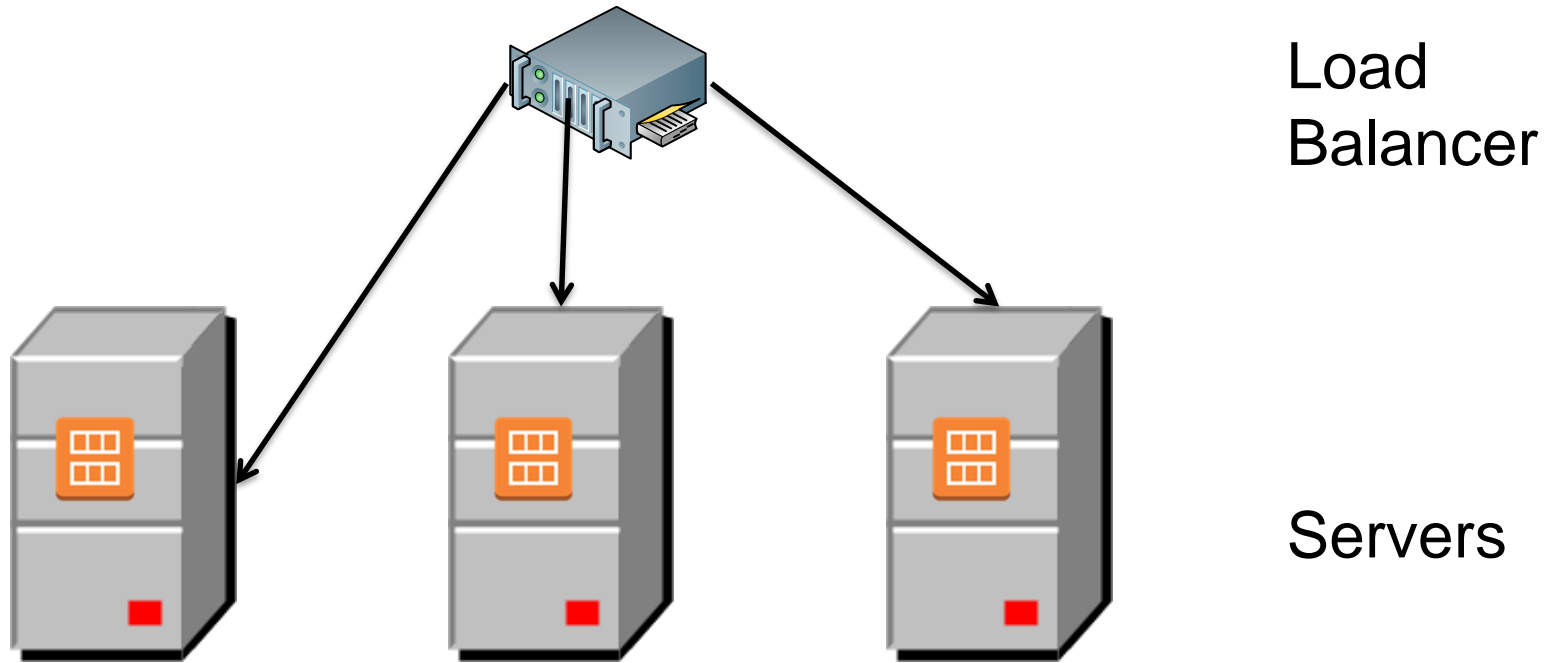
# Suppose servers become overloaded

---

- As load grows, existing resources may not be sufficient.
- Autoscaling is a mechanism for creating new instances of a server.
- Set up a collection of rules that determine
  - Under what conditions are new servers added
  - Under what conditions are servers deleted

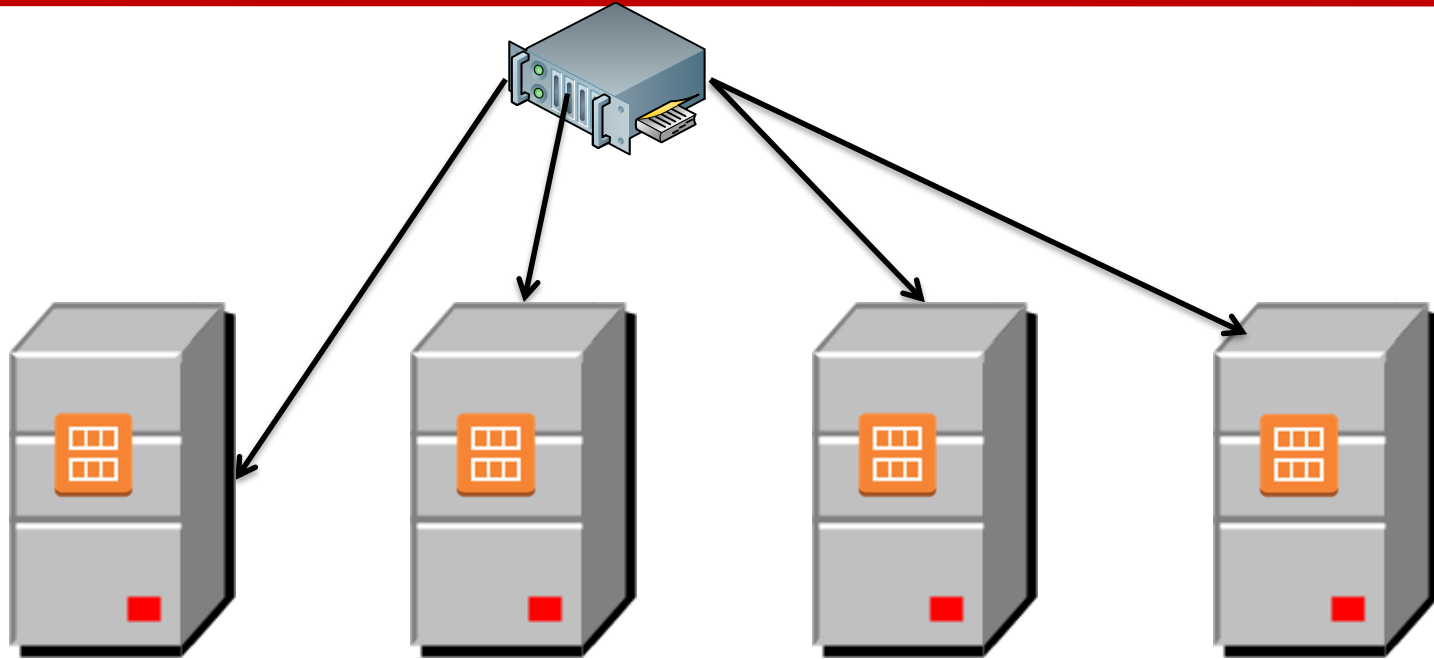
# First there were three servers

---





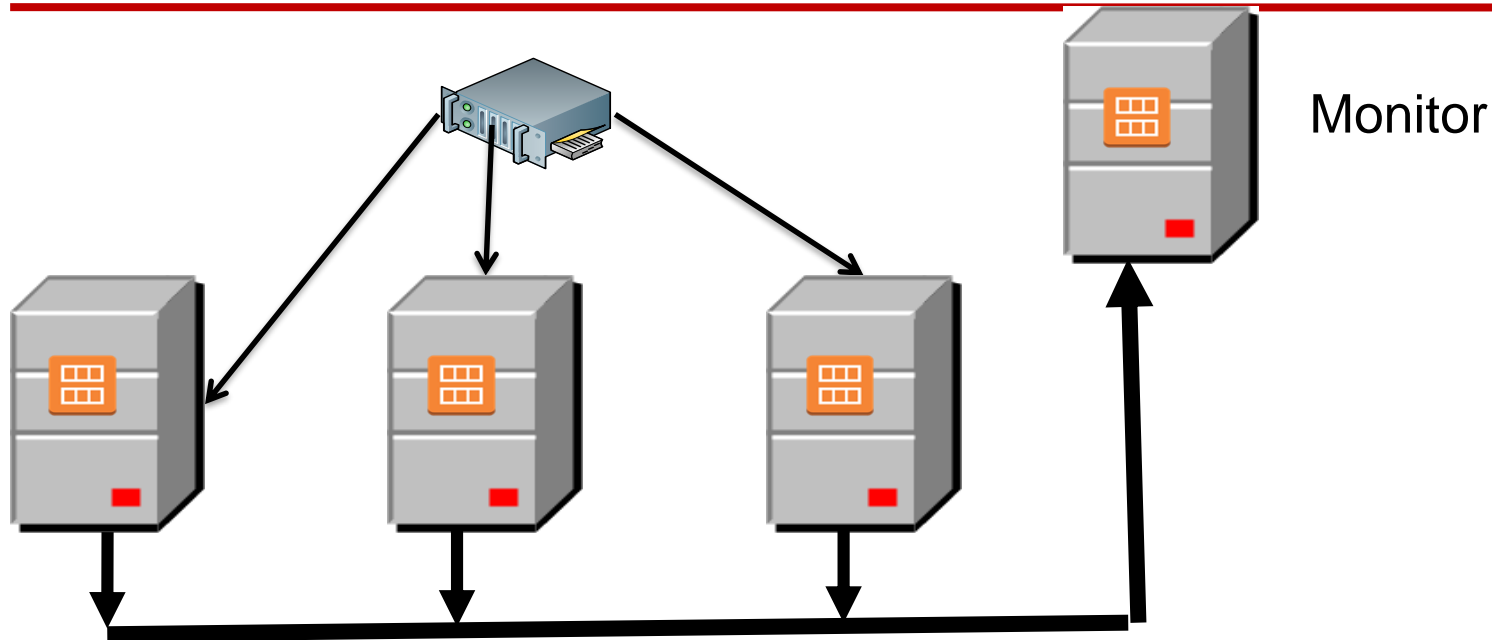
# Now there are four



Issues:

- What makes the decision to add a new server?
- How does the new server get loaded with software?
- How does the load balancer know about the new server?

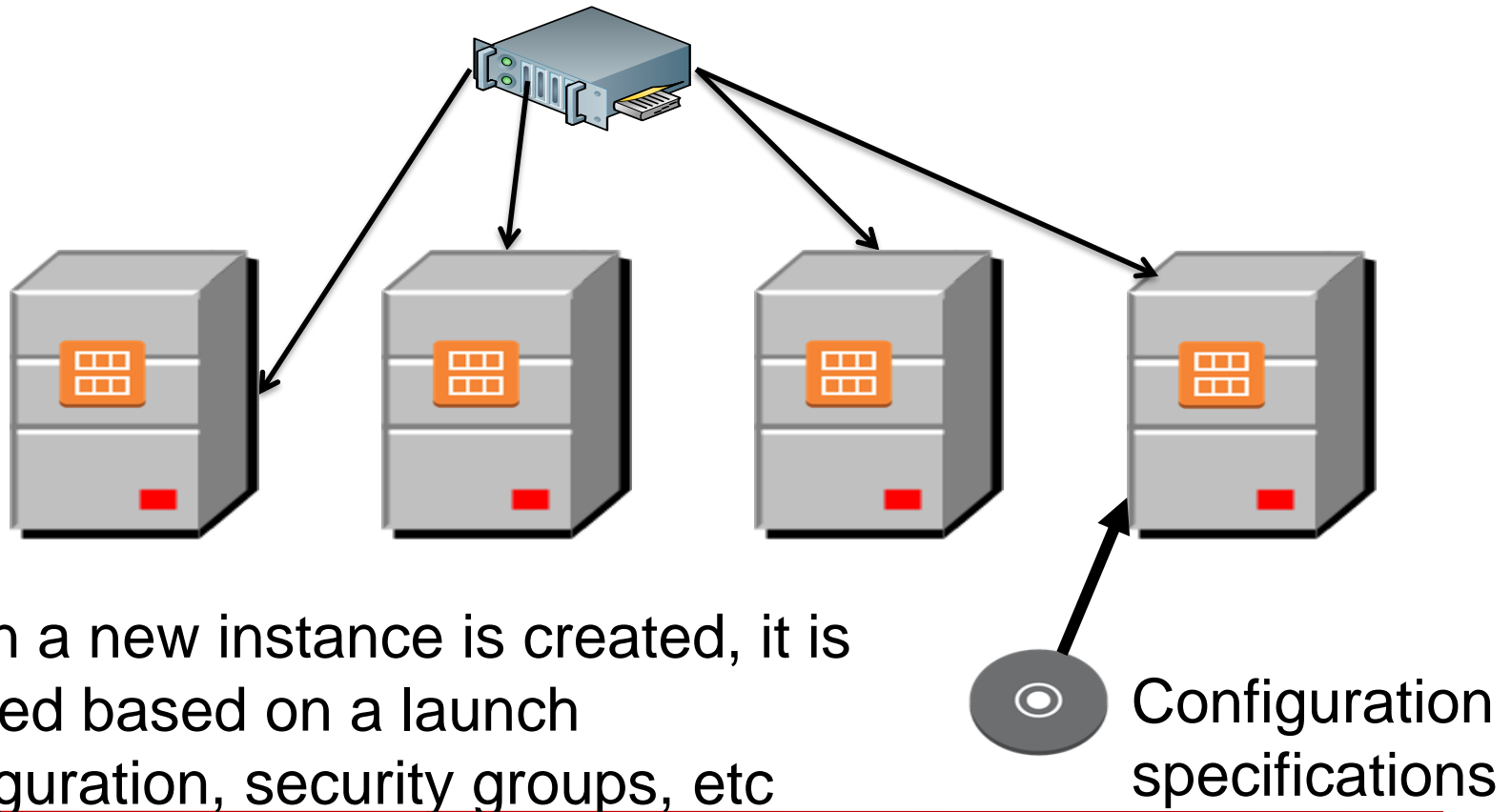
# Making the decision



Each server reports its CPU usage to a monitor  
Monitor has a collection of rules to determine whether to  
add new server. E.g. one server is over 80% utilization for  
15 minutes

# Loading the new server with software

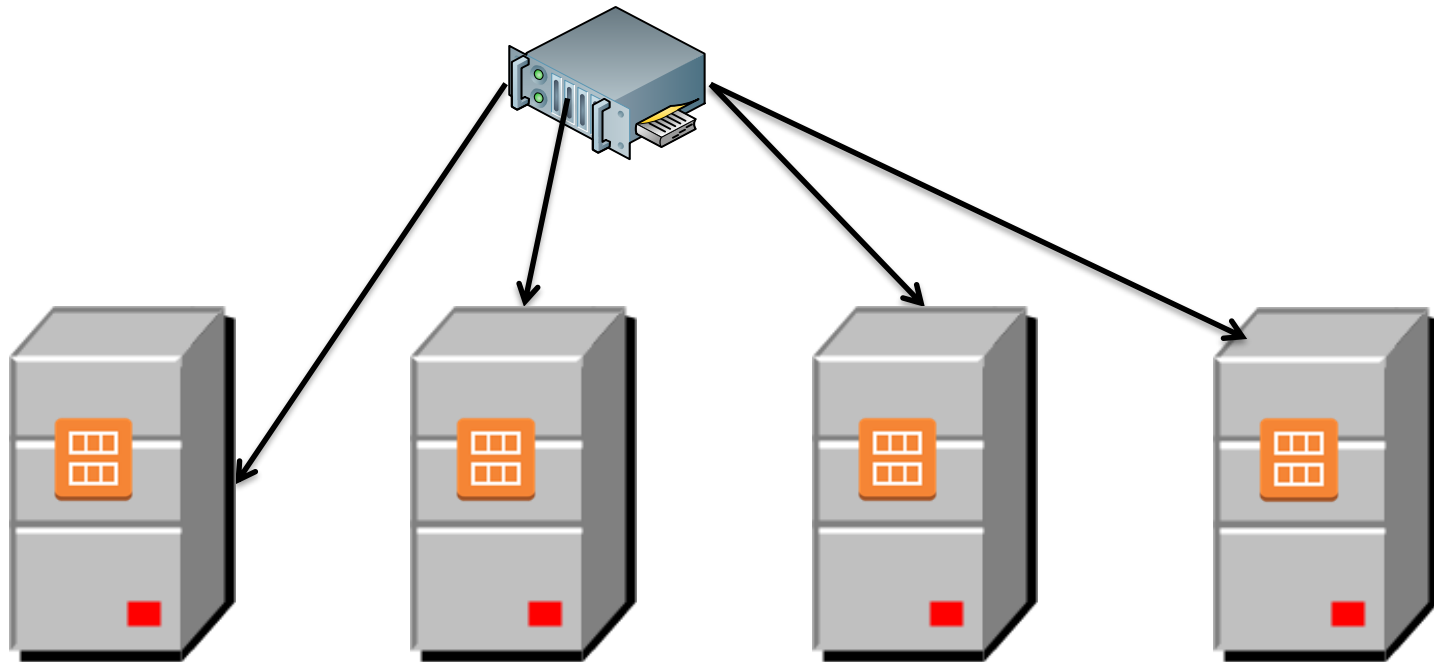
---



When a new instance is created, it is created based on a launch configuration, security groups, etc

# Making the load balancer new server aware

---



The new server is registered with a load balancer by the monitor.

---

# Overview

---

- Autoscaling
- **Distributed Coordination**
  - Atomicity
  - Locking
  - Consensus algorithms

# Atomicity

---

- Some operations must be done atomically (or appear to be atomic).
- Race conditions result from non atomic operations.
  - Data base updates may result in inconsistent data if not atomic
  - Resources (e.g. memory) may be overwritten if not atomic
  - Interrupt handlers must be atomic
  - etc

# How do I achieve (or appear to achieve) atomicity?

---

- Time stamps?
  - Many protocols involve putting a time stamp on messages for error detection and ordering purposes.
  - Time stamps are often used to identify log messages used for debugging problems.
- Couldn't time stamps be used to order activities?
- Problem is that clocks drift and so clock time may be different on different computers.

# Synchronizing clocks over networks

- 
- Suppose two different computers are connected via a network. How do they synchronize their clocks?
  - If one computer sends its time reading to another, it takes time for the message to arrive.
  - NTP (Network Time Protocol) can be used to synchronize time on a collection of computers.
    - Accurate to around 1 millisecond in local area networks
    - Accurate to around 10 milliseconds over public internet
  - ~~Congestion can cause errors of 100 milliseconds or more.~~



# Suppose NTP is insufficiently accurate

---

- Financial industry is spending 100s of millions of dollars to reduce latency between Chicago and New York by 3 milliseconds.
    - Well within error range of NTP
  - GPS time is accurate within
    - 14 nanoseconds (theoretically)
    - 100 nanoseconds (mostly) Atomic
  - Timestamp messages with GPS time
    - Used by electric companies to measure phase angle
  - Atomic clocks
    - Used by Google to coordinate time across all of their distributed systems.
-

# How about using locks?

- 
- One technique to synchronize is to lock critical resources.
  - Can lead to deadlock – two processes waiting for each other to release critical resources
    - Process one gets a lock on row 1 of a data base
    - Process two gets a lock on row 2.
    - Process one waits for process 2 to release its lock on row 2
    - Process two waits for process 1 to release its lock on row 1
  - No progress.
-

# More problems with locks

- 
- Locks are logical structures maintained in software or in persistent storage.
  - Getting a lock across distributed systems is not an atomic operation.
    - It is possible that while requesting a lock another process can acquire the lock. This can go on for a long time (it is called livelock if there is no possibility of ever acquiring a lock)
  - Suppose the virtual machine holding the lock fails. Then the owner of the lock can never release it.

# Locks in distributed systems

---

- Utilizing locks in a distributed system has a more fundamental problem.
- It takes time to send a message (create a lock, for example) from one computer to another.

# How much time do operations in distributed systems take

---

- Main memory reference 100 ns
  - Send 1K bytes over 1 Gbps network 0.01 ms
  - Read 4K randomly from SSD. 15 ms
  - Read 1 MB sequentially from memory 0.25 ms
  - Round trip within same datacenter 0.5 ms
  - Read 1 MB sequentially from SSD 1 ms  
(4X memory)
  - Disk seek 10 ms  
(20x datacenter roundtrip)
  - Read 1 MB sequentially from disk 20 ms  
(80x memory, 20X SSD)
  - Send packet CA->Netherlands->CA 150 ms
-

# Implications of latency numbers

---

- State stored in persistent storage (disk or SSD) will take longer to fetch than state stored in memory.
- State stored in a different datacenter will take longer to access than state stored locally, especially across continents.
- Persistent store is typically replicated both for performance (latency) reasons and for availability (failure) reasons.
- => keeping data consistent across different occurrences is important but difficult.

# Is there a solution?

- 
- The general problem is that you want to manage synchronization of data across a distributed set of servers where up to half of the servers can fail.
  - Paxos is a family of algorithms that use consensus to manage concurrency. Complicated and difficult to implement.
  - An example of the implementation difficulty
    - Choose one server as the master that keeps the “authoritative” state.
    - Now that server fails. Need to
      - Find a new master
      - Make sure it is up to data with the authoritative state.

# Luckily

- 
- Several open source systems are now available that
    - Implement Paxos or an alternative consensus algorithm
    - Are reasonably easy to use.
  - Sample systems
    - Zookeeper
    - Consul
    - etcd



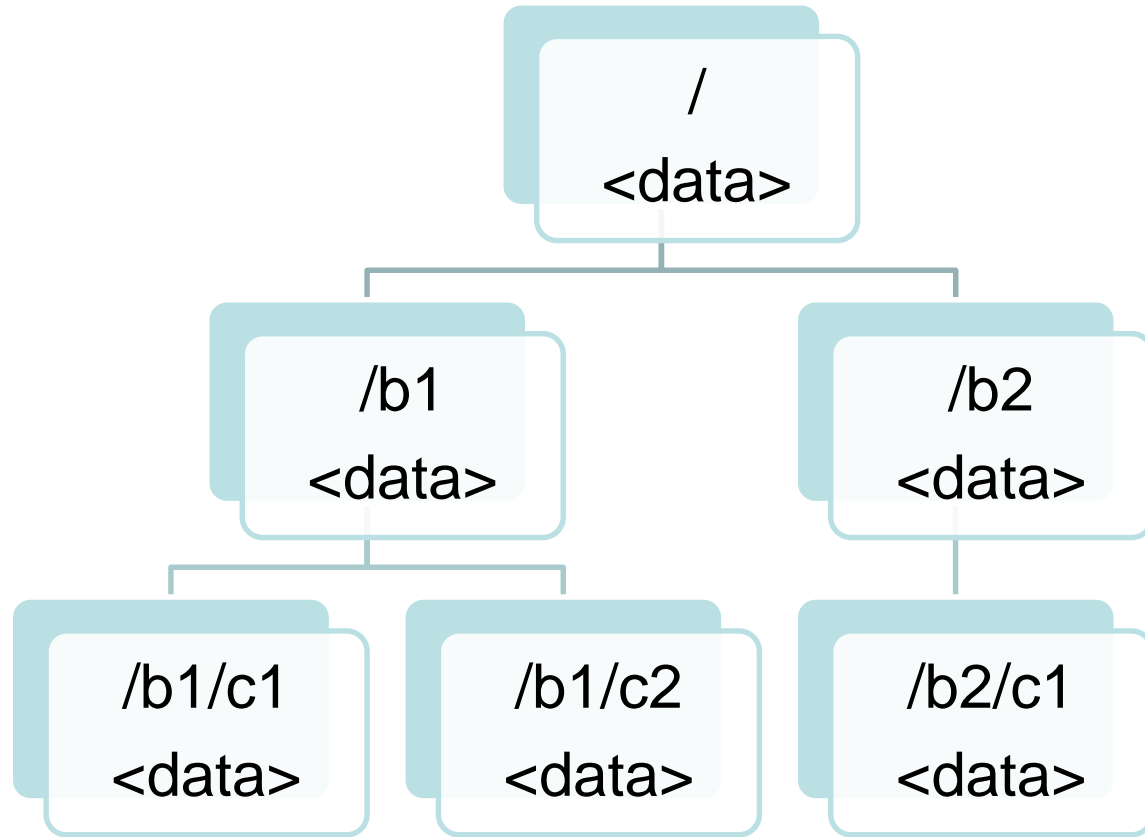
# Zookeeper as an example

---

- Zookeeper provides a guaranteed consistent (mostly) data structure for every instance of a distributed application.
  - Definition of “mostly” is within eventual consistency lag (but this is small).
  - Zookeeper keeps data in memory. This is why it is fast.
- Zookeeper deals with managing failure as well as consistency.
  - Done using consensus algorithm.

# Zookeeper znode structure

---



# Zookeeper API

- All calls return atomic views of state – either succeed or fail. No partial state returned. Writes also are atomic. Either succeed or fail. If they fail, no side effects.

Function	Type
create	write
delete	write
Exists	read
Get children	Read
Get data	Read
Set data	write
+ others	

# Example – Lock creation

- 
- Locks have names.
  - Client N
    - Create Lock1
      - If success then client owns lock.
      - If failure, then it is placed on waitlist for lock 1 and control is returned to Client N
    - When client finishes, it deletes Lock1
    - If there is a waitlist, then those clients are informed about Lock 1 deletion and they try again to create Lock 1
  - If Client N fails, Zookeeper will delete Lock 1 and waitlist clients are informed of deletion
-

# Other use cases

---

- Leader election
- Groups membership
- Synchronization
- Configuration

# Summary

- 
- Autoscaling creates new instances based on utilization of existing instances
  - Synchronization of data across a distributed system is complicated
  - It is difficult because of
    - Timing delays across distributed systems
    - Possible failures of nodes of various types
  - Open source systems implement complicated consensus algorithms in a fashion that is relatively easy to use.



# Deployment and Operations for Software Engineers

## Chapter 4 Container Management

# Overview

---

- **Container repositories**
- Clusters and orchestration
- Serverless Architecture



# Container repositories

---

- Container images are typically stored in repositories
  - Similar to version control systems
    - Accessible with permissions
    - Push/pull interface
    - Images can be tagged with version numbers
  - Docker Hub is a publicly available repository
-

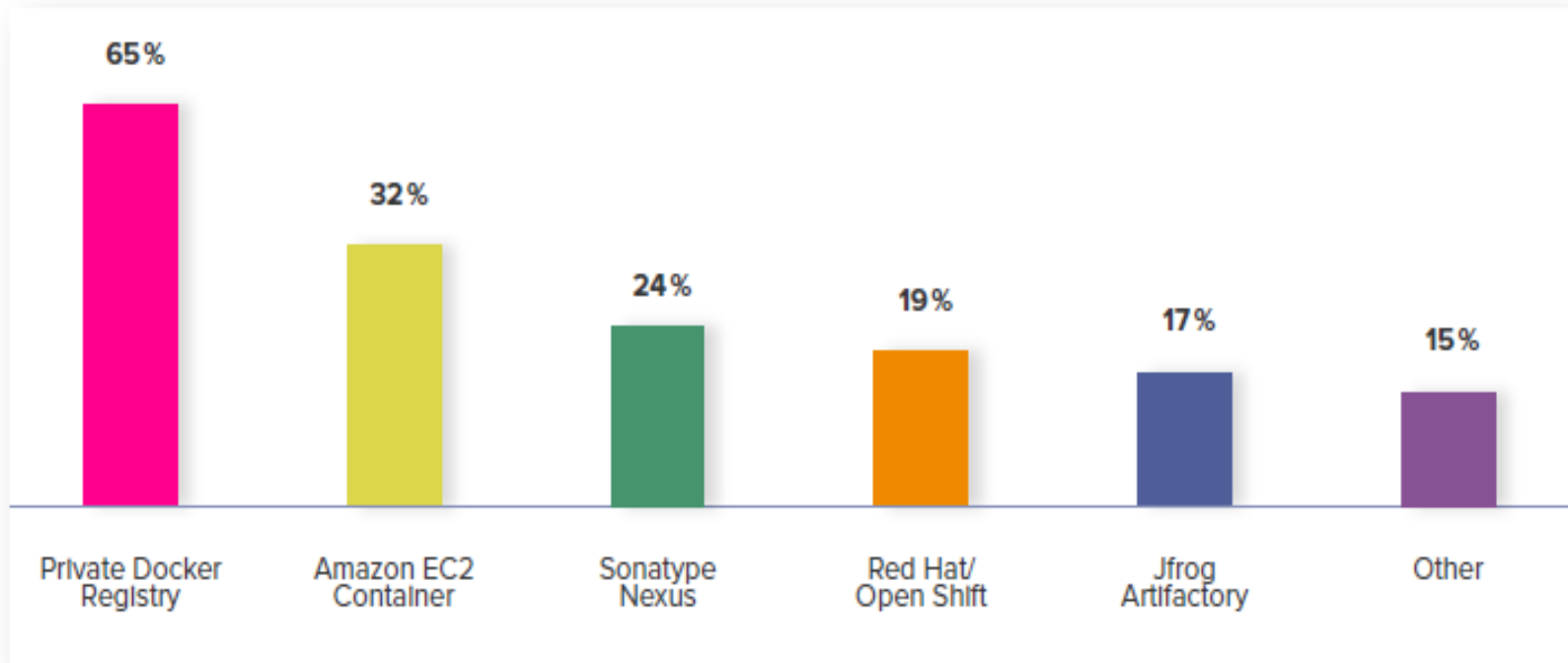
# Integrating with development workflow

---

- Multiple team members may wish to share images
- Images can be in production, under development or under test
- Private container repository allows images to be stored and shared.
  - Any image can be “pulled” to any host
  - Tagging as “latest” allows updates to be propagated. Pull <image name>:latest gets the last image checked into repository with that name.

# Private container registries used by organizations

---



# Overview

---

- Container repositories
- **Clusters and orchestration**
- Serverless Architecture

# Allocation of images to hosts

---

images



To run an image, the image and the host must be specified

hosts



With basic Docker this allocation must be done manually

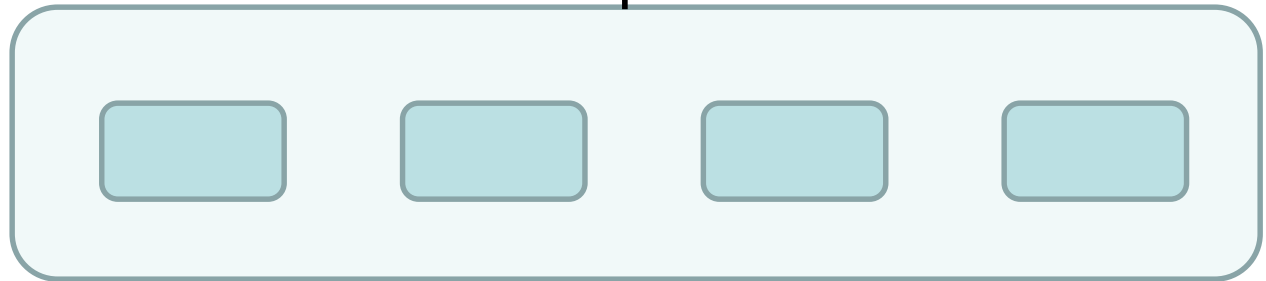
# Container orchestrator

image



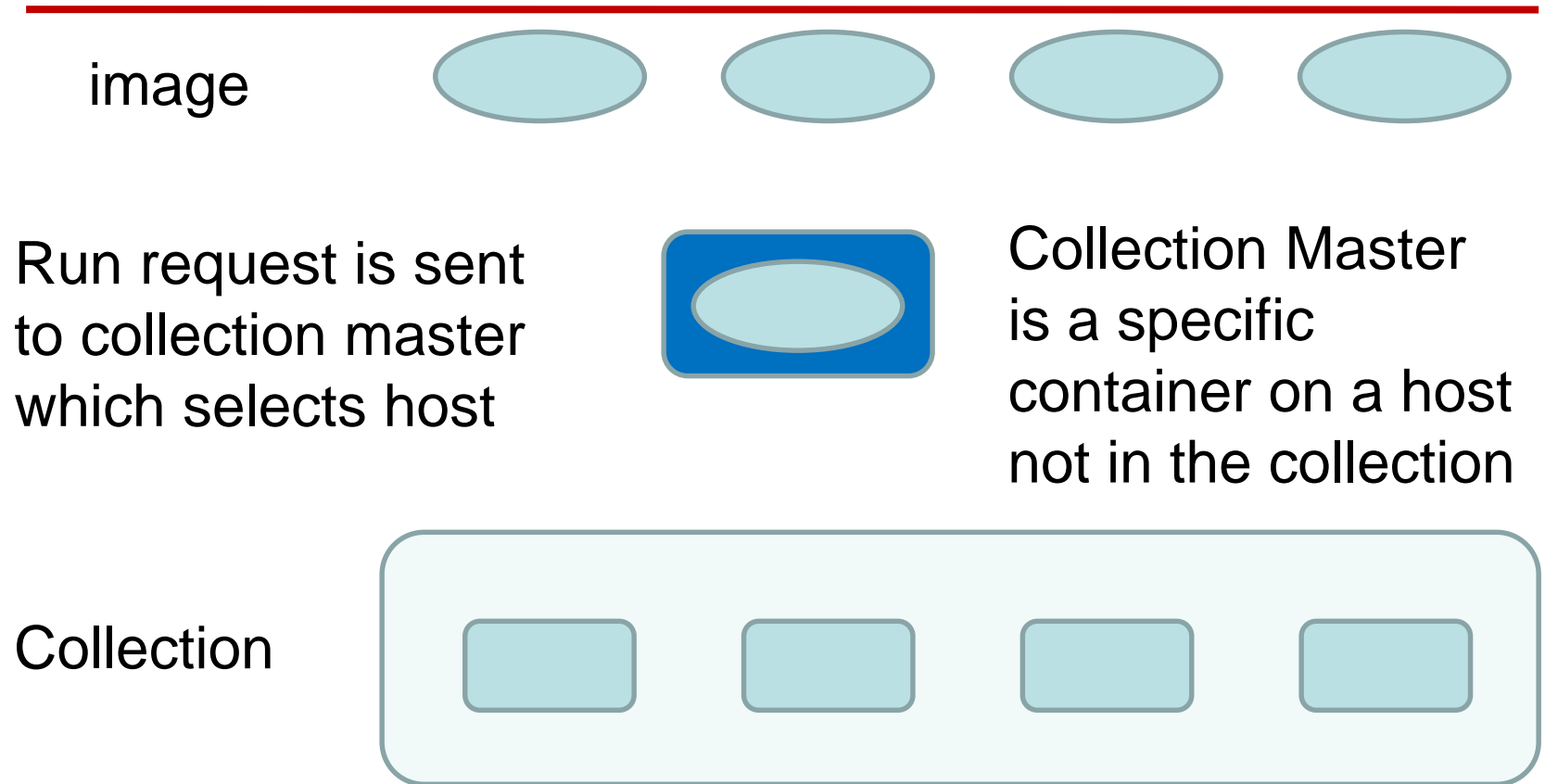
To run an image, the image but not the host must be specified

Orchestrator  
encapsulates  
hosts into a  
collection



A collection looks like a single host  
from the point of view of allocation but  
actually consists of multiple hosts

# Collection Master

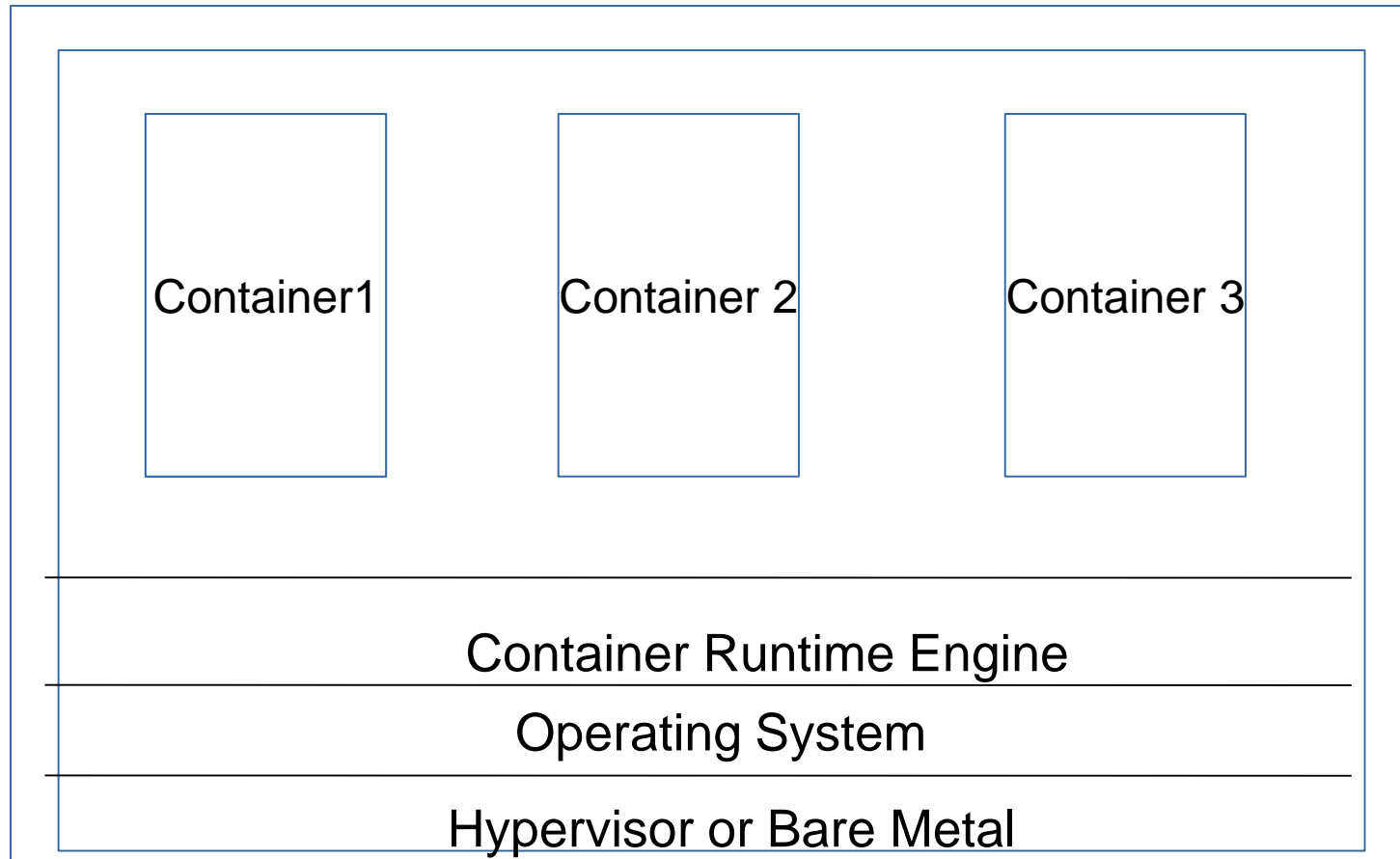


# Grouping containers

- 
- Up to this point, containers are arbitrarily assigned to available host without regard for communication among containers.
  - Suppose two containers communicate frequently. E.g. app container and logging containers. Then you would like them to be allocated into same host to reduce communication time.
  - This is rationale for “pods”.
  - A pod is a group of containers treated as a single unit for allocation.

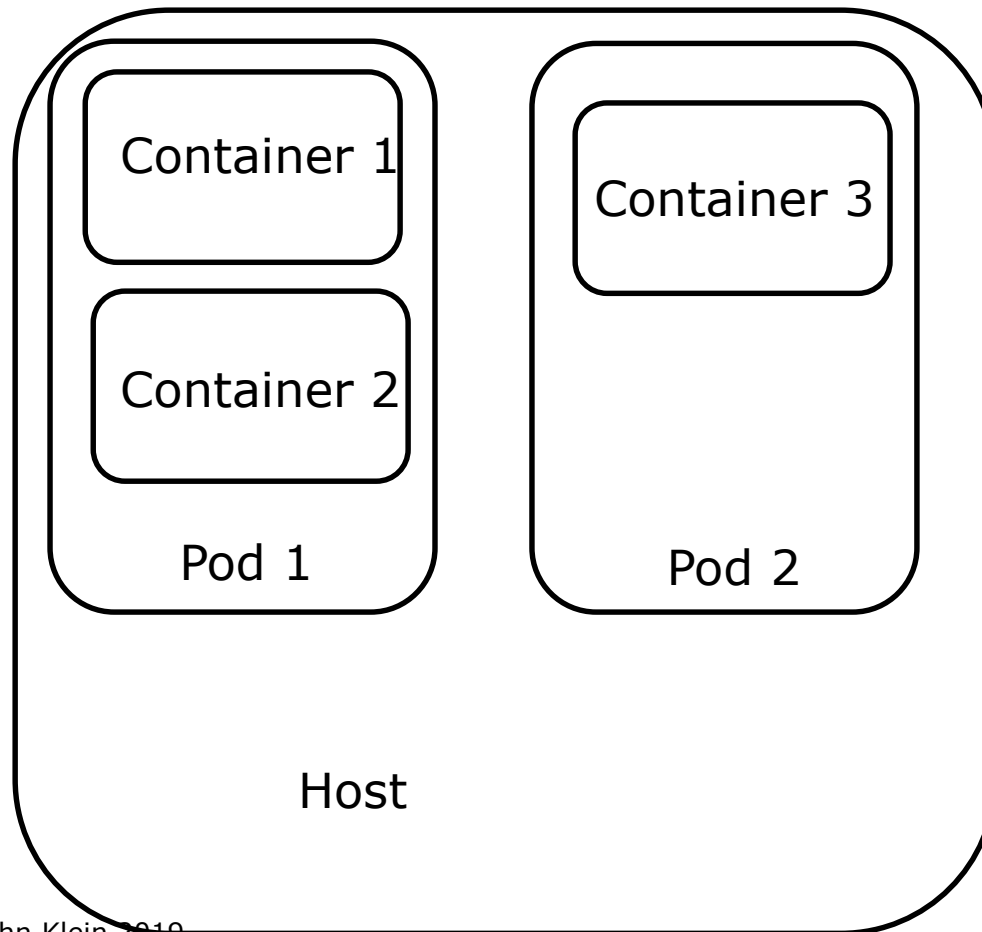


# Container hierarchy (again)

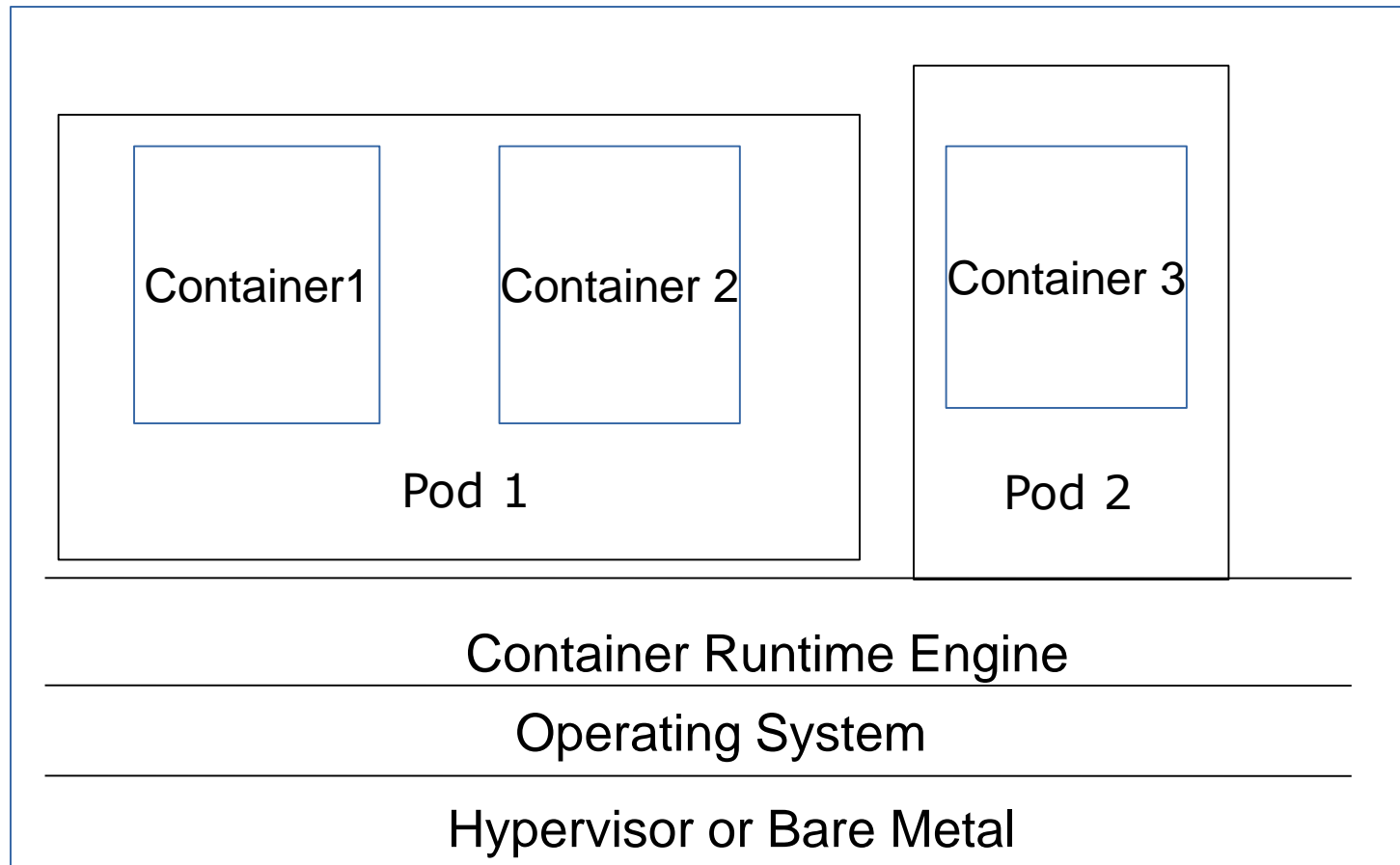


# Adding pods to hierarchy

---



# Revisiting container hierarchy



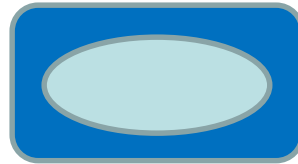
# Collection Master revisited with pods

---

Images collected  
into pods

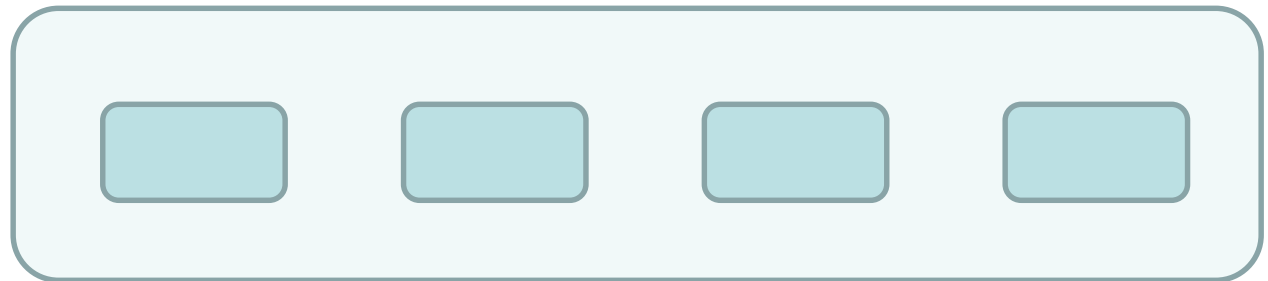


Run request is sent  
to collection master  
which selects host



Collection Master  
is a specific  
container on a host  
not in the collection

Collection



# Scaling collections

---

- Having an instance in a collection be automatically replicated depending on workload is accomplished by utilizing autoscaling facilities of orchestrator
- Kubernetes is a container scaling and orchestration engine that supports pods.

# Overview

---

- Container repositories
- Clusters and orchestration
- **Serverless Architecture**

# Serverless

- 
- Cloud providers such as AWS maintain pool of partially loaded containers that only require application specific layer. The term “serverless” is used to describe these partially loaded containers.
  - Load in milli secs.
  - For AWS, only one request per instance. In AWS, serverless hosts are called Lambda
  - Impacts architecture of application.
-

# Summary

---

- Repositories of container images act much like version control systems
- Containers can be clustered and can be deployed and scaled by cluster.
- Serverless architecture takes advantage of fast load time of container images but with restrictions on usage.





# Deployment and Operations for Software Engineers

Chapter 5 Infrastructure Security - 1

# Overview

---

- **What is security?**
- Cryptography
- Public Key Infrastructure and Certificates

# Basic security definition

- 
- Short (but memorable form) - CIA
    - Confidentiality
    - Integrity
    - Availability

# More precise security definition

---

- **Authentication:** assurance that communicating entity is the one claimed
  - **Access Control:** prevention of the unauthorized use of a resource
  - **Data Confidentiality** protection of data from unauthorized disclosure
  - **Data Integrity** assurance that data received is as sent by an authorized entity
  - **Non-Repudiation** protection against denial by one of the parties in a communication
  - **Availability** – resource accessible/usable
-

# Categorizing Security Activities

---

- Identify: Identify the valuable data or resources.
- Protect: Protect valuable data.
- Detect: Detect attacks
- React: If a breach or intrusion occurs, you must react.
- Recover: Recover from a security incident.

# Overview

---

- What is security?
- **Cryptography**
- Public Key Infrastructure and Certificates

# Cryptography

---

- Three forms of encryption
  - Symmetric
  - Asymmetric
  - One way - hash
- NIST (US National Institute for Science and Technology) certifies algorithms and implementations for encryption.

# Data

---

- One more concept – data
  - At rest – on disk or in memory
  - In transit - on the network



# Symmetric encryption

---

- Use same key for encrypting and decrypting
- 4000x faster than asymmetric encryption
- Suitable for data at rest
- Diffie-Hellman (discussed shortly) is an algorithm for establishing a symmetric key

# Weaknesses of Symmetric encryption

---

- If attacker discovers key, then has access to all encrypted data
- No authentication with symmetric encryption
- NIST approved algorithm is AES with key lengths of  $>128$  bits

# Asymmetric Encryption

---

- Also known as public/private key encryption
- Uses different keys for encryption and decryption
- Based on difficulty of factoring product of two large primes (NP difficult)
- NIST approved algorithms: DSA, RSA, ECDSA >1024 bits

# Hashing

---

- A hash is a one way encryption based on a public algorithm with no key
- Not possible (very difficult) to decrypt
- Used to verify integrity of data
  - Passwords: save hash of password but not password. When user enters password, compare to hash to verify.
  - Downloads: publish hash of software available for download. Compare hash of downloaded software. Verifies that software has not been modified.
- NIST approved algorithm is SHA-3.

# Overview

---

- What is security?
- Cryptography
- **Public Key Infrastructure and Certificates**

# Public Key Infrastructure (PKI)

---



- PKI is based on public and private keys
- Public key is known to everyone
- Private key is known only to you
- Messages encrypted with public key can be decrypted by private key (and vice versa)
- Message sent *to* you is encrypted with your public key. Only you can read it
- Message sent *by* you is encrypted with your private key. Decrypting it with your public key guarantees that you sent it

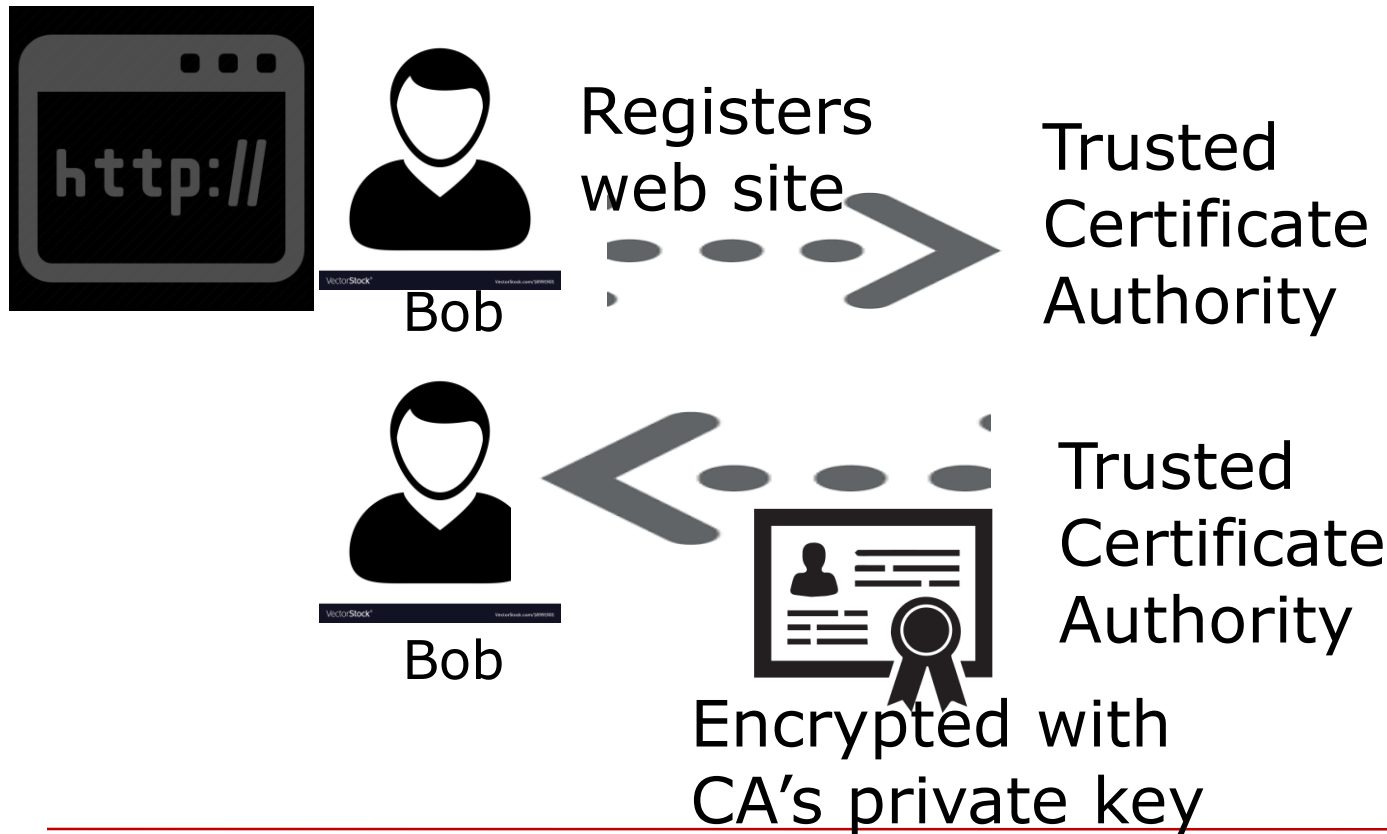
# Certificates

---

- Certificates are used to establish that a web site is what it claims to be.
- Certificates are based on PKI
- Two important elements of a certificate
  - URL of web site that has been certified
  - Signature of a trusted certificate authority

# Registering with Certificate Authority

---





# Accessing Web Site



Alice

Accesses  
Bob's web  
site



Alice decrypts using  
CA's public key and  
knows she is talking  
to Bob's web site



# Format of a certificate

## End-entity Certificate

Owner's name
Owner's public key
Issuer's (CA's) name
Issuer's signature

*reference*

## Intermediate Certificate

Owner's (CA's) name
Owner's public key
Issuer's (root CA's) name
Issuer's signature

*reference*

*sign*

*self-sign*

Root CA's name
Root CA's public key
Root CA's signature

## Root Certificate

# Summary

---

- Security is CIA + authentication + authorization
  - Three different encryption styles.
    - Symmetric
    - Asymmetric
    - Hashing
  - PKI is based on Asymmetric encryption and is the basis for certificates to validate web sites.
-



# Deployment and Operations for Software Engineers

Chapter 5 Infrastructure Security - 2

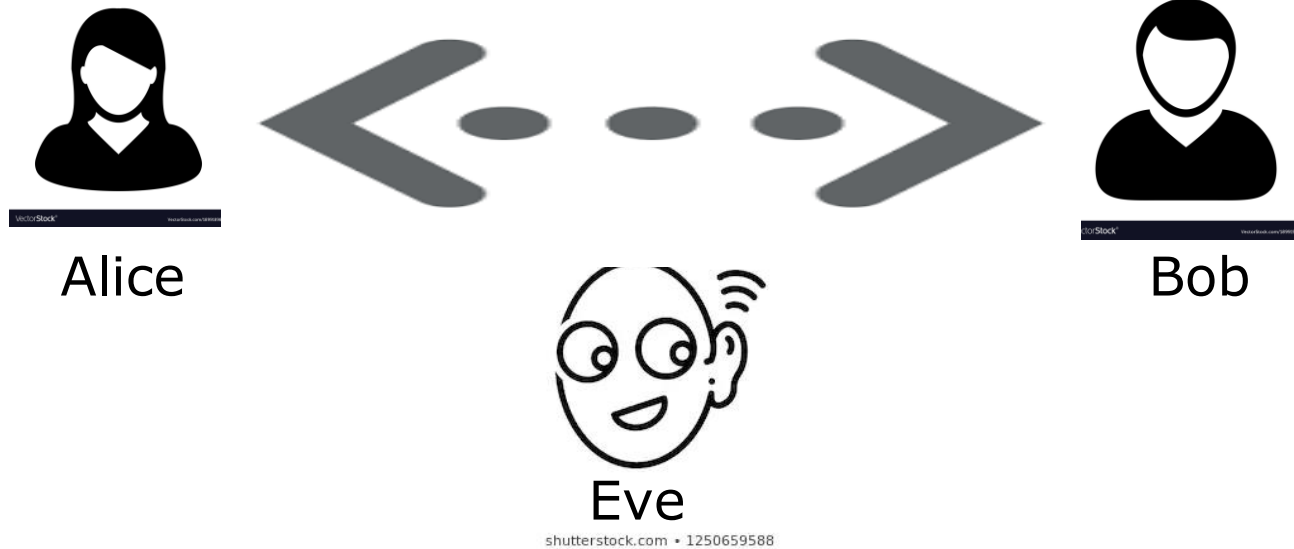
# Overview

---

- **Key Exchange**
- Transport Level Security (TLS)
- Secure Shell (SSH)
- Secure File Transfer
- Intrusion Detection

# Key exchange

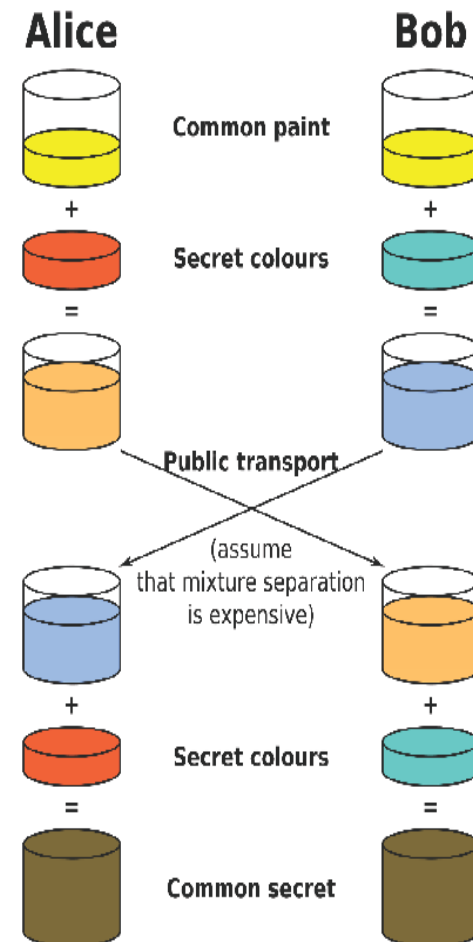
---



Alice and Bob wish to communicate securely even though Eve is eavesdropping.  
Diffie-Hellman algorithm allows this.

# Diffie-Hellman (intuitively)

- Alice and Bob agree on a common color
- Each chooses a secret color
- Each mixes their secret color with the common color
- Each sends their mixture to the other
- Each now adds their secret color
- Alice and Bob end up with the same color but decoding it is difficult
- This color is the shared key for symmetric encryption



# Overview

---

- Key Exchange
- **Transport Level Security (TLS)**
- Secure Shell (SSH)
- Secure File Transfer
- Intrusion Detection



# Man in the middle attack

---

- You are in the airport scanning for an available ISP
- You find “freewifi” and get an IP address from them.
- “freewifi” may be an attacker
- “freewifi” can modify messages to spoof you and steal your credentials

# TLS

---

- TLS (Transport Layer Security) is the basis for https
- Builds on Diffie-Hellman and PKI
- Thwarts man-in-the-middle attacks

# TLS protocol - 1

---

- You wish to access a web site using https
  - Handshake to determine which version of the protocol to use
  - Web site sends certificate to you to demonstrate authenticity. Recall certificate is encrypted using CA private key.
  - Your OS has been pre-loaded with public keys of major CAs so you can decrypt certificate.
-

# TLS protocol – 2

---

- Diffie-Hellman is used to establish secure key
- Information being exchanged is encrypted/decrypted using this key
- Once session is terminated, key is discarded.
- If you reconnect to the same web site the protocol is used to establish a different secure key

# Thwarting man in the middle

---

- Man in the middle may see all messages but
  - Credential is encrypted so it cannot be modified
  - Diffie-Hellman protects against eavesdropper (the man in the middle)
  - Your communication with web site is encrypted using key unknown to man in the middle

# Overview

---

- Key Exchange
- Transport Level Security (TLS)
- **Secure Shell (SSH)**
- Secure File Transfer
- Intrusion Detection

# SSH

- 
- Secure Shell (SSH) is a standard protocol and supporting software that enables the control of one computer remotely from another
  - Uses public/private key but SSH is unrelated to PKI and TLS
  - SSH has a concept of “known addresses” that allows logging into remote computer without a password.
  - SSH is used by tools to provision and manage collections of computers.

# TLS vs SSH

---

- TLS allows communication between two arbitrary parties using PKI
- SSH allows communication where one party knows the IP address it wishes to communicate with
- Could TLS be used instead of SSH? Yes, but:
  - They have different historical roots and SSH is very embedded in practice
  - Using TLS would require having certificates for many more machines.



# Overview

---

- Key Exchange
- Transport Level Security (TLS)
- Secure Shell (SSH)
- **Secure File Transfer**
- Intrusion Detection

# Secure File Transfer

---

- Two families of secure file transfer
  - FTPS (FTP+TLS)
    - Operating system agnostic
    - Can transfer text or binary
  - SFTP (SSH + FTP)
    - Binary transfer only
    - Designed for Unix based systems although there are utilities for other operating systems

# Overview

---

- Key Exchange
- Transport Level Security (TLS)
- Secure Shell (SSH)
- Secure File Transfer
- **Intrusion Detection**

# Intrusion Detection Systems (IDS)

---

- Host based.
- Network based

# Host Based IDS

---

- Runs on physical or virtual machine under control of an operating system.
- Looks for anomalous signatures of files
- Relies on a database of known attack signatures

# Network Based IDS

---

- Specialized machine that monitors all network traffic
- Looks for attack patterns
  - Port scans
  - Failed login
  - Other anomalous traffic patterns
    - May generate false positives
    - Because network traffic is very diffuse difficult to detect anomalous patterns

# Summary

---

- Diffie-Hellman supports secure communication even when there is an eavesdropper
  - TLS builds on Diffie-Hellman to thwart man in the middle attacks
  - SSH allows one computer to control another. Used heavily in operations
  - Secure file transfer is built on top of SSH or TLS
  - Intrusion detection systems can be either host based (anti virus) or network based (monitoring network traffic)
-



# Deployment and Operations for Software Engineers

Chapter 6 Microservice Architecture - 1



# Overview

---

- Definition
- Microservices and DevOps
- Quality attributes

# Definition

---

- A microservice architecture is
  - A collection of independently deployable processes
  - Packaged as services
  - Communicating only via messages

# ~2002 Amazon instituted the following design rules - 1

---

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

# Amazon design rules - 2

- 
- It doesn't matter what technology they[services] use.
  - All service interfaces, without exception, must be designed from the ground up to be externalizable.
  - Amazon is providing the specifications for the "Microservice Architecture".

# In Addition

- Amazon has a “two pizza” rule.
- No team should be larger than can be fed with two pizzas (~7 members).
- Each (micro) service is the responsibility of one team
- This means that microservices are small and intra team bandwidth is high
- Large systems are made up of many microservices.
- There may be as many as 140 in a typical Amazon page.



# Services can have multiple instances

---

- The elasticity of the cloud will adjust the number of instances of each service to reflect the workload.
- Requests are routed through a load balancer for each service
- This leads to
  - Lots of load balancers
  - Overhead for each request.

# Digression into Service Oriented Architecture (SOA)

---

- The definition of microservice architecture sounds a lot like SOA.
- What is the difference?
- Amazon did not use the term “microservice architecture” when they introduced their rules. They said “this is SOA done right”

# SOA typically has but microservice architecture does not

---

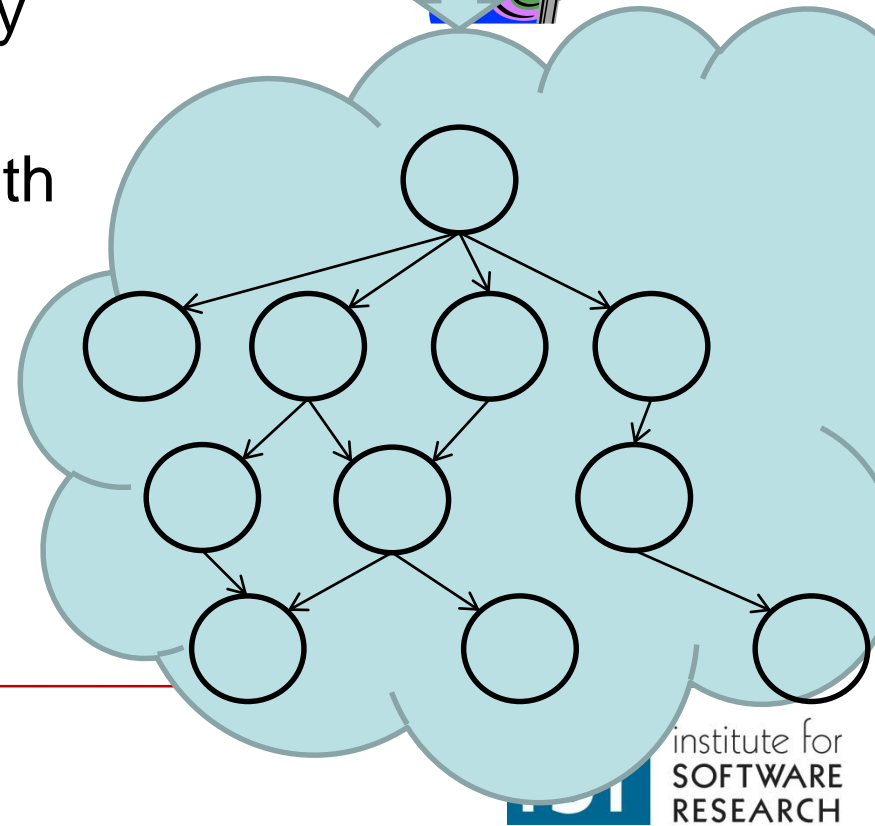
- Enterprise service bus
- Elaborate protocols for sending messages to services (WDSL\*)
- Each service may be under the control of different organization
- Brokers
- etc



# Micro service architecture

- Each user request is satisfied by some sequence of services.
- Most services are not externally available.
- Each service communicates with other services through service interfaces.
- Service depth may
  - Shallow (large fan out)
  - Deep (small fan out, more dependent services)

○  
Service



# Microservice architecture supports DevOps processes

---

- Reduces need for coordination
- Reduces pipeline errors
- Supports continuous deployment

# How does microservice architecture reduce requirements for coordination?

---

- Coordination decisions can be made
  - incrementally as system evolves or
  - be built into the architecture.
- Microservice architecture builds most coordination decisions into architecture
- Consequently they only need to be made once for a system, not once per release.

# Seven Decision Categories

- 
- Architectures can be categorized by means of seven categories
    1. Allocation of functionality
    2. Coordination model
    3. Data model
    4. Management of resources
    5. Mapping among architectural elements
    6. Binding time decisions
    7. Technology choices

# Design decisions with microservices

---

- Microservice architecture either specifies or delegates to the development team five out of the seven categories of design decisions.
  1. Allocation of responsibilities.
  - ~~2. Coordination model.~~
  3. Data model.
  - ~~4. Management of resources.~~
  - ~~5. Mapping among architectural elements.~~
  - ~~6. Binding time decisions.~~
  - ~~7. Choice of technology~~

# How do microservices reduce pipeline errors?

---

- Many integration errors are caused by incompatibilities.
- Each team makes its own technology choices for their services. This reduces errors during integration
  - No problems with inconsistent versions of dependent software
  - No problems with language choices

# How do microservices support continuous deployment?

---

- Each team can deploy their service when it is ready without consultation with other teams.
  - Since services are small, this can happen frequently
  - New feature implementation in one service can be deployed without waiting for that feature to be implemented in other services.
  - Requires architectural support – discussed in lectures on deployment

# Quality attributes

---

- Availability
- Modifiability
- Performance
- Reusability



# Availability

---

- Availability is achieved using
  - Multiple instances
  - Timeouts to recognize failure
  - Stateless instances
- If instances are stateless, when an instance failure occurs a new instance can be created and placed into service

# Modifiability

---

- Modifiability depends on the design of the various microservices.
  - Low coupling and high cohesion are design principles to achieve modifiability.
  - Also, a catalog of microservices must be maintained since there will be many microservices and developers must be able to understand how functionality is distributed among them.
-

# Performance

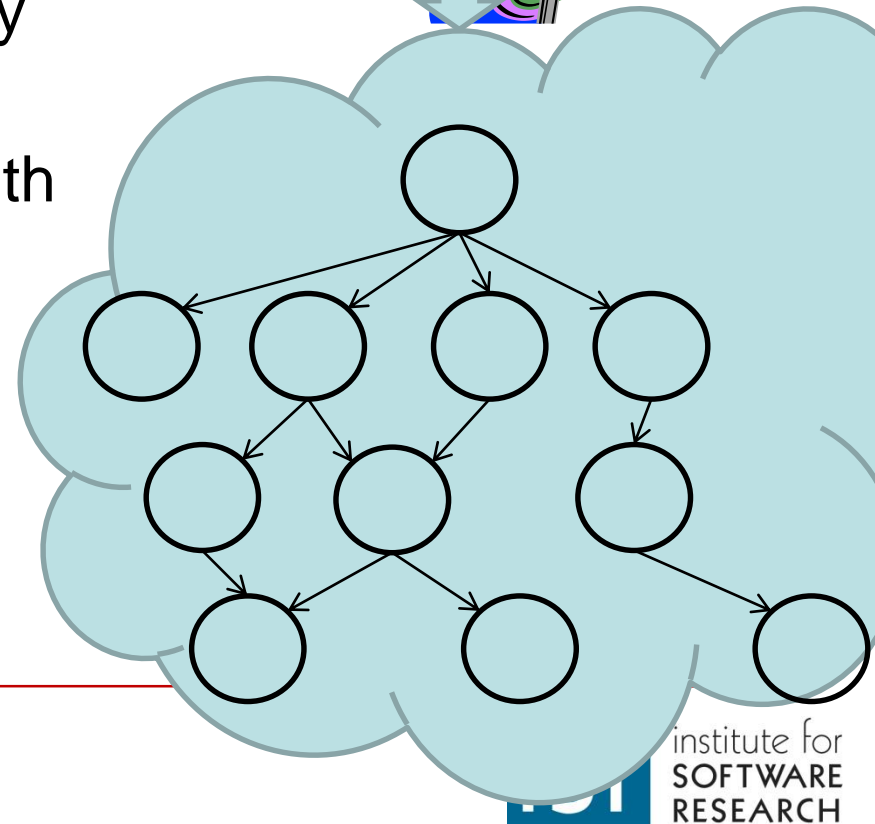
---

- Microservices are based on message passing.
- Network communication takes time.
- Important to measure and monitor microservices to determine where time is spent so that performance objectives can be met.
- Pods are means for improving performance

# Reusability

- Each user request is satisfied by some sequence of services.
- Most services are not externally available.
- Each service communicates with other services through service interfaces.
- Service depth may
  - Shallow (large fan out)
  - Deep (small fan out, more dependent services)

○  
Service



# Distinguish between coarse grain and fine grain reuse

---

- Large microservices (coarse grained) are built into architecture and reuse is achieved by using these microservices
- Small microservices (fine grained) can be designed for reuse (deep service hierarchy) or reuse can be ignored at this level (shallow service hierarchy)

# Trade off

---

- Shallow hierarchy
    - higher performance since fewer messages,
    - low reuse since each service is developed by independent team with little coordination
  - Deep hierarchy
    - Lower performance since more messages
    - Higher reuse since services can be designed for multiple different clients.
-

# Summary

---

- Microservice architecture consists of collection of independently deployable services
- Microservices supports devops processes by reducing need for communication among teams and making technology choices independent
- Microservice architecture favors modifiability over performance.
- Reuse can be achieved, if desired, by having deep service hierarchy.



# Deployment and Operations for Software Engineers

Chapter 6 Microservice Architecture - 2



# Overview

---

- **Discovery**
- Making requests in a distributed system
- Microservices and Containers

# Discovery

---

- Discovery is the process of finding an IP address of an instance of a microservice.
- Isn't that what DNS and load balancers are for?
  - Yes, but ...

# Why not DNS + Load Balancer

---

- DNS assumes static IP addresses. So IP addresses of load balancer is used
- Load balancer does not know the name of the microservice it manages
- DNS + Load Balancer adds overhead.
- Recall that microservices inherently have overhead from messages.
- Combining DNS and Load Balancer functions in one place will reduce overhead.

# Mid tier discovery service

---

- A mid tier discovery service is accessed with a microservice name and returns an IP address of an instance of that microservice
- The discovery service can be made version aware to support deployment options (discussed later)

# Overview

---

- Discovery
- **Making requests in a distributed system**
- Microservices and Containers

# Communication between two microservices

---

- RPC (Remote Procedure Request).
- REST (REpresentational State Transfer)

# RPC

---

- RPC is a remote analog to traditional procedure calls.
  - One procedure calls another with a set of typed arguments, control is transferred to called procedure, and called procedure may respond by returning control
  - Synchronous although multiple threads allow for multiple simultaneous RPCs
- Assumes calling procedure knows address of called procedure (no discovery)

# gRPC

---

- gRPC is a binary version of RPC
- Used in conjunction with protocol buffers (discussed shortly)
- Builds on HTTP 2.0
- Typically asynchronous



# REST

---

- Can be used between arbitrary services on the internet
- Designed for web services

# REST Characteristics

---

- The requests are stateless. Every request must contain all the information necessary to act on that operation,
  - The information exchanged is textual – services and methods are accessed by name.
  - REST restricts methods to PUT, GET, POST, and DELETE.
-

# Structuring Data

---

- XML
- JSON
- Protocol Buffers

# XML (Extensible Markup Language)

---

- XML = textual document + schema
- The schema provides tags to describe how to interpret the document
- Tags are used to enclose fields just as in HTML.

# JSON (JavaScript Object Notation)

---

- Textual
- Each data item is designated by a name and a value. E.g.

*"addressCountry": "United States",*

# Protocol Buffers - 1

---

- Schema defines data types
- Binary format
- A protocol buffer specification is used to specify an interface
- Language specific compilers used for each side of an interface
- Allows different languages to communicate across a message based interface

# Protocol Buffers – 2

---

- Service A written in Java calls Service B written in C
- Interface specification written as .proto file
- Java protocol buffer compiler produces Java procedure interface for Service A
- C protocol buffer compiler produces procedure interface for Service B
- Service A code calls Java procedure interface which sends binary data received by Service B procedure (written in C)

# Overview

---

- Discovery
- Making requests in a distributed system
- Microservices and Containers



# Microservices and Containers

---

- Although microservices and containers were developed independently, they are a natural fit and are evolving together.
- A microservice will use a number of dependent services. Common ones are:
  - Metrics
  - Logging
  - Tracing
  - Messaging
    - gRPC
    - Protocol buffers
  - Discovery
  - Registration
  - Configuration management
  - Dashboards
  - Alerts

# Packaging microservices

---

- Each dependent service will be packaged in its own containers.
- Containers can be grouped in pods
- Also called service mesh
- Allows for deploying and scaling together

# Designing for Deployment

---

- It is possible that different versions of a single microservice may simultaneously be in service
- It is possible that new features may be supported in some microservices but not in others.

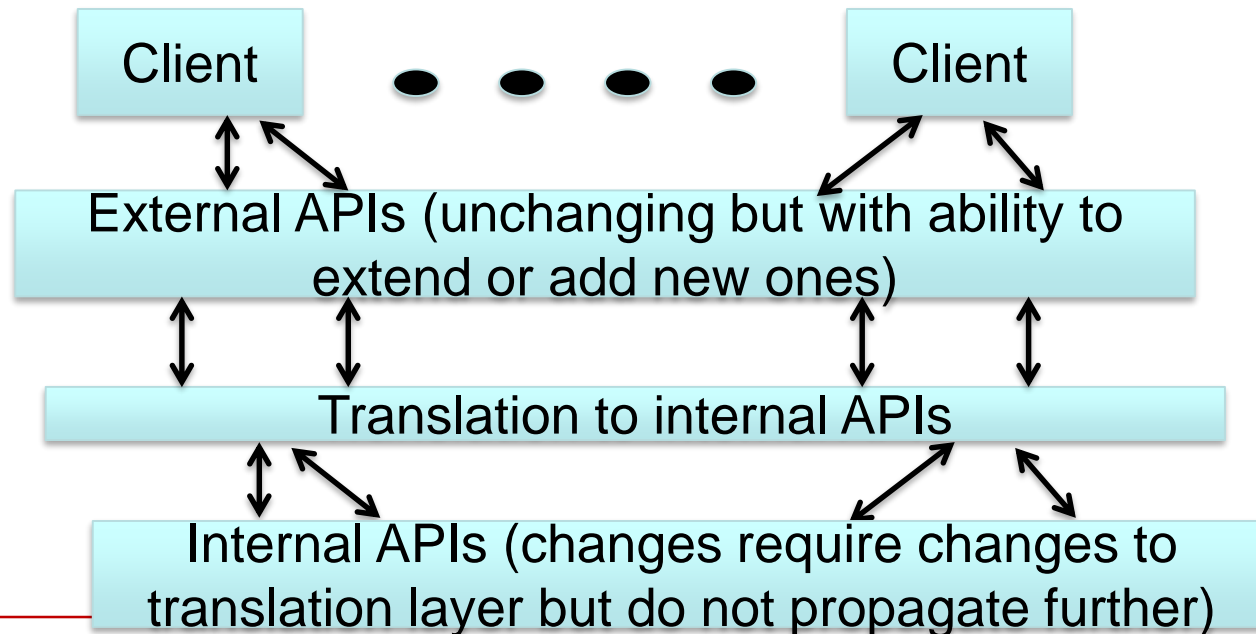
# Forward and Backward compability

---

- Microservices must be forward and backward compatible to support interoperability
  - Forward compatibility means that unknown calls are treated gracefully since one service may be assuming dependent services have features that have not yet been deployed.
-

# Achieving Backwards Compatibility

- APIs can be extended but must always be backward compatible.
- Leads to a translation layer



# Summary

---

- Mid tier discovery services combine functions of DNS and load balancer
- RPC variants and REST are two methods for communicating between services
- XML, JSON, Protocol Buffers are different ways of managing data sent between two services.
- Microservices have a standard set of dependent services and pods allow packing these services together for deployment and scaling.
- Deployment options introduce requirements to design microservices for deployment



# Deployment and Operations for Software Engineers

Chapter 7 Configuration management

# Overview

---

- Version Control
- Configuration Management Tools
- Configuration Parameters
- Managing Secrets – 1



# Version Control Systems (VCS)

---

- Maintains textual information
- Shared among team members
- Centralized or distributed
- Three functions
  1. A check-out/check-in process
  2. A branch/merge process
  3. Tagging or labeling versions

# Centralized version control system

---

- A centralized VCS has a central repository.
  - Users must connect to that central repository to check in or check out files. I.e. internet connection is required.
  - System can maintain knowledge of who is working on which files.
    - Allows informing team members if another member checks out a file
    - Allows locking of files or locking of check in.
  - Subversion is common centralized VCS.
-

# Distributed VCS

---

- A distributed VCS also has a central repository but interactions are different from centralized VCS
- User gets a copy of the repository for local machine
- Check in/check out of a file is from local copy.
- Does not require internet connection to check out
- System has no knowledge of which team member is working on which file.
- Git is common distributed VCS.

# Check in/Check out

---

- Check out results in a local copy
- User can modify local copy.
- Check in copies it back to repository – local or central.
- New version gets new number.
- VCS can perform style checks on check-in

# Branch process

---

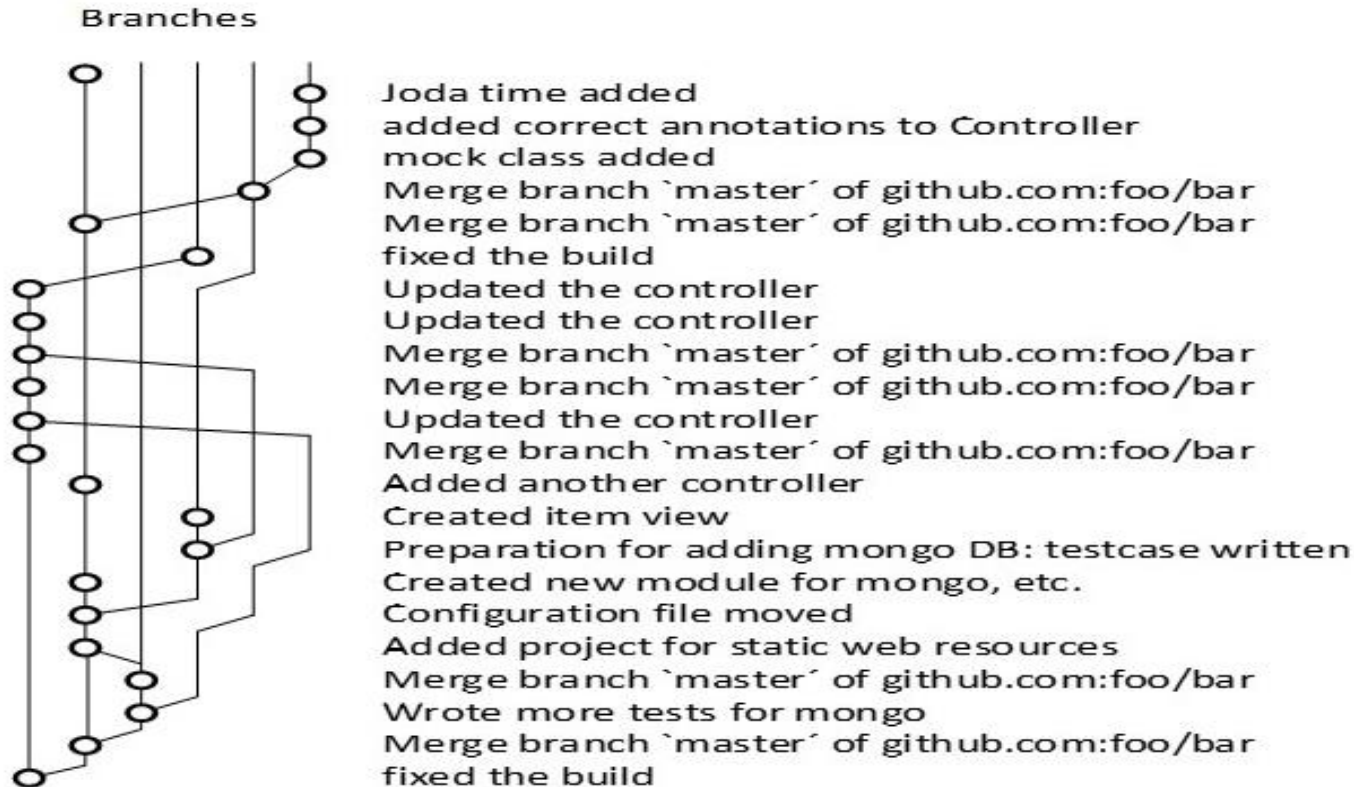
- Repository is structured as a tree with branches
- Files exist on a branch
- New branches can be created with a duplicate set of files. Allows for different tasks on same file. E.g. adding features and performing bug fixes. These actions might be done on different branches.
- Each branch has a purpose and the files on one branch are isolated from the files on a different branch.

# Merge process

---

- Two branches can be merged.
- Differences in two versions of the same file must be resolved.
- Best practice is to merge frequently since reduces number of differences that must be resolved.

# Sample branch structure



# Best practices for version control

---

- Use a descriptive commit message
  - Make each commit a logical unit
  - Avoid indiscriminate commits
  - Incorporate others' changes frequently
  - Share your changes frequently
  - Coordinate with your co-workers
  - Remember that the tools are line-based
  - Don't commit generated files
-



# Overview

---

- Version Control
- Configuration Management Tools
- Configuration Parameters
- Managing Secrets – 1

# Consistency is important

---

- Consistency on platforms reduces error possibilities.
    - Different team members should develop on the same platform
    - Development platform should be consistent with the production platform
    - Updates should be applied to all machines in a fleet automatically
    - Replicas across data centers should have same software installed down to version and patch numbers.
-

# Configuration Management (CM) Systems

---

- The purpose of a configuration management system is to maintain consistency across a set of machines.
- Common tools are:
  - Chef
  - Puppet
  - Ansible

# Actions of a CM tool

---

- A CM tool resides on a server
- Controls collection of other machines – virtual or physical
  - Identified by IP addresses and grouped by functions. E.g. Web server, Development platform, DB server
  - Controls them through SSH from CM server to clients
- Extensible through plug-ins

# Specification

---

- Specification of actions is done through scripting language
- E.g. “make sure all web servers have latest patch for nginx 15.8”
- Specification should be version controlled.

# Overview

---

- Version Control
- Configuration Management Tools
- Configuration Parameters
- Managing Secrets – 1

# Configuration parameters

---

- A configuration parameter is a parameter set by system administrator
- . E.g., logging level, DB URL, background color
- As service moves from development to production, different values will be used for configuration parameters.

# Management of configuration parameters

---

- Incorrect parameters a major source of errors
  - Libraries exist for some types of checking. E.g.
    - Values have been specified
    - Range of value is correct
  - URLs for external entities should be checked during startup to ensure
    - They are accessible from current environment
    - They are of the correct type.
  - Need to worry about maintaining confidentiality of secret configuration parameters – will discuss later
-



# Getting parameter values to application

---

- Resource file. Read by app, typically on initialization
- Environment variables. Set in operating system and used by name in app.
- Database. Read by app.
- Specialized tool. Used prior to invocation to set parameter values.

# What parameters should be configuration parameters?

---

- Developers decide which parameters are configuration parameters.
- Form of late binding.
- Any parameter can be made a configuration parameter
  - Must have default value if not set
  - Value must be checked for reasonableness
  - Possible to have too many configuration parameters, e.g. >10,000 in some systems.

# Overview

---

- Version Control
- Configuration Management Tools
- Configuration Parameters
- Managing Secrets – 1

# Credentials as configuration parameters?

---

- It is tempting to make credentials, e.g. passwords, keys, be configuration parameters
  - Making a credential a configuration parameters will expose secrets too broadly.
  - Configuration parameters show up in textual scripts. Means anyone with access to script has access to credentials. May be stored in public repository such as Github.
-

# Managing credentials

---

- Credentials should be treated differently from other parameters.
- We discuss treatment of credentials when we return to security in a later lecture.

# Summary

---

- Version control systems enable controlled sharing and modification
  - Configuration management systems maintain consistency of software and versions across classes of machines
  - Configuration parameters are a form of deferred binding.
  - Credentials should not be made configuration parameters.
-



# Deployment and Operations for Software Engineers

Chapter 8 Pipeline - 1

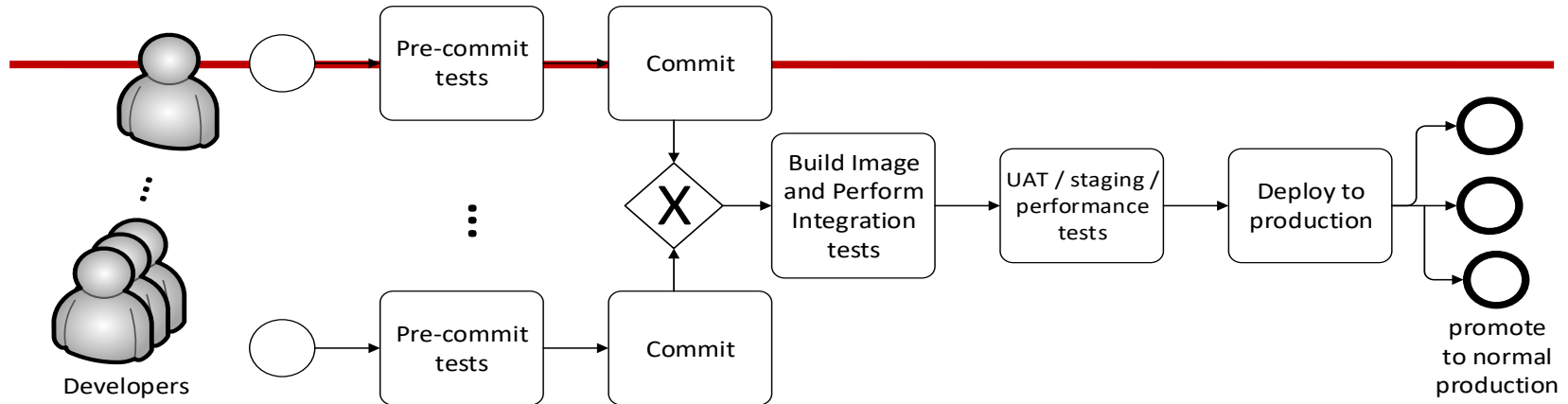
# Overview

---

- Introduction to a deployment pipeline
- Environments

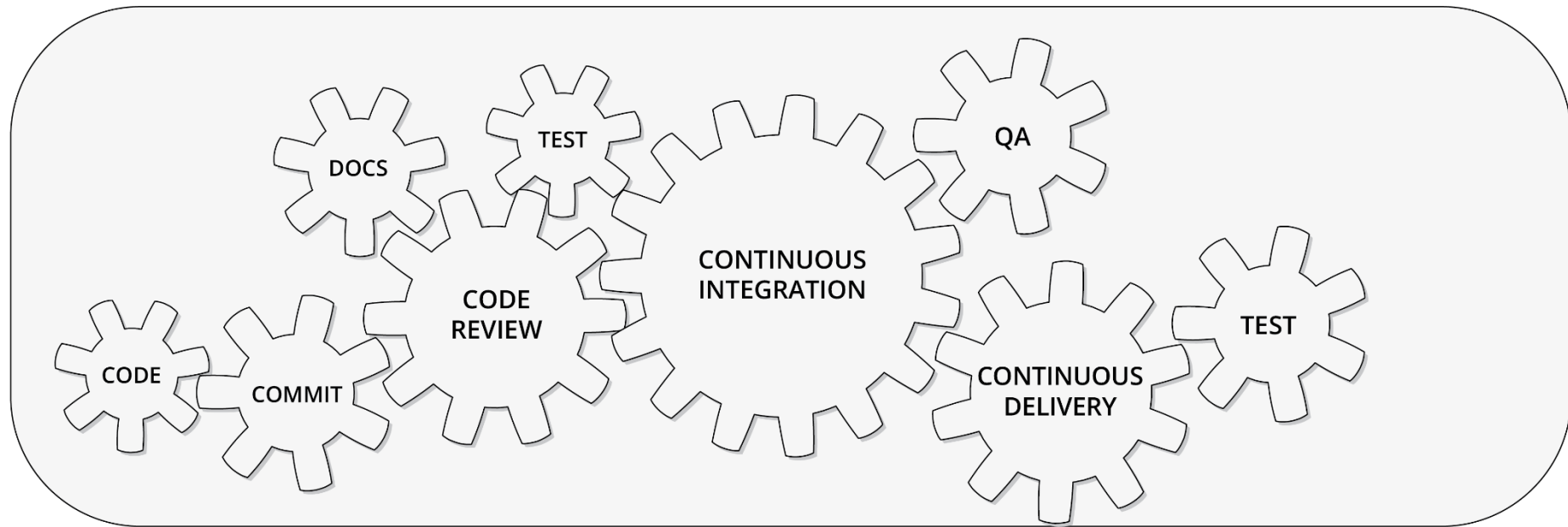


# Deployment pipeline



- Developer creates and tests code on local machine.
- Checks code into a version control system
- Continuous integration server (CI) builds the system and runs a series of integration tests.
- After passing the tests, the system is promoted to a staging environment where it undergoes more tests including performance, security, and user acceptance tests.
- After passing those tests, the system is promoted to provisional production where it undergoes even more tests.
- The system is finally promoted to normal production but the tests do not necessarily stop.

# Another view of the pipeline



# Goals during deployment pipeline

---

- Team members can work on different versions of the system concurrently
  - Code developed by one team member does not overwrite code developed by others
  - Code produced by one team can be integrated with code produced by other teams
  - Code is the same during different stages
  - Different stages serve different testing purposes
  - Different stages are isolated from each other
  - A deployment can be easily rolled back if there is a problem.
-

# Desirable qualities of deployment pipeline

---

- Traceability
- Testability
- Tooling
- Cycle time

# Traceability

---

When any code gets into production, it must be reconstructable

- All components that go into the executable image can be identified

- All tests that were run can be identified

- All scripts used during the pipeline process can be identified.

When a problem occurs in the system when it goes into production, all of the elements that contributed to the system can be traced.

# Testing

- 
- Every stage (except production) has an associated test harness.
  - Tests should be automated.
  - Types of tests are
    - Sunny day tests
    - Negative tests
    - Regression tests
    - Static analysis
  - We will discuss stage specific tests when we discuss the stage.
-

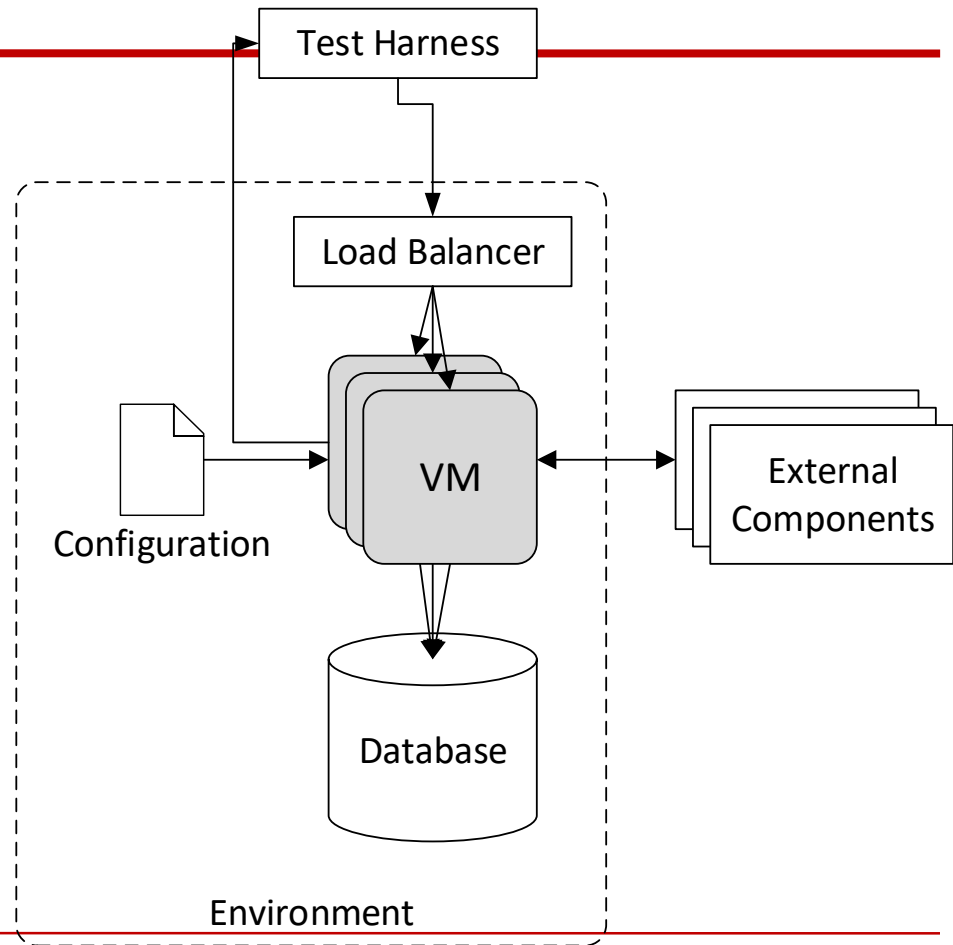
# Database test data

---

- Every stage has a database
- Minimal during development
- More substantial during build
- Realistic during staging
- After each test, database must be reconstituted to ensure repeatability
- Personally Identifiable Information (PII) must be obscured.

# Test Harness

A test harness generates inputs and compares outputs to verify the correctness of the code.



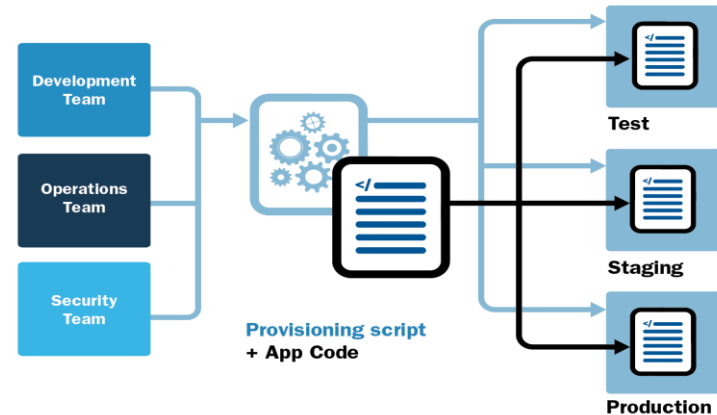


# Tooling

- 
- Version control system – manages different versions for development and production – e.g. Git
  - Configuration management tool – controls configuration of deployed image – e.g. Chef, Puppet, or Ansible
  - Continuous Integration tool – builds image and runs test – e.g. Jenkins
  - Deployment tool – places images into production. E.g. Spinnaker, Ansible, Chef
  - These tools must execute on a platform
  - ~~Vagrant is a tool to help provision the platform~~
-

# Scripts for pipeline tools

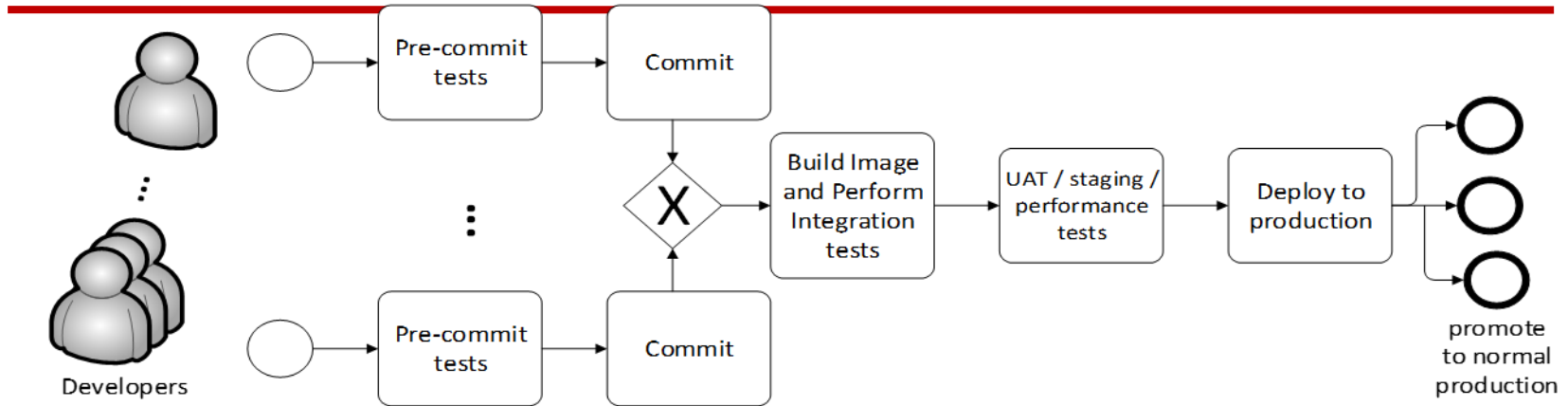
Scripts for pipeline tools are an example of infrastructure as code (IaC). They should be version controlled and tested just like application code.



# Cycle time

- 
- Cycle time is the time between the commit and the placing into provisional production.
  - Systems should move through the pipeline quickly
  - The time depends on
    - How large are the components that are constructed in the build stage
    - How long does it take to run tests on the system

# Different environments in pipeline



- Development environment
- Build (also called Integration) environment
- Staging environment
- Production environment
- Different organizations may have additional environments defined

# Sample pipeline

- 
- A sample pipeline without staging environment can be found at

<https://github.com/cmudevops/class-materials/blob/master/deployment%20workflow.pdf>

# Overview

---

- Introduction to a deployment pipeline
- Environments

# Environment

- 
- An environment is a set of computing resources sufficient to execute a software system including all of the supporting software, data sets, network communications, and necessary external entities.
  - Essentially, an environment is self contained except for explicitly defined external entities.
  - Multiple different environments in a pipeline.

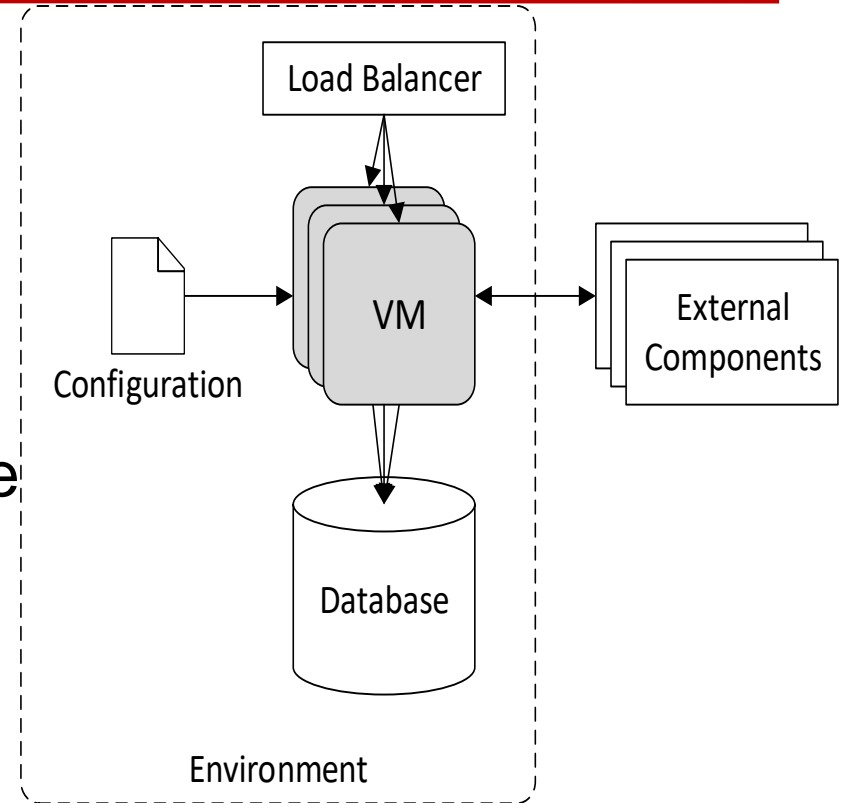
# Isolating environments

- 
- Environments should run same code but should be isolated from each other.
  - One solution is to connect the code to all other elements of the environment through configuration parameters.
  - During initialization of the code, it acquires and makes operational all of its configuration parameters.
  - Elevates setting and management of configuration parameters to important element of deployment.



# Environment is isolated

- Any references to database, external entities, network is controlled by configuration parameters.
- As long as configuration parameters are correct, there is no way that the VMs can access other environments.



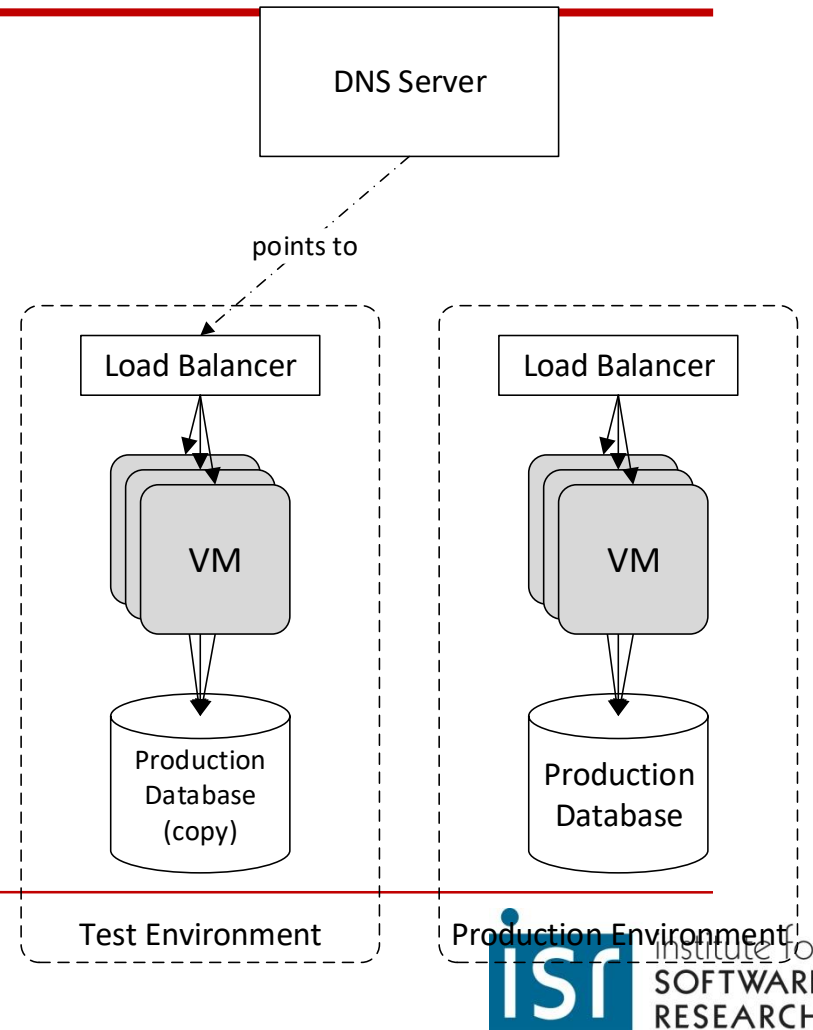
# Life Cycle of an environment

---

- Each environment has a life cycle. When is it created, how is it used, when is it destroyed.
- Having a defined life cycle allows scripting to control moving from one phase of the life cycle to the next.
- One organization, for example, creates a new environment whenever the version control tree has a branch. So branching the version control system triggers a script to build the environment.

# Moving a system from one environment to another

- Messages are routed through DNS server
- Routing messages from one environment to another becomes partially a matter of changing the DNS server.
- Messages can come from test harness or from users or from production portions of the system



# Routing of messages does not totally solve movement problem

---

- Dependencies on particular environment cause problems
  - The version of software used by the component may cause problems
  - Data in database for testing may cause problems
  - Other types of dependencies
    - OS version dependencies
    - Library versions

# Are OS dependencies really a problem?

Linux kernel  
changes –  
x86\_64

Version	Added Symbols	Removed Symbols	Total Changes
3.15-rc6	<u>387</u>	<u>194</u>	<u>1120</u>
3.14.4	<u>363</u>	<u>232</u>	<u>981</u>
3.13.11	<u>317</u>	<u>149</u>	<u>842</u>
3.12.20	<u>436</u>	<u>141</u>	<u>907</u>
3.11.8	<u>333</u>	<u>109</u>	<u>1514</u>
3.10.19	<u>726</u>	<u>200</u>	<u>2541</u>
3.9.11	<u>2102</u>	<u>859</u>	<u>6746</u>
3.4.69	<u>771</u>	<u>324</u>	<u>3665</u>
3.2.52	<u>474</u>	<u>128</u>	<u>3263</u>
3.0.101	<u>1860</u>	<u>863</u>	<u>8054</u>
2.6.34.14	<u>323</u>	<u>201</u>	<u>1261</u>
2.6.33.20	<u>401</u>	<u>185</u>	<u>1317</u>
2.6.32.61	initial		

# Two paths to support movement of system

---

- **Manage relationship of component to other software in its environment**
- Turn other dependencies into configuration parameters and manage them
- **Both must be done**

# Managing relationship of component to other software

---

- Two basic strategies
  - Ensure software in each environment is identical
  - Move environment (or portion of the environment) together with the component

# Ensure software in each environment is identical

---

- Do development in environment that is the same as staging and production environments, down to version numbers.
- Requires that every development team develop on same environment – no version changes without all teams agreeing



# Move environment together with component

---

- Move total environment
  - Create a virtual machine or container with all of the dependent software
  - Virtual machine or container is tested in staging and moved to production

# Two paths to support movement of system

---

- Manage relationship of component to other software in its environment
- **Turn other dependencies into configuration parameters and manage them**
- **Both must be done**

# Summary - 1

- 
- A deployment pipeline is a series of stages where each stage has
    - A specific testing purpose
    - An isolated environment
  - Each stage in the pipeline is controlled by some set of tools.
    - Scripts for these tools should be version controlled and tested.
  - The pipeline has a set of desirable properties that can be used to measure it

# Summary - 2

- 
- Environments are isolated from each other
  - As systems move through the pipeline they
    - Move into environments that are kept identical or
    - Bring their environment with them
  - Configuration parameters are used to manage differences among environments



# Deployment and Operations for Software Engineers

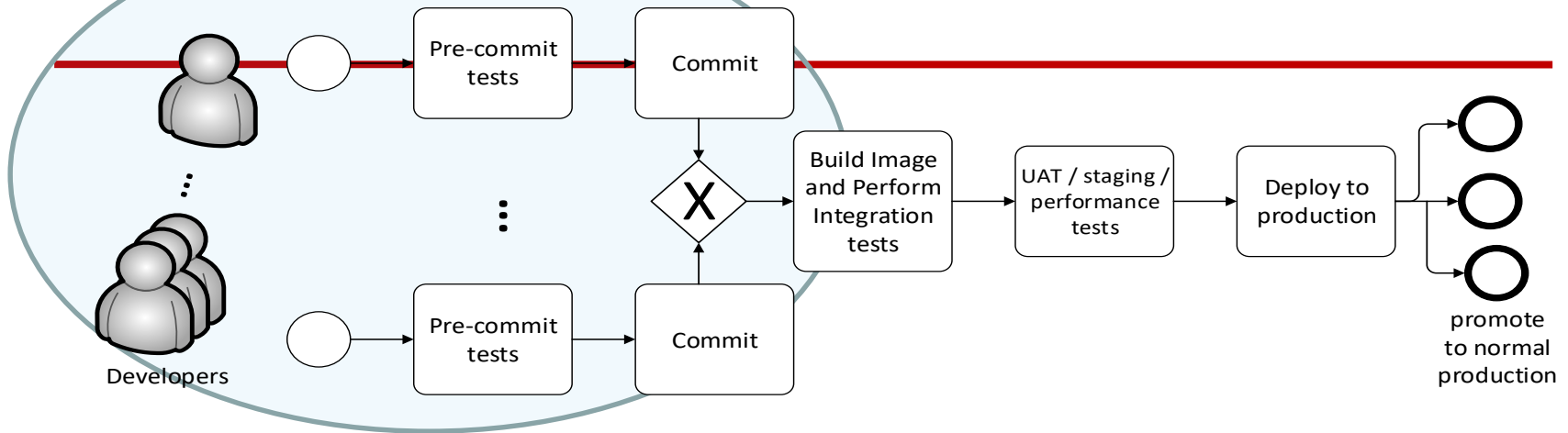
Chapter 8 Pipeline - 2

# Overview

---

- Development environment
- Build environment

# Development environment



- Developer creates and tests code on local machine.
  - Version control
  - Uniform development environment across developers
  - Pre-commit unit tests
  - Feature toggles

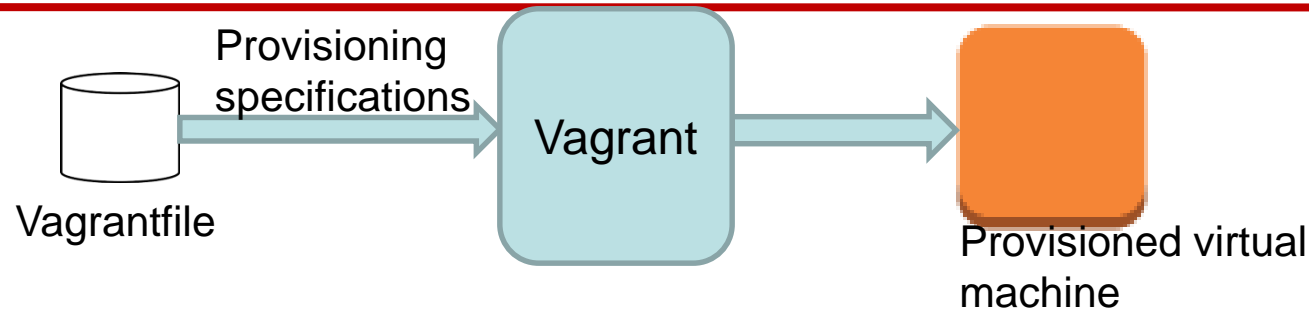
# Uniform development environment

---

- Goal:
  - Developers all have the same development environment
  - Developers have the same development environment as the production environment
- Reduces errors caused by incompatibilities in versions/dependencies

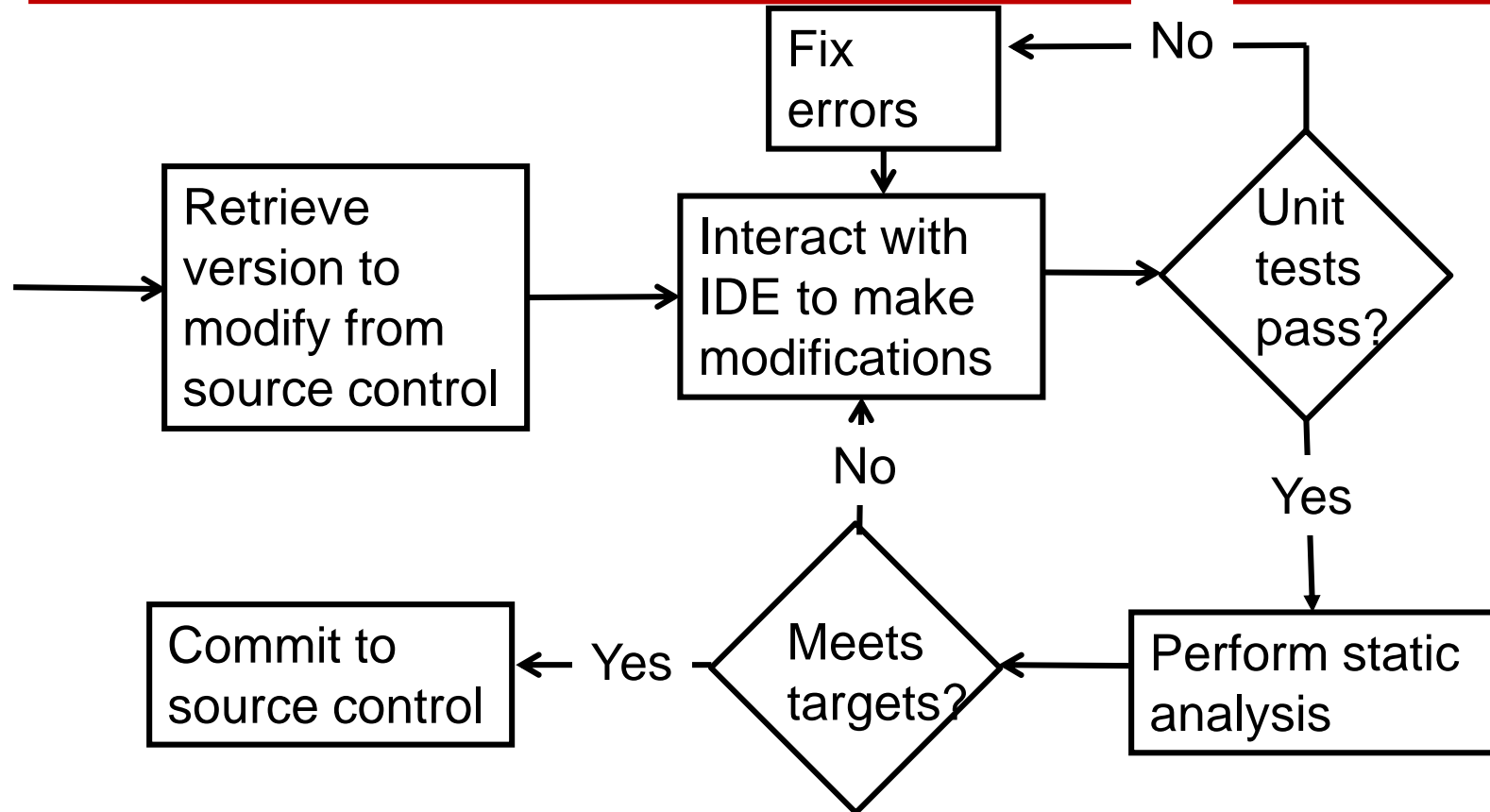


# Vagrant



- Vagrant takes a specification for a provisioned VM and produces such a VM on a provider such as VirtualBox
- Specification is version controlled and shared among team members through version control system
- Result is a common environment for development and production

# Workflow during development



# Pre commit testing

---

- Tests run on a single module or class.
  - Dependent services are stubbed out or mocked
  - Very limited test database.
- Test driven development. Write the tests before writing the code.
- Unit tests. Specialized tests for single modules.

# Testing in Development Environment

---

- Unit Testing/component testing
    - Verify the functionality of specific section of code
    - Includes static code analyzers, data flow, metrics analysis or peer code reviews
  - Acceptance Testing
    - Done by developer prior to integration or regression
  - Smoke/Sanity Testing
    - Reasonable to proceed with further testing or not
    - Minimal attempts to operate software
-

# Security Requirements

---

- Security Requirements (SFR/SAR)
- Risk Assessment
- Abuse Case Development
- Threat Modelling
- Security Stories
- Screen Development Tools
- Secure/Hardened Environments

# Security processes during design

---

- Threat Modelling
- Vulnerability Analysis and Flow Hypothesis
- Security Design Review
- Dependencies List, Open-source libraries

# Security practices during development

---

- Secure Coding Practices
- Security Focused Code Review
- Deprecate Unsafe Functions
- Perform Security Unit Testing
- Static Code Analysis
- Checking of process and procedures for secure coding & traceability

# Test tools-Unit Testing

---

- Junit : Testing framework for the Java/J2EE
- Nunit: Testing framework with GUI for .Net platform
- MSTest: Testing framework with CL outside Visual Studio
- PYunit: Testing framework with Python
- TestNG: Very similar to Junit but covers more (functional, end-to-end, integration etc.)



# Dealing with partial code

---

- Scenario
    - Begin changing a particular module
    - Bug discovered in production version of that module
    - Fix bug
    - Check in
    - Error because new changes are not complete
  - Fix – put new code inside of a “feature toggle”. It still must compile but it does not have to execute correctly.
-

# Feature Toggle

- 
- A feature toggle is one kind of configuration parameter. Later it will be bound at run time, for this use it is development time

```
If (feature_toggle) then
    New code
else
    Old code
end;
```

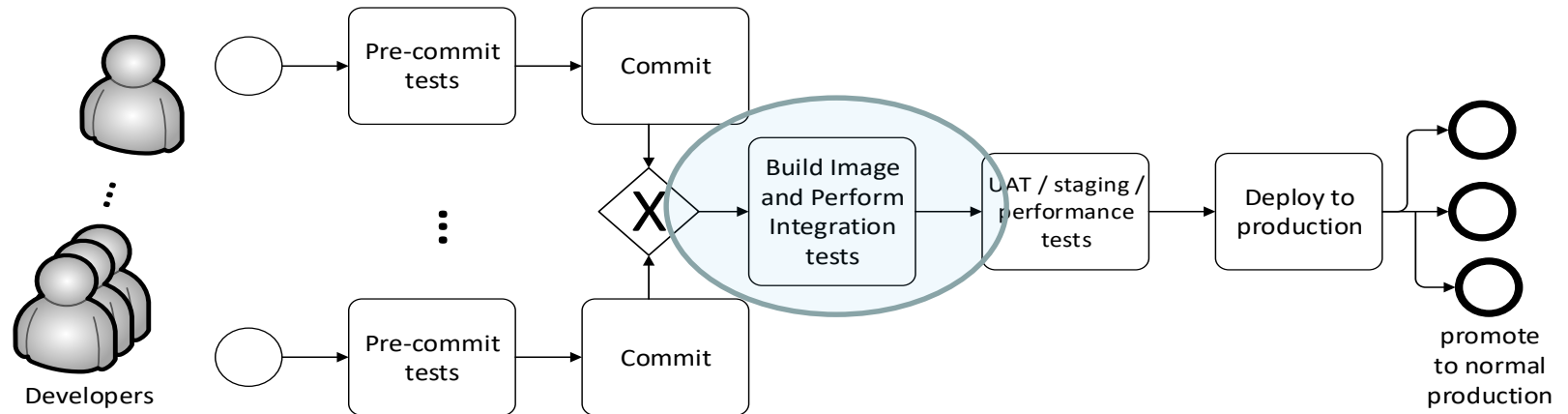
- Feature toggles have other uses (later)
- Feature toggles should be removed when no longer needed
  - They clutter up code
  - They were one source of Knight Capital meltdown (\$440 million lost in 45 minutes)

# Overview

---

- Development environment
- Build environment

# Build Environment



- Create executable image
- Perform functional tests

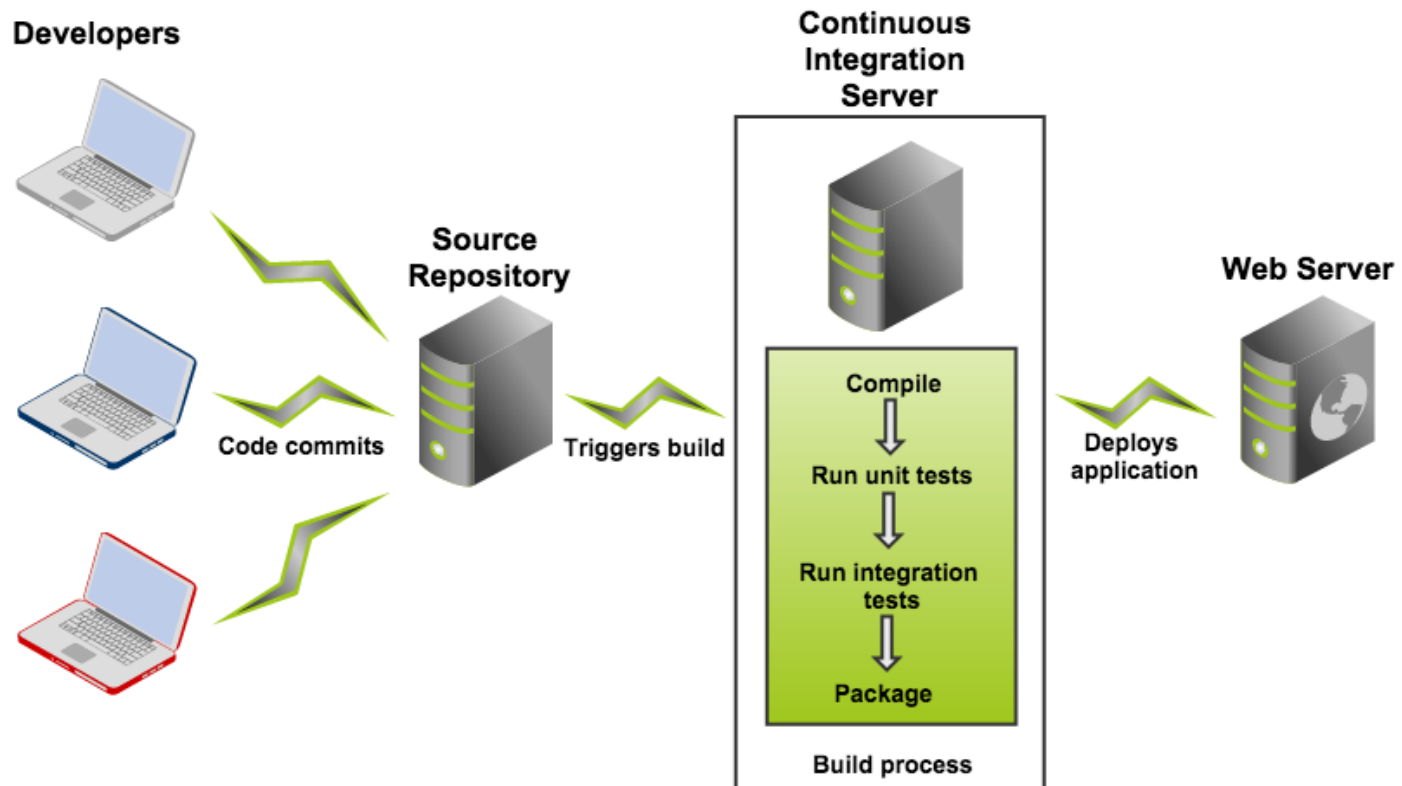
# Purpose of Build Stage

---

- Create executable image
    - Locating and compiling (if necessary) all of the elements of the executable image
    - Linking these elements together
  - Perform functional tests
    - Test integrated (whole) system
    - Automated
    - Limited amount of data in database but as real as possible
  - Same suite of test generators as during development
-

# Overview from Atlassian

## Continuous Integration



# Creating an executable image

- 
- Continuous Integration (CI) server is notified of new commits (or polls for them)
  - When a new commit is detected, the CI server retrieves it
  - The CI server retrieves all relevant code and their dependencies
  - Executable image is created by compiling and linking all of the code and their dependencies.

# Potential errors during build

---

- Code does not compile
- Dependencies do not exist
- Provided interfaces do not match required interfaces
- Configuration parameters do not exist or are inconsistent



# Functional testing

- 
- Created image is subjected to a collection of automated tests
  - Tests are performed using a test harness
  - Tests are the result of
    - User stories
    - Regression tests
    - Rainy day scenarios
  - Trade off between comprehensiveness of tests and time it takes to run the tests
  - CI server reports results through e-mail or a web page.
-

# Testing in Continuous Integration Environment

---

- Much like testing in the development environment except with multiple modules rather than a single one
  - Unit Testing/component testing
- Built system testing
  - Detect defects in the interface and interaction between modules
  - Module/subsystem integration

# Test data

---

- Test data is real (ish) but limited.
  - Need to worry about exposing private data to developers
  - Limited so that tests will run fast and keep build queue from growing
  - If tests change database, it must be refreshed before next set of tests.
  - No results should be sent to any production service – results in corrupting production version.

# External services

- 
- The environment includes connections to external services.
    - If the services are read only, they can be directly used
    - If the services are read/write then a test version must be furnished

# Promoting an image

- 
- Once an image is successfully tested it is promoted to the staging environment
  - Successful may not mean passing all of the tests
    - Some failures may stop the process
    - Other failures may just be noted but may allow the process to continue.
    - Success may be “95% of the tests succeed”

# Orchestration

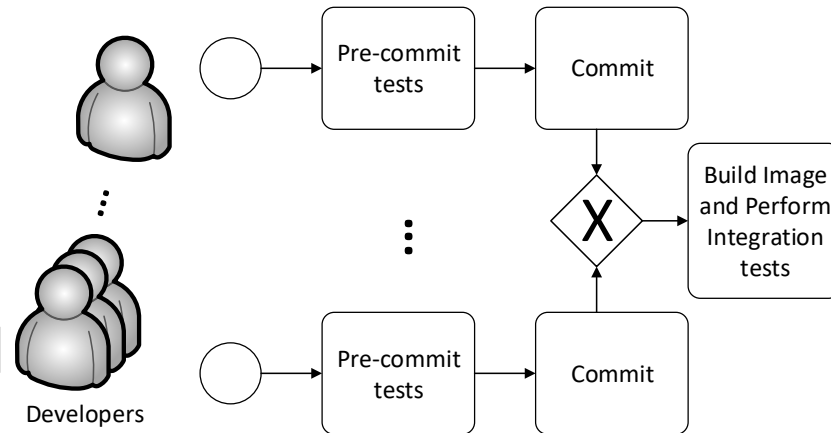
- 
- **Orchestration** (in the context of a build server) is the automated arrangement, coordination, and management of the elements necessary to build, test, report, and migrate an executable image.
  - E.g.
    - Initiate a build upon commit of code
    - If build successful, then initiate test
    - Generate report every evening of daily activity
    - If tests successful promote image to staging

# Build scripts

- 
- Continual Integration servers are programmable
    - Where to find various components and tests
    - Logic to determine validity of components and tests and report results
    - Logic to determine promotion criteria
  - Programs for CI servers are called “scripts”
  - Scripts should be configuration managed and version controlled.

# Build has multiple users

- Build is performed by a continuous integration server (CI server)
- CI is single threaded from perspective of systems (one build of a given system at a time)
- Build error introduced by one developer affects all developers





# Best Practices

- 
- Executable image created during build stage does not change as it is promoted through to production
  - “Breaking the Build” is to be avoided and fixing a broken build is high priority
  - Traceability is maintained for all elements of the executable created.

# Summary

- 
- Every developers environment should be identical and should reflect the production environment
  - Security processes are important to harden code.
  - Pre-commit tests are of a single module.
  - Feature toggles are used in the Dev environment to deactivate code that is not yet ready for other environments
  - Build stage creates an executable impage
  - Build stage tests the executable image for functional correctness
  - Limited data used in the tests
  - CI servers are programmable



# Deployment and Operations for Software Engineers

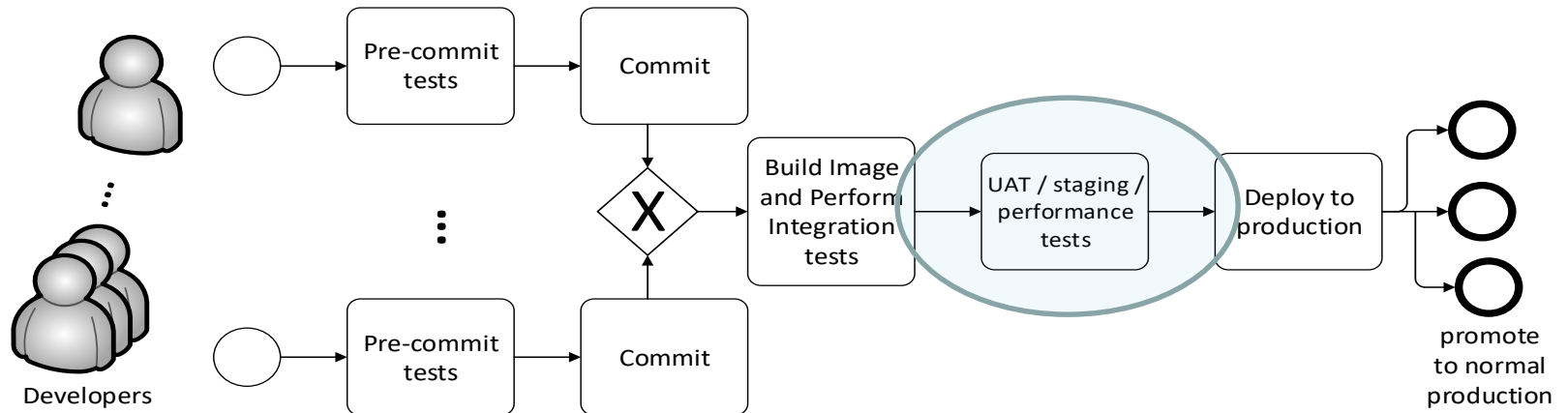
Chapter 8 Pipeline - 3

# Overview

---

- Staging environment
- Deployment strategies

# Staging Environment



- Perform security and load testing
- Sign off

# Testing in Staging Environment

---

- Regression testing
- Smoke testing
- Compatibility testing
- Integration testing
- Functional Testing
- Usability Testing
- Install/uninstall testing
- Performance testing
- Security testing

# User Tests

- 
- Real (or simulated) users exercise the system.
    - Having real users is expensive and difficult to ensure coverage.
      - Record/playback works in some contexts
      - Teeing actual input works in some contexts
    - Having real users allows for exception testing in a better fashion than automated tests

# Performance tests

---

- Test for performance with generated loads
- Environment should be as real as possible
  - Load balanced
  - Auto scaled
  - Multiple instances



# Data base

- 
- Use as close to real data as possible.
    - One organization uses database that is one hour old (created as a back up by cloud vendor).
  - Constraints
    - Restore database after each test
    - Keep personal information private

# Security testing

- 
- Common Vulnerabilities and Exposures data base (CVE)
    - Buffer overflow
    - SQL injection
  - Penetration testing tools
  - Static analyzers
  - Fuzz Testing
  - Model checking

# Do not access production database!

---

- Staging environment must be isolated from production database.
- Accessing production database is one source of errors that came up several times in a survey of operations caused outages.

# Tooling

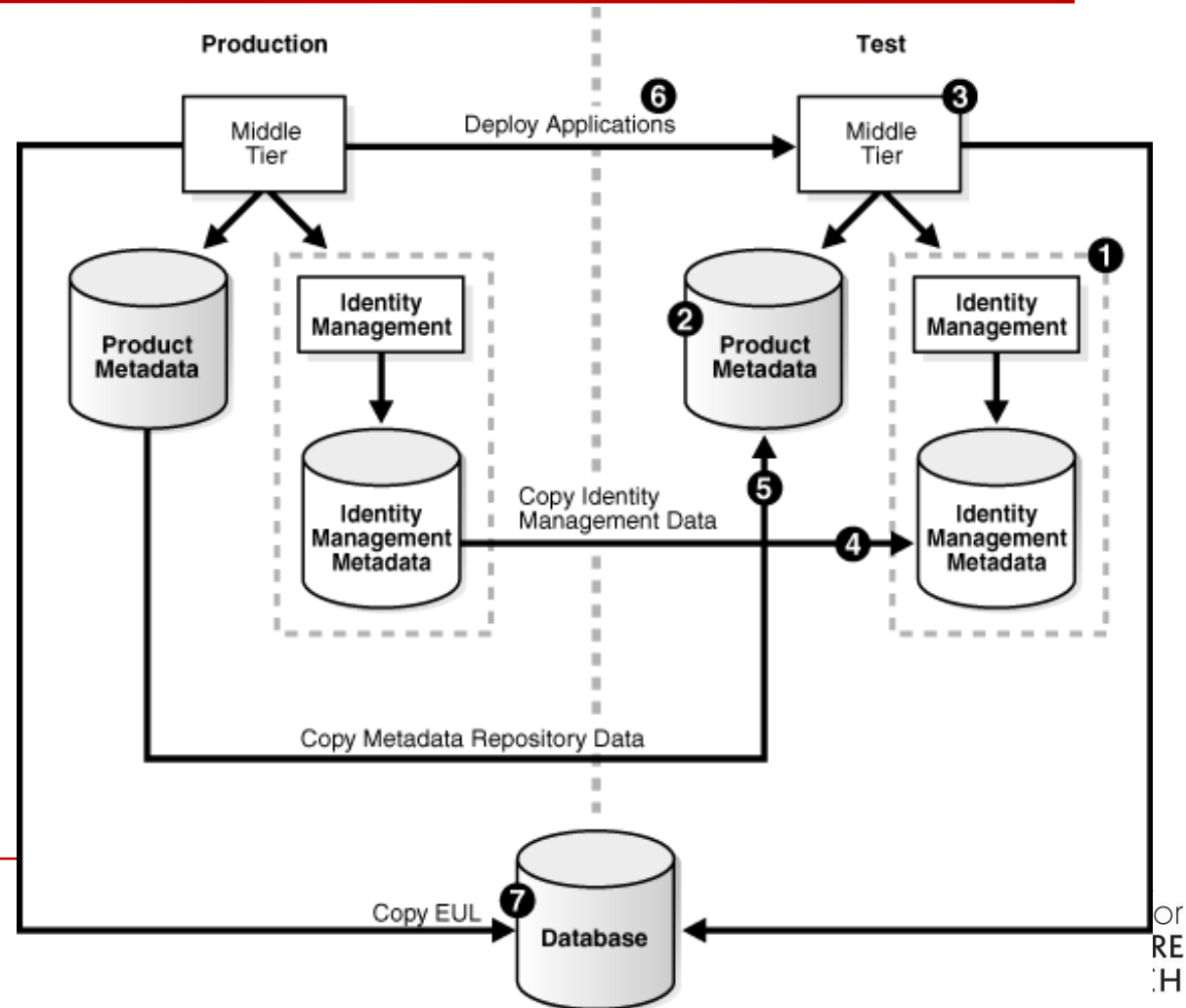
- 
- There are specialized load testing tools, e.g. Artillery
  - There are specialized security testing tools, e.g.
    - OWASP
    - Static analyzers
    - Model checkers

# Creating the staging data base

This is Oracle's recommendation.

EUL is end user layer.

EUL can be used to protect PII although this is not an ideal solution for testing.



# Overview

---

- Staging environment
- Deployment strategies

# Situation

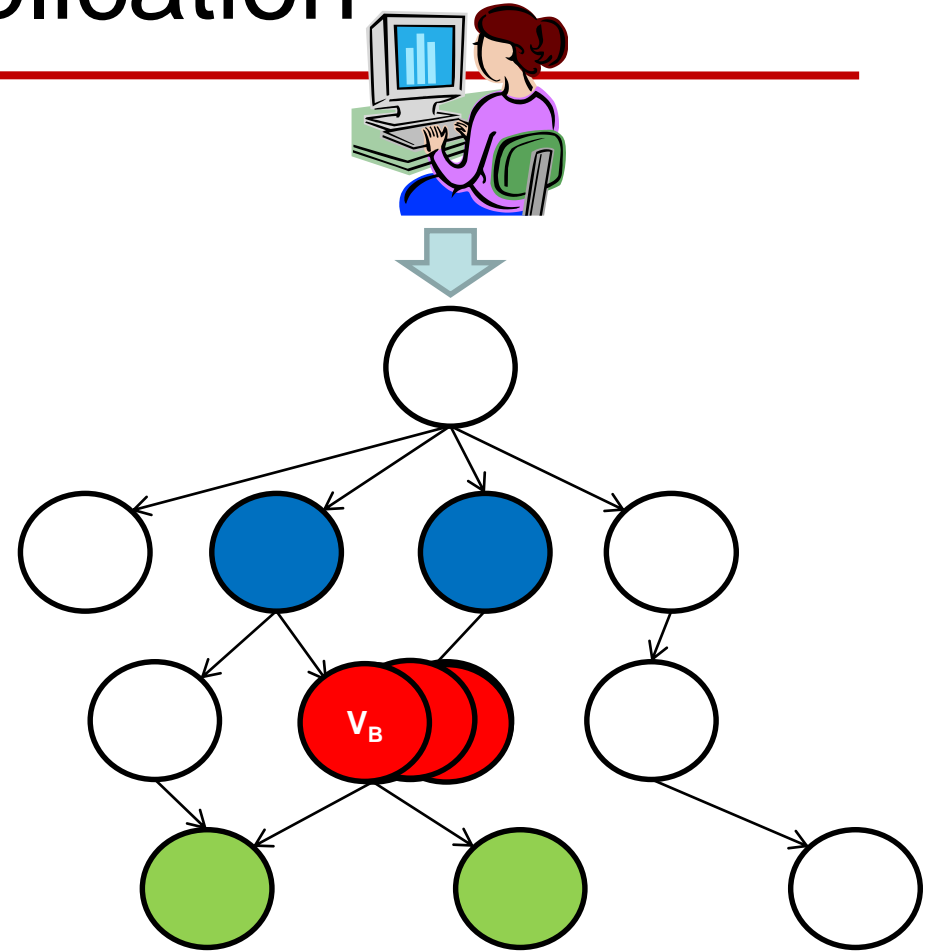
- 
- Your application is executing
    - Multiple independent deployment units
    - Some of these deployment units may have multiple instances serving requests
  - You have a new version of one of the deployment units to be placed into production
  - An image of the new version is on the staging server or in a container repository

# Deploying a new version of an application

Multiple instances of a service are executing

- Red is service being replaced with new version
- Blue are clients
- Green are dependent services

Staging/container repository





# Deployment goal and constraints

- 
- Goal of a deployment is to move from current state (N instances of version A of a service) to a new state (N instances of version B of a service)
  - Constraints:
    - Any development team can deploy their service at any time. I.e. New version of a service can be deployed either before or after a new version of a client. (no synchronization among development teams)
    - It takes time to replace one instance of version A with an instance of version B (order of minutes for VMs)
    - Service to clients must be maintained while the new version is being deployed.

# Deployment strategies

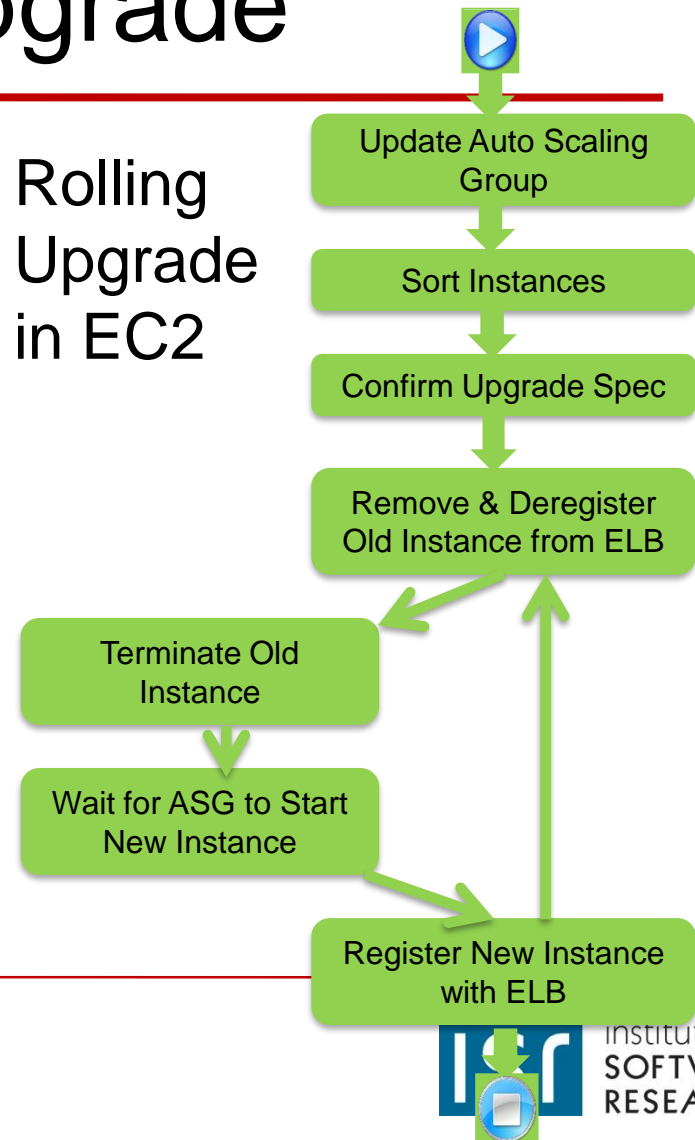
---

- Two basic all of nothing strategies
    - Red/Black (also called Blue/Green) – leave N instances with version A as they are, allocate and provision N instances with version B and then switch to version B and release instances with version A.
    - Rolling Upgrade – allocate one instance, provision it with version B, release one version A instance. Repeat N times.
  - Partial strategies are canary testing and A/B testing.
-

# Trade offs – Red/Black and Rolling Upgrade

- Red/Black
  - Only one version available to the client at any particular time.
  - Requires  $2N$  instances (additional costs)
- Rolling Upgrade
  - Multiple versions are available for service at the same time
  - Requires  $N+1$  instances.
- Rolling upgrade is widely used.

## Rolling Upgrade in EC2



# Problems to be dealt with

---

- Temporal inconsistency
- Interface mismatch
- Data inconsistency

# Temporal inconsistency example

---

- Shopping cart example
  - Suppose your organization changes its discount strategy from discount per item to discount per shopping cart.
  - Version A' of your service does discounts per item
  - Version A'' does discounts per shopping cart.
- Client C's first call goes to version A' and its second call goes to version A''.
- Results in inconsistent discounts being calculated.
- ~~Caused by update occurring between call 1 and call~~

# Temporal inconsistency

---

- Can occur with either Blue/Green or rolling upgrade
- Prevented by using feature toggles.
- Remember feature toggles?

# Preventing Temporal Inconsistency

---

- Write new code for Service A'' under control of a feature toggle
- Install N instances of Service A'' using either Rolling Upgrade or Blue/Green
- When a new instance is installed begin sending requests to it
  - No temporal inconsistency, as the new code is toggled off.
- When all instances of Service A are running Service A'', activate the new code using the feature toggle.

# Feature toggle manager

---

- There will be many different feature toggles
    - One for each feature
  - A feature toggle manager maintains a catalog of feature toggles
    - Current toggles vs instance version id
    - Current toggles vs module version
    - Status of each toggle
    - Activate/de-activate feature
    - Remove toggle (will place removal on backlog of appropriate development team).
-



# Activating feature

- 
- The feature toggle manager changes the value of the feature toggle.
  - A coordination mechanism such as Zookeeper or Consul could be used to synchronize the activation.

# Interface mismatch

---

- Suppose version A'' has a different interface from version A'
  - Then if Service C calls version A'' with an Interface designed for version A' an interface mismatch occurs.
  - Recall that Service A can be upgraded either before or after Service C.
  - Interface mismatch is prevented by making interfaces backward compatible.
-

# Two types of data consistency problems during upgrade

---

1. Persistent data
2. Transient data

# Maintaining consistency between a service and persistent data

- 
- Assume new version is correct
  - Inconsistency in persistent data can come about because data schema or semantics change from one version to the next
  - Effect can be minimized by the following practices (if possible).
    - Only extend schema – do not change semantics of existing fields. This preserves backwards compatibility.
    - Treat schema modifications as features to be toggled. This maintains consistency among various services that access data.

# I *really* must change the schema

- 
- In this case, apply pattern for backward compatibility of interfaces to schema evolution.

# Transient data

---

- An instance of a service may be maintaining transient data for some purpose
    - Caching for performance purposes
    - Maintaining session state
  - New instance may need access to this transient data – whether new version or instance of old version.
-

# Solution

- 
- Use coordination manager to maintain transient data
  - Ensure there is always at least one instance of a service that uses the coordination manager—otherwise data stored in coordination manager might be lost.

# Rollback

- 
- New versions of a service may be unacceptable either for logical or performance reasons.
  - Two options in this case
    - Roll back (undo deployment)
    - Roll forward (discontinue current deployment and create a new release without the problem).
  - Decision to rollback can be automated. Some organizations do this.
  - Decision to roll forward is never automated because there are multiple factors to consider.
    - Forward or backward recovery
    - Consequences and severity of problem
    - Importance of upgrade



# States of upgrade.

- 
- An upgrade can be in one of two states when an error is detected.
    - Installed (fully or partially) but new features not activated
    - Installed and new features activated.

# Possibilities

---

- Installed but new features not activated
  - Error must be in backward compatibility
  - Halt deployment
  - Roll back by reinstalling old version
  - Roll forward by creating new version and installing that
- Installed with new features activated
  - Turn off new features
  - If that is insufficient, we are at prior case.

# Partial deployments

- 
- Limited production testing (canary)
  - Marketing testing (A/B)

# Canary testing

- 
- Canaries are a small number of instances of a new version placed in production in order to perform live testing in a production environment.
  - Canaries are observed closely to determine whether the new version introduces any logical or performance problems. If not, roll out new version globally. If so, roll back canaries
  - Named after canaries in coal mines.
  - Equivalent to beta testing
- 
- ~~for shrink wrapped software~~

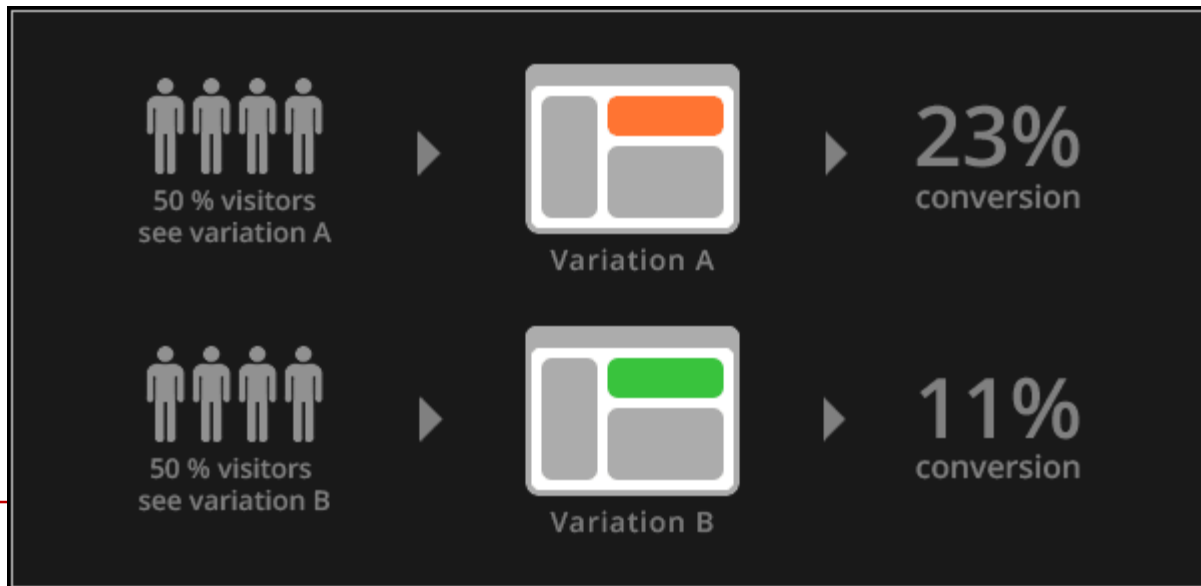


# Implementation of canaries

- 
- Designate a collection of instances as canaries. They do not need to be aware of their designation.
  - Designate a collection of customers as testing the canaries. Can be, for example
    - Organizationally based
    - Geographically based
  - Then
    - Activate feature or version to be tested for canaries. Can be done through feature activation synchronization mechanism
    - Route messages from canary customers to canaries. Can be done through load balancer or DNS server.
  - Measure performance metrics
-

# A/B testing

- A/B testing is done for marketing purposes, not testing purposes like canary testing
- Show different customers different web sites
- Compare results



# Examples

- 
- Do eBay users bid higher in auctions when they can pay by credit card?
  - Which promotional offers will most efficiently drive checking account acquisition at PNC Bank?
  - Which shade of blue for Google search results will result in more click throughs?

# Implementation

---

- The same as canary testing.
  - Use feature toggles
  - Make load balancer or DNS A/B aware
- Measure business measure responses.



# Summary - 1

- 
- Staging environment is place to test for non-functional qualities
    - Performance
    - Usability
    - Security
  - Management of database during staging is complicated
    - Sufficient data for test
    - Restoring data base
    - Maintaining private data
-

# Summary - 2

- 
- Two basic deployment strategies – Blue/Green and Rolling Upgrade
  - Use feature toggles to keep updates from being executed until all instances have been upgraded
  - Activate all instances simultaneously using coordination manager.
  - Maintain backward/forward compatibility
  - Treat database schema evolution as interface medication.
  - Paartial deployment strategies can be used for quality or marketing purposes.



# Deployment and Operations for Software Engineers

Chapter 9 Postproduction

# Overview

---

- **You Build It, You Run It**
- Logs and metrics
- Live Testing

# You build it, you run it

---

*“There is another lesson here: Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. The traditional model is that you take your software to the wall that separates development and operations and throw it over and then forget about it. Not at Amazon. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service.”*

*-Werner Vogels*

---

<https://queue.acm.org/detail.cfm?id=1142065>

# Scenario

---

- It is 3:00AM and your pager goes off.
  - There is a problem with your service!
  - You get out of bed and log onto the production environment and look at the services dashboard.
  - One instance of your service has high latency
  - You drill down and discover the problem is a slow disk
  - You move temporary files for your service to another disk and place the message “replace disk” on the operators queue.
-

# Troubleshooting process

---

- First step is to isolate problem
    - Current service
    - Upstream service (too many requests)
    - Downstream service (too slow)
  - Second step is to decide whether it is a hardware or software problem
    - What has changed in the software?
    - Has hardware shown signs of problems with other services?
    - If a single instance of multiple instances has problems, look for hardware first.
-

# Single service – single server

---

- Look at following data
  - CPU
  - Memory
  - I/O activity
  - Number of requests
  - Response time to inbound requests
  - Response time for outbound requests
  - Error rates
- Look for abnormal values



# Single service – multiple servers

- 
- Multiple servers served through a load balancer
  - Look at same set of data as for single server
    - CPU
    - Memory
    - I/O activity
    - Number of requests
    - Response time to inbound requests
    - Response time for outbound requests
    - Error rates
  - Look at aggregate values over multiple servers
-

# Isolating problem

---

- Is problem with this service or client or dependent services?
- If problem is with this service is it manifested across all servers or just one. I.e. drill down into aggregates to get individual values

# Multiple services – multiple servers

---

- Same basic strategy
  - Isolate problem through identifying problem by looking at aggregates
  - Drill down to decide service and server that contributes to problem
  - Look at what has changed in software and whether hardware has manifested problems earlier

# Overall requirements from this sequence of trouble shooting

---

- Gather variety of different kinds of data
  - Either resource usage or things that contribute to resource usage
  - Ensure each data item can be traced as to source and activity
- Collect data into a location where it can be queried and drilled into.

# Overview

---

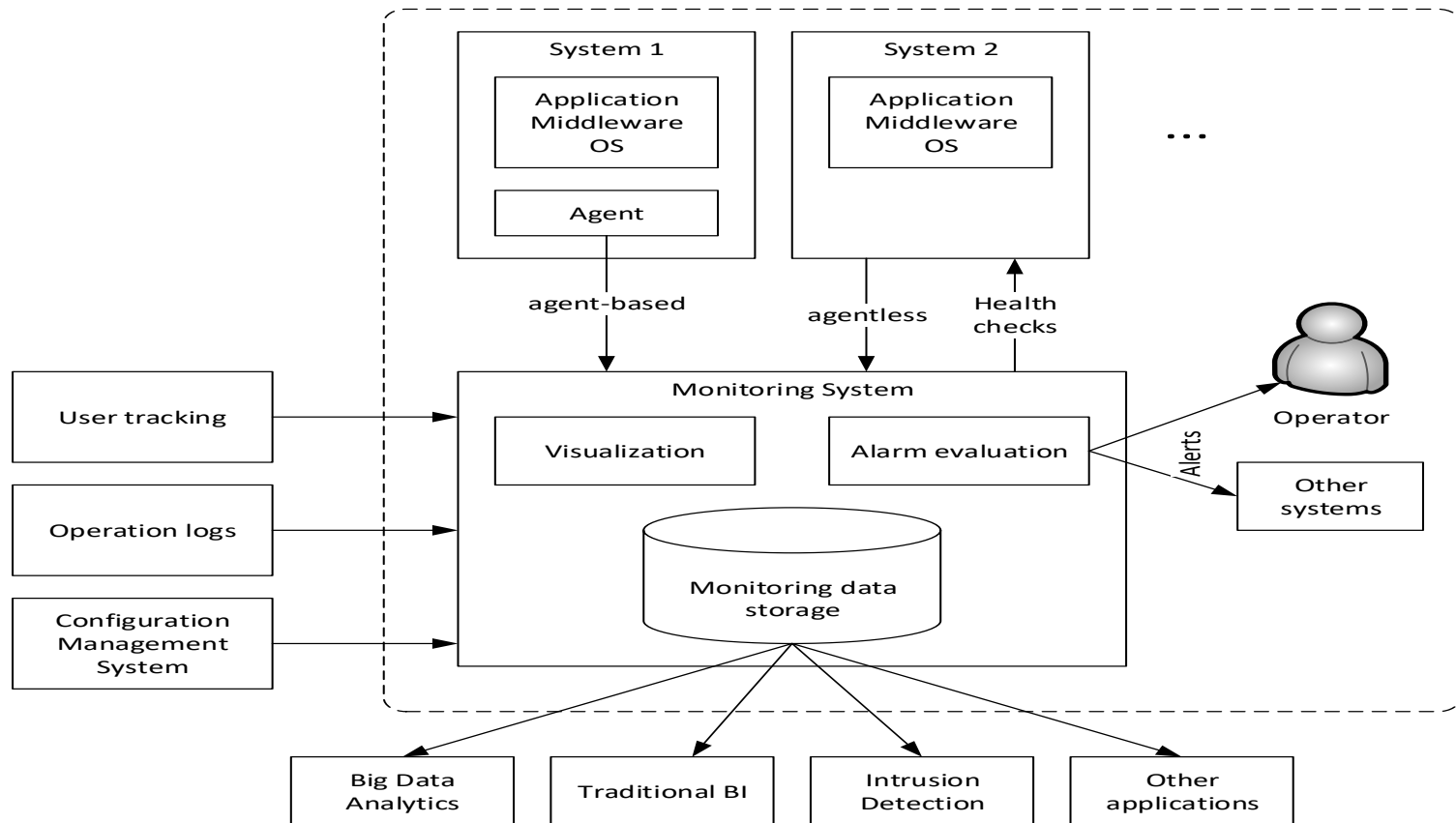
- You Build It, You Run It
- **Logs and metrics**
- Live Testing

# Information needs

---

- Metrics collected by infrastructure
- Logs from instance with relevant information
- Central repository for logs
- Dashboard that displays metrics
- Alerting system
  - Monitoring latency of instances
  - Rule: if high latency then alarm

# Architecture of Monitoring System



# Logs

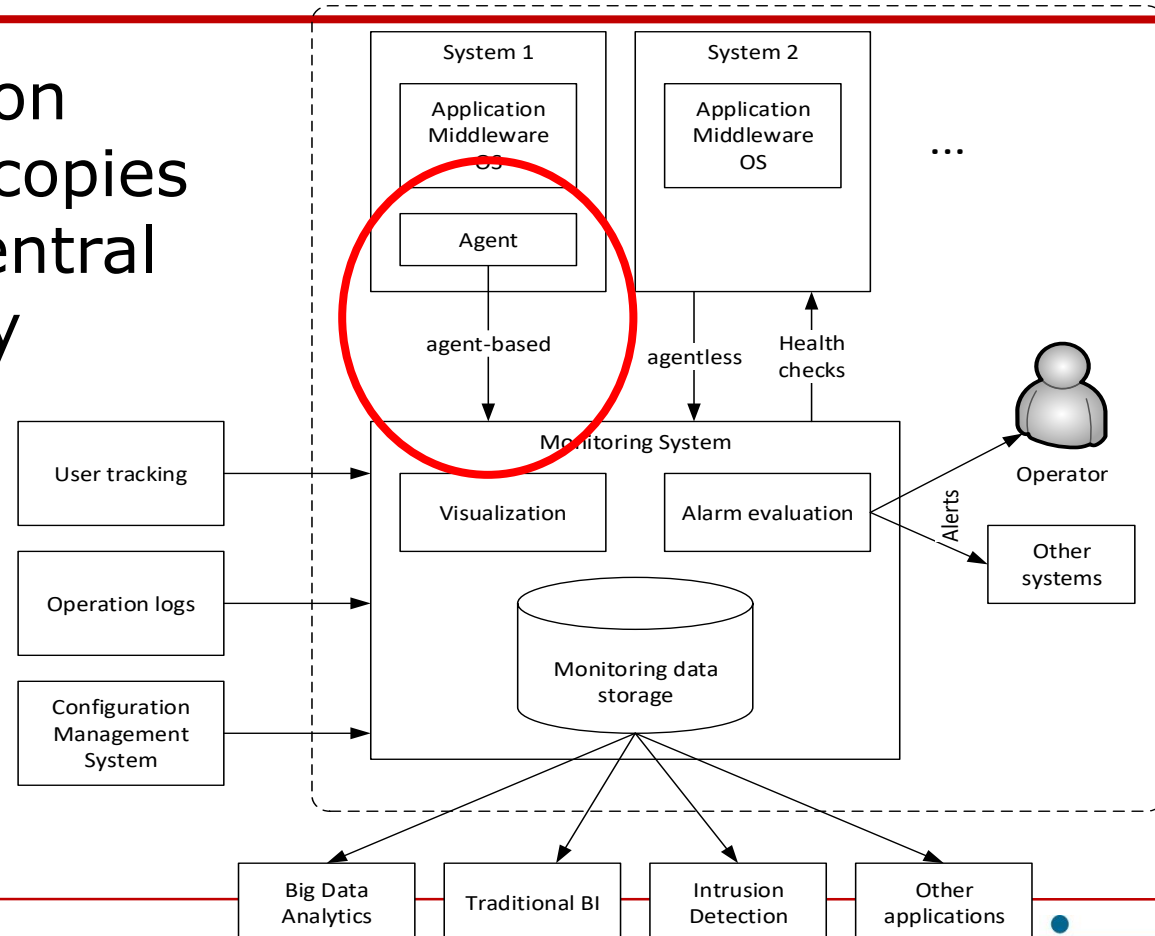
---

- A log is an append only data structure
- Written by each software system.
- Located in a fixed directory within the operating system
- Enumerates events from within software system
  - Entry/exit
  - Troubleshooting
  - DB modifications
  - ...



# Instance Log

Daemon on  
instance copies  
logs to central  
repository



# Logs on Entry/Exit

---

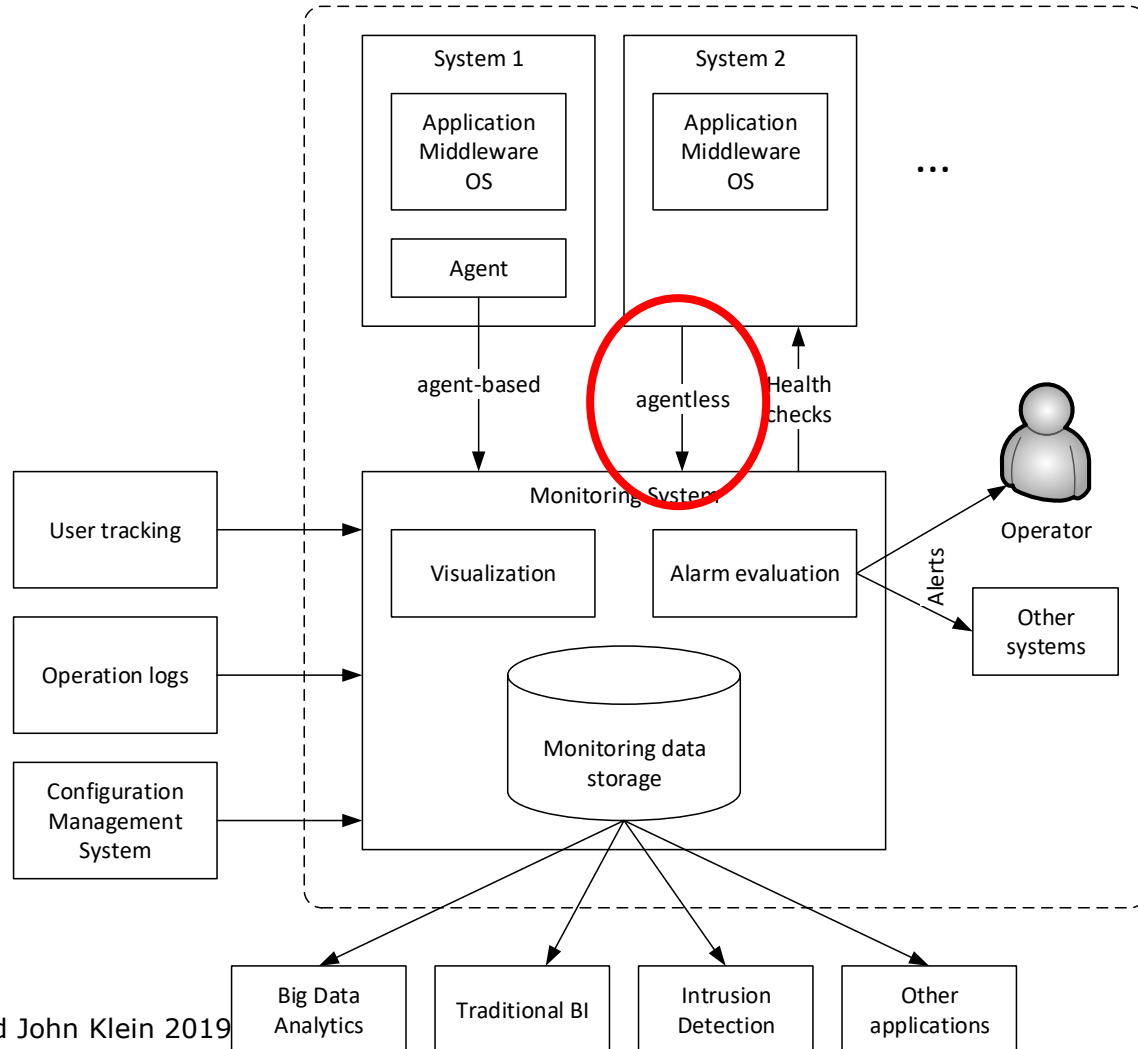
- Recall that Protocol Buffers automatically generate procedures that are called on entry/exit to a service
  - These procedures can be made to call logging service with parameters and identification information.
  - Logs on entry/exit can be made without additional developer activity
-

# Metrics

---

- Metrics are measures of activity over some period of time
- Collected automatically by infrastructure over externally visible activities of VM
  - CPU
  - I/O
  - etc

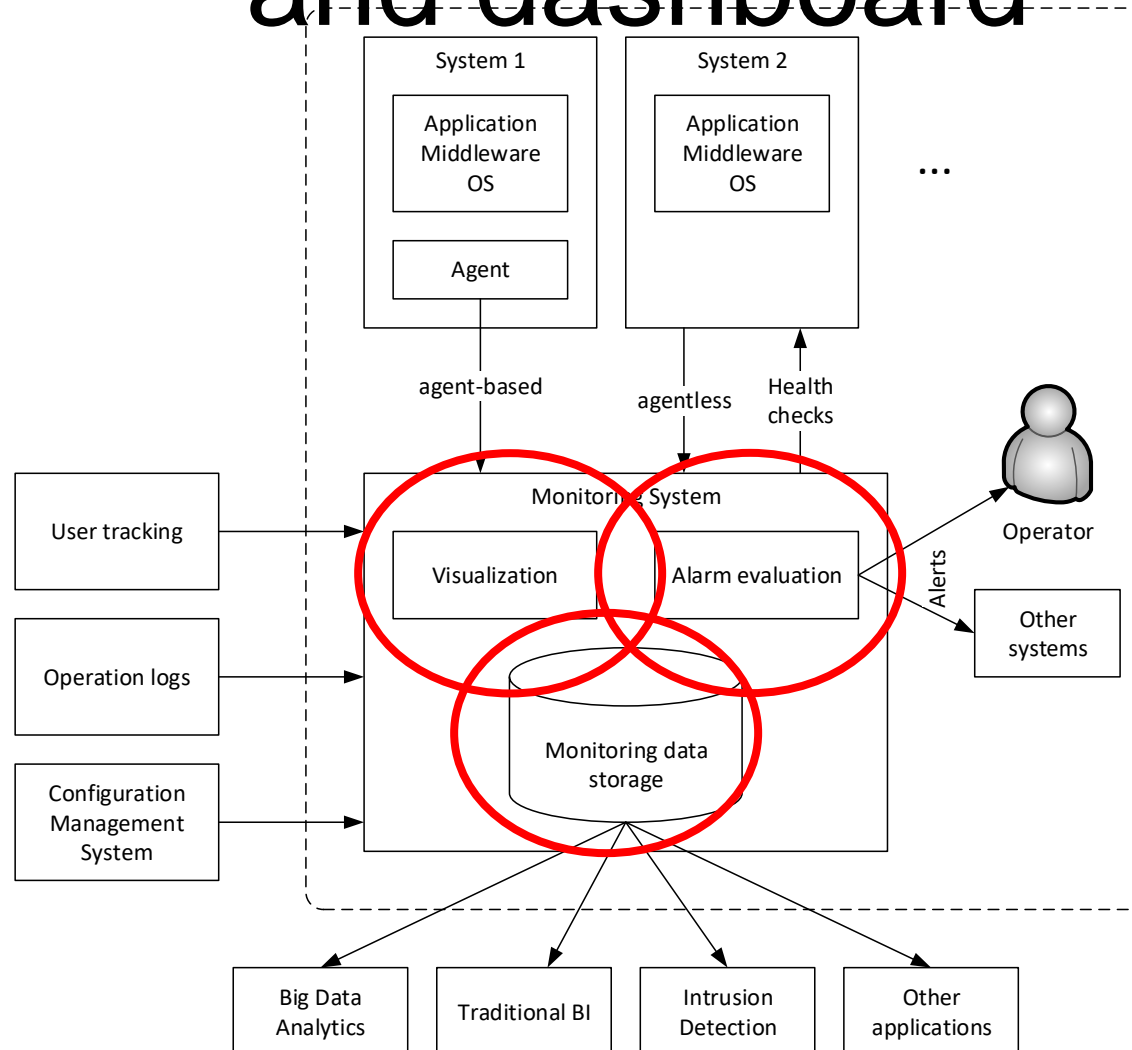
# Metrics collected by infrastructure



# Repository

- Logs and metrics are placed in central repository
- Repository generates alarms based on rules
- Provides central location for examination when problem occurs
- Displays information in dashboard that allows for drilling down to understand source of particular readings.

# Central Repository with alerting and dashboard



# Overview

---

- You Build it, You Run It
- Logs and metrics
- **Live Testing**

# Live testing

- 
- Netflix has a “Simian Army” to perform testing after a service is in production.
    - Chaos Monkey kills production processes
    - Latency Monkey introduces extra latency into the network.
    - Various other monkeys perform janitor services
      - Looking for certificates or licenses about to expire
      - Ensuring appropriate localization
      - Cleaning up unused resources
      - Ensuring security groups are appropriately used.



# Summary

---

- Developers may carry pagers and be first responders
- Determining problem requires access to a wide variety of data
  - Logs
  - Metrics
- Postproduction testing may introduce errors or provide janitorial services



# Deployment and Operations for Software Engineers

Chapter 10 Disaster Recovery

# Overview

---

- **Basic concepts**
- Data Strategies for Tiers 2-4
- Tier 1 Data Management
- Big Data
- Software in the Secondary Data Centers
- Fail Over

# Terminology

---

- Business continuity – keeping your business going in the event of a disaster
  - Involves customers, employees, protecting people and equipment
- Disaster Recovery – the IT portion of business continuity. Maintaining service to customers

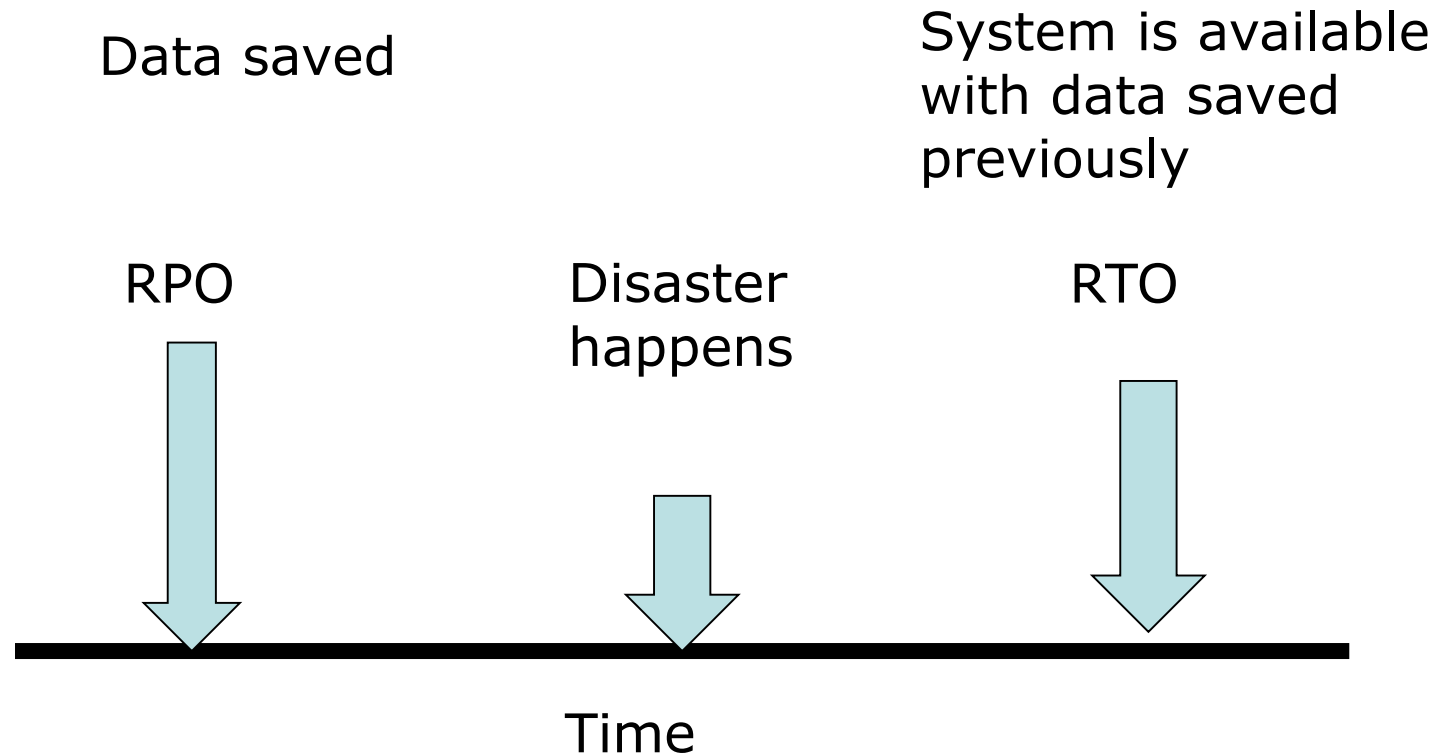
# Key measures per system

---

- RTO – recovery time objective
  - How long before system is in service again
- RPO – recovery point objective
  - How much data can be lost in the event of a disaster
- Will vary for each system in an organization

# Graphical representation

---



# Tiers

---

- Divide your systems into tiers based on RTO and RPO
  - Tier 1 (mission critical) – 15 minutes
  - Tier 2 (important support ) - 2 hours
  - Tier 3 (less important support) - 4 hours
  - Tier 4 (everything else) - 24 hours

# Secondary data center

---

- When a disaster occurs, your data center will be out of operation for days/weeks/months
- You need a secondary (back up) data center





# Types of secondary data centers

---

- Warm – computing facilities in place but your software has not been loaded
- Hot – computing facilities in place with your software loaded and either executing or ready to execute. Current data not in data center
- Mirrored – computing facilities in place, current software in place and executing, data up to date.
- All secondary locations are geographically separated from primary data center

# Overview

---

- Basic concepts
- **Data Strategies for Tiers 2-4**
- Tier 1 Data Management
- Big Data
- Software in the Secondary Data Centers
- Fail Over

# Online or offline?

---

- Tiers 2-4 have recovery times in terms of hours
- Major decision is whether to keep data
  - Online – in a different region or availability zone or
  - Offline – on tape that is stored remotely from primary data center

# Considerations

---

- Volume of data
- Storage costs
- Encryption of data on tape
- Recovery time for tape from storage site
- Transfer time from online storage to secondary site

# Overview

---

- Basic concepts
- Data Strategies for Tiers 2-4
- **Tier 1 Data Management**
- Big Data
- Software in the Secondary Data Centers
- Fail Over

# Data in Tier 1

---

- RTO and RPO in minutes, not hours
  - Data must be kept online and up to date in secondary data center
  - Secondary data center must be mirrored.
  - Database system can be used to keep transactional data consistent at both sites
-

# Non transactional Tier 1 data

---

- Non replicated data.
    - Session data may not be replicated
    - Requires user to log in again if disaster
  - Slowly changing data
    - Static web pages, videos, other data changes only slowly
    - Can be kept up to date with configuration management system
-

# Overview

---

- Basic concepts
- Data Strategies for Tiers 2-4
- Tier 1 Data Management
- **Big Data**
- Software in the Secondary Data Centers
- Fail Over



# Big Data

---

- “Big Data” is a data set too large to back up
- Data is divided into groups – “shards”
- Each shard is replicated several times
  - For performance and availability reasons
  - Each shard is managed by database system that keeps replicas up to date

# Overview

---

- Basic concepts
- Data Strategies for Tiers 2-4
- Tier 1 Data Management
- Big Data
- **Software in the Secondary Data Centers**
- Fail Over

# Software in secondary data center

---

- Must be kept in alignment with software in primary data center
  - Version inconsistency may lead to behavioral inconsistency.
- Configuration management systems can work across data centers
- Deployment process for modified software should consider secondary data center

# Overview

---

- Basic concepts
- Data Strategies for Tiers 2-4
- Tier 1 Data Management
- Big Data
- Software in the Secondary Data Centers
- **Fail Over**

# Fail over

---

- Three activities to a fail over process
  - Trigger switch to secondary data center
  - Activate secondary data center
    - Involves ensuring data and software are up to date
  - Resume operations at secondary data center

# Trigger

---

- Manual or automatic
- In either case, the trigger is scripted so that it is a one button/command
- Automatic trigger should only be used with very short RTO
  - Requires data center failure detector that may have false positives
  - Any failover has business implications.

# Activate secondary data center

---

- Data and software must be brought up to date.
- If secondary data center is not mirrored, the last back up must be restored
- User requests sent to secondary data center
- Software is activated based on tiers

# Resume operations

---

- Make secondary data center be primary
- May need to locate a new secondary data center in case of disaster at currently operating center



# Testing fail over

---

- If you can afford down time, test during scheduled down time
- If no scheduled down time, then test using staging environment where care is taken to avoid corrupting production data base

# Summary

---

- RPO and RTO are basic measures to describe behavior in case of failure. Used to divide apps into tiers
  - Secondary data center must be identified
  - Tiers 2-4 can use back ups, either online or offline
  - Tier 1 data has database support to keep copy at secondary data center up to date
  - Software in secondary data center kept current with configuration management system
  - Fail over is scripted and must be tested
-



# Deployment and Operations for Software Engineers

Chapter 11 Secure Development – 1

# Overview

---

- **Identify and protect critical data and resources**
- Managing credentials for access to services
- Managing credentials for individuals

# Data to be protected

---

- Credentials
- Organization sensitive data
- Personally sensitive
- Legally or contractually required data
  - Health data
  - Credit card data
- Data relating to citizens of particular jurisdictions.

# How to protect data

---

- Do not collect it if possible
- Anonymize as soon as possible
- Encrypt data at rest and in transit
- Do not put sensitive data in logs
- Use data models that separate critical data from other data (allows different access controls)

# Resources to be protected

---

- Hardware
  - CPU
  - Memory
  - Disk
- APIs

# Some techniques for protecting resources

---

- Sanitize inputs both from client side and server side. Prevents buffer overflow and SQL injection attacks.
- Encrypt request/response.
- Do not store sensitive data inside cookies.



# Overview

---

- Identify and protect critical data and resources
- **Managing credentials for access to services**
- Managing credentials for individuals

# Problem

---

- One service (client) wishes to access a second service
- Second service requires access credentials.
- How does client get credentials?

# Solution 1

---

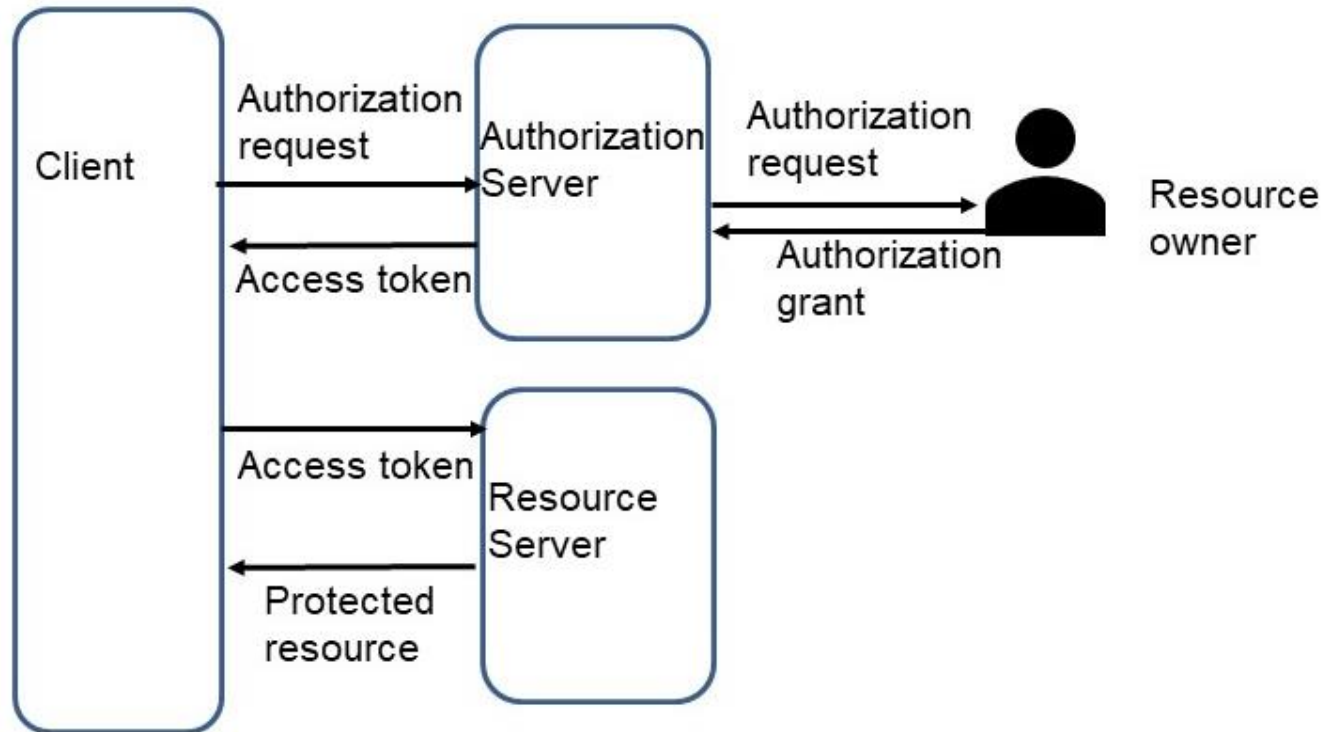
- Configuration management systems have extensions called “vaults”
  - Vaults maintain information in an encrypted form and require special registrations to access.
  - Store credential for service in vault
  - Client retrieves credential from value
  - Requires that client be authenticated to configuration management system
  - Moves problem from service to vault management which can be better controlled by an organization
-

# Solution 2

---

- Use OAuth
  - Standard
  - Has four roles:
    - Resource owner – owns resource
    - Resource server – controls the resource
    - Client – wishes to access resource
    - Authorization server – controls authorization to resource
-

# Sample OAuth Protocol



# Overview

---

- Identify and protect critical data and resources
- Managing credentials for access to services
- **Managing credentials for individuals**

# Principle of least privilege

---

- Individuals should be given the least privileges they need to do their job
- System administrators need different privileges than developers than testers, etc

# RBAC

---

- Role Based Access Control (RBAC)
- Managing credentials for all of the employees as individuals gets too complicated when there are >500 employees
- Define a group of roles and provide credentials based on roles
- Assign individuals to one or more roles.



# Problems with RBAC

---

- Explosion of roles
  - Suppose in a bank with multiple branches, one branch gives managers the power to approve loans up to \$10,000 and another branch gives managers power to approve loans up to \$20,000
- Are these two different roles?
- Does manager need multiple roles? –  
including max loan that can be approved.

# Centralizing account management

---

- Managing individual accounts and roles is usually done using *Lightweight Directory Access Protocol* (LDAP)
- Individuals are registered in the LDAP server with account information and roles.

# Employee leaves organization

---

- LDAP simplifies managing of computer accounts for individuals joining or leaving an organization
- An employee who leaves has their account deleted in the LDAP and they can no longer access computer system.
- What about the credentials for employees who leave?

# Credentials are assigned to roles

---

- Suppose individual who leaves has maintenance responsibilities
- This allows access to certain rooms via key code.
- If key code is changed then remaining maintenance personnel cannot access rooms.

# Credential rotation

---

- Credentials can be rotated periodically or upon individual leaving.
- Credential rotation is not instantaneous.
- An application may have to be designed for alternative credentials.

# Summary

---

- Critical data comes in a variety of forms. Protection may be required by law, regulations or corporate needs.
- Vaults and OAuth are used to control access to services.
- RBAC and LDAP are used to manage credentials for individuals.



# Deployment and Operations for Software Engineers

Chapter 11 Secure Development – 2

# Overview

---

- **Software Supply Chain and Software Assurance**
- Weaknesses and vulnerabilities
- Vulnerability discovery and patching



# Software Supply Chain

---

- The set of software that contributes to the content of a service.
  - Includes
    - Operating system
    - Middle ware
    - Frameworks
    - Supporting tooling
    - ...
  - May be open source or commercial off the shelf (COTS)
-

# Open source selection criteria

---

- Project maturity and development activity.
- Identified and engaged maintainer(s).
- Repository used for the project.
- The download confirmation hash.
- Pedigree.

# Overview

---

- Software Supply Chain and Software Assurance
- **Weaknesses and vulnerabilities**
- Vulnerability discovery and patching

# Definitions

---

- A software vulnerability allows an attacker to gain access to a system or network.
- Software weaknesses are errors that can lead to software vulnerabilities.
- Similar to distinction between error, fault, and failure in reliability theory.
- An attack is the use of vulnerabilities by an adversary to achieve a technical impact

# Catalogs

---

- Common Weakness Enumeration (CWE)
- Common Vulnerability and Exposure
- List vulnerabilities (or weaknesses) for specific version and release of specific software system.
- Common Attack Pattern Enumeration and Classification (CAPEC)

# Deployment pipeline

---

- ~15-25% of attacks are by insiders (people working for the organization being attacked).
  - The deployment pipeline is a good vehicle for an insider attack. E.g. modify CI server to include malware in every build
  - Processes to protect deployment pipeline include restricting modifications to authorized personnel, broadcasting information about modifications to all team members.
-

# Overview

---

- Software Supply Chain and Software Assurance
- Weaknesses and vulnerabilities
- **Vulnerability discovery and patching**

# Vulnerability/patch process

---

1. A vulnerability is discovered
2. The vulnerability is reported to Computer Emergency Response Team (CERT/CC). This starts the disclosure window (currently 45 days)
3. CERT/CC publicly discloses the vulnerability and lists the vulnerability in the CVE.
4. The vendor issues a patch, referencing the CVE ID.
5. You are informed of the patch
6. You apply the patch.



# Problems with process

---

- Too much information
  - Multiple vendors with multiple products are continually issuing patches
  - Which ones apply to the software you are using?
- Deciding whether and when to apply a patch. Depends on severity of problem being fixed and your operational procedures.

# One process

---

- Rebuild all software periodically.
- Guarantees that latest patches are incorporated for software that you use
- Costs rebuilding time

# Summary

---

- Supply chain may be a source of vulnerabilities
- Catalogs of vulnerabilities and patches
- Patching process must be defined for your organization