Carnegie Mellon

institute for SOFTWARE RESEARCH

Introduction to DevOps

Len Bass

Agenda AM

9:00-9:15	Introductions
9:15-9:45	What is DevOps
9:45-10:30	Security and SSH
10:30-10:45	Break
10:45-11:30	VM assignment
11:30-12:00	Basic tools
12:00-1:00	Lunch

Agenda PM

1:00-1:30	Introduction to deployment pipeline
1:30-2:30	Vagrant Assignment
2:30-3:00	Containers
3:00-3:15	Break
3:15-3:45	Docker assignment
3:45-4:45	Microservice architecture
4:45-5:00	Wrap up

Introductions

- What is your background?

Carnegie Mellon

institute for SOFTWARE RESEARCH

What is DevOps?

Overview

- **Velocity of releases**
- Causes of delay in releases
- Definition of DevOps and categorization of DevOps processes

Velocity of new releases is important

- Traditionally organizations deployed a new release quarterly or monthly.
- In the modern world, this is too slow.
- Internet companies deploy multiple times a day

Release schedule statistics

- Etsy releases 90 times a day
- Facebook releases 2 times a day
- Amazon had a new release to production every 11.6 seconds in May of 2011

Overview

- Velocity of releases
- **Causes of delay in releases**
- Definition of DevOps and categorization of DevOps processes

Why have a new releases?

- An event triggers a requirement for a new release
  - It could be an idea for a new feature on an existing system
  - It could be a merger between two organizations
  - It could be a problem with a system in production
  - it could be competitive pressure
  - It could be a desire for more efficiency
- In any case, the event causes a requirement for creating a new system or modifying an existing system or systems.
- Modifying an existing system is, by far, the most common response.

Over the wall development

–

Board has idea

Developers implement

Operators place in production

Time

The triggering event causes requirements to be generated

- Requirements could be expressed in a variety of fashions
  - Documents
  - User stories
  - Intuition of the developers
- Requirements are divided and assigned to development teams

Developers organization

- There are a variety of different development teams
- Each team has its requirements
- Developers on each team work on requirements, in some fashion
- At some point a team feels they have satisfied the requirements they have been assigned
- Their code is complete – their job is done.

What is wrong?

- Code Complete ≠ Code in Production
- Between the completion of the code and the placing of the code into production is a step called: Deployment
- Deploying completed code can be very time consuming because of concern about errors that could occur.

What is the work flow for a multi-team effort?

- You develop and test your code in isolation
- Your code is integrated with code developed by other teams to see if an executable can be constructed.
- The built system is tested for correctness
- The built system is tested for performance and other qualities (staging)
- The built system is placed into production

What can go wrong – Integration

- Not all portions of the system are available
  - Portions developed by other teams
  - Portions developed by 3rd party
  - Names and signatures of methods from other software are inconsistent
- Sequencing errors
  - Other teams do not follow the contract with your code in terms of sequence of method calls
- Version incompatibility
  - Your team assumed version A of 3rd party software but another team assumes version B

What can go wrong – integration 2

- Data problems
  - Database data is not refreshed for each test
  - Data does not flow correctly to your code
- Configuration problems
  - Configuration parameter settings for code developed by different teams are incompatible
  - Configuration parameters are not specified
  - External services are not reachable for security or configuration reasons
- Etc

What can go wrong – production

- Configuration problems
    - Requires authentication and authorization
    - Keys must be kept securely
    - Inconsistent configurations
- Performance problems
    - Under actual load, system may not have adequate performance
- Logical problems
    - May require new version to be rolled back
    - Database may have been corrupted
- Etc

Time is passing

- Every error must either be corrected or prevented.
- Correcting errors takes time
- Preventing errors can be done through some combination of
  - Process
  - Architecture
  - Tooling
  - Coordination among teams.
- Coordination takes time.

How much time?

- Historically, releases are scheduled for once a quarter or once or twice a year to give time to coordinate and adequately test.
- This means there may be months delay before a new concept or feature is added to a system.
- This delay has become more and more unacceptable.

- Velocity translates into time to market!!

Overview

- Velocity of releases
- Causes of delay in releases
- **Definition of DevOps and categorization of DevOps processes**

What is DevOps?

DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.

- DevOps practices involve developers and operators' processes, architectures, and tools.
- DevOps is also a movement – like agile.

DevOps processes can be divided into three categories

1. Reduce errors during deployment
2. Reduce time to deploy
3. Reduce time to resolve discovered errors

Summary

- DevOps is motivated by a desire to reduce the time to market for changes to a system
- Historically releases were scheduled events – taking months to prepare
- DevOps is defined as a collection of processes whose goal is to reduce the time between commit and normal production
- These processes either
  - Prevent errors
  - Speed up the detection and correction of errors

Questions?

Carnegie Mellon

institute for SOFTWARE RESEARCH

Infrastructure Security

Overview

- **What is security?**
- Cryptography
- Public Key Infrastructure and Certificates
- Key Exchange
- Transport Level Security (TLS)
- Secure Shell (SSH)

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

Overview

- What is security?
- **Cryptography**
- Public Key Infrastructure and Certificates
- Key Exchange
- Transport Level Security (TLS)
- Secure Shell (SSH)

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

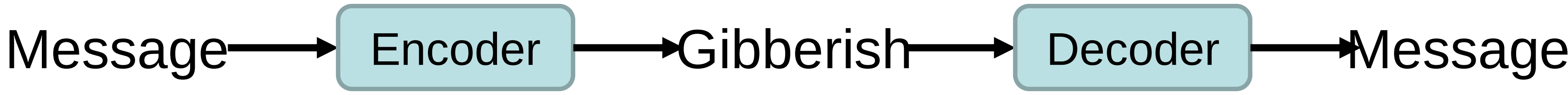
Hashing

- 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**
- Key Exchange
- Transport Level Security (TLS)
- Secure Shell (SSH)

Public Key Infrastructure (PKI)

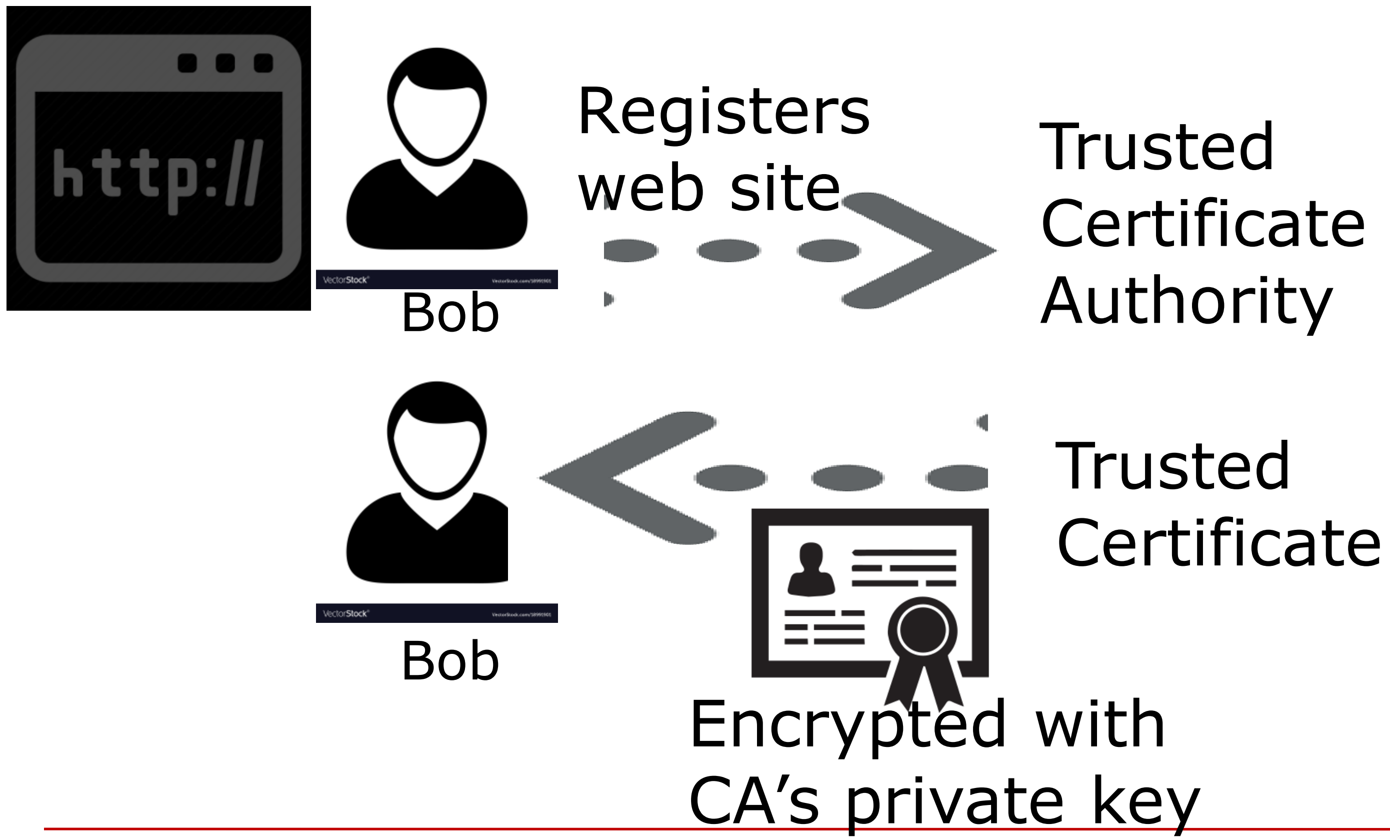


- PKI is based on public and private keys
- Public key is known to everyone
- Private key is known only to you
- Messages encrypted with public key can be decrypted by private key (and vice versa)
- Message sent *to* you is encrypted with your public key. Only you can read it
- Message sent *by* you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

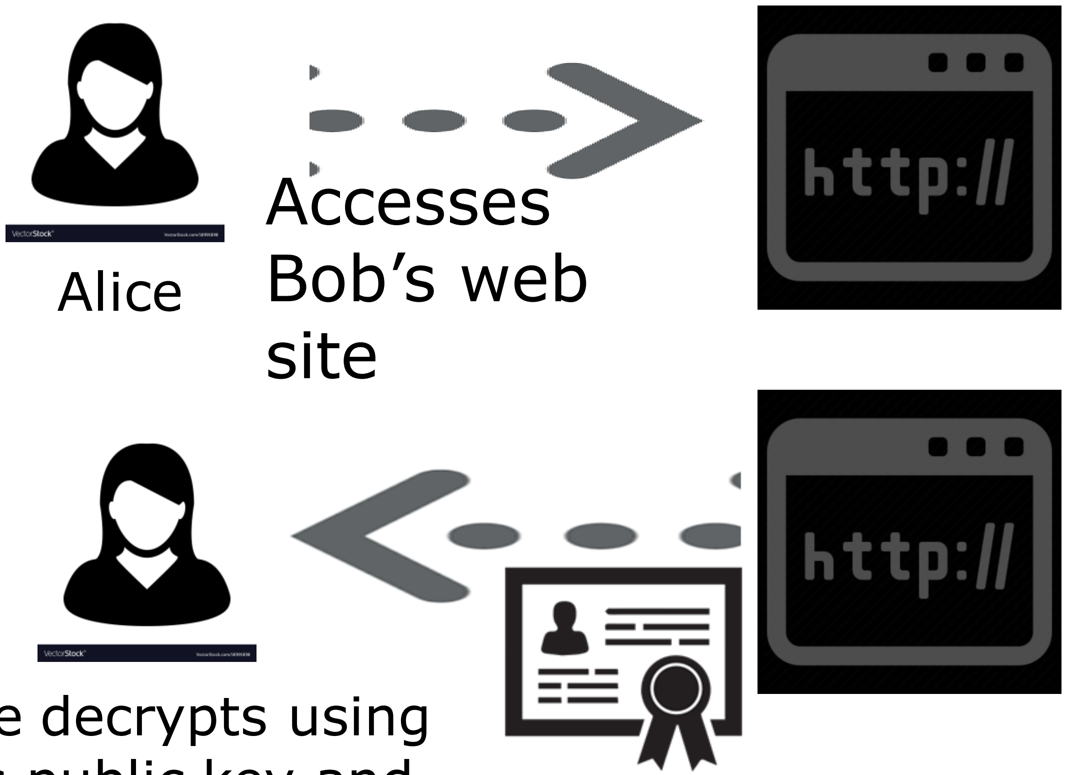
Certificates

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

Registering with Certificate Authority (CA)

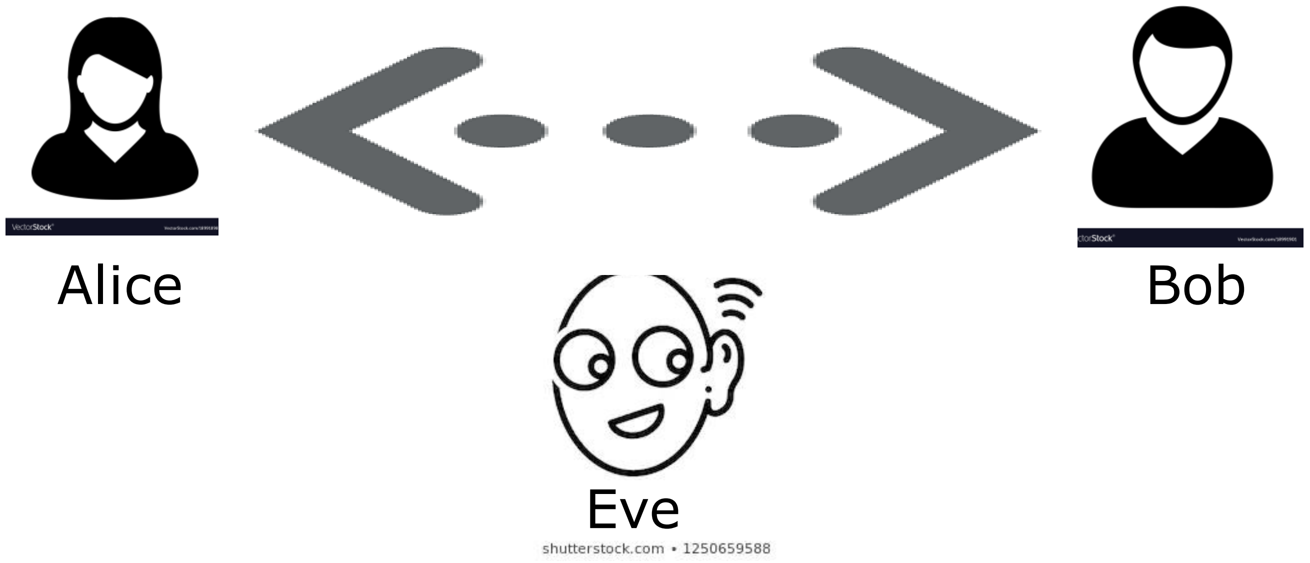

http://
Bob
Registers web site
Trusted Certificate Authority
Bob
Trusted Certificate
Encrypted with CA's private key

Accessing Web Site



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

Key exchange

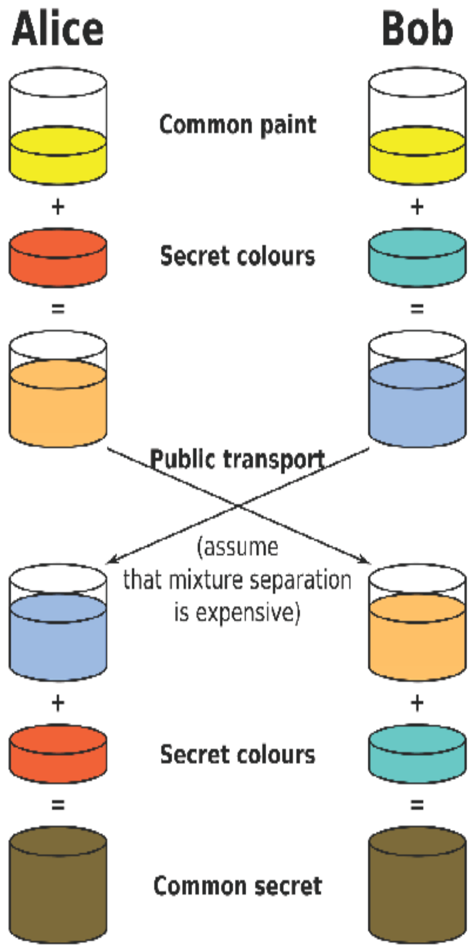


Alice and Bob wish to communicate securely even though Eve is eavesdropping.

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

- What is security?
- Cryptography
- Public Key Infrastructure and Certificates
- Key Exchange
- **Transport Level Security (TLS)**
- Secure Shell (SSH)

Carnegie Mellon University

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

institute for SOFTWARE RESEARCH

TLS

- TLS (Transport Layer Security) is the basis for https
- Builds on Diffie-Hellman and Public Key Infrastructure
- 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. Certificate is encrypted using Certificate Authority 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

- What is security?
- Cryptography
- Public Key Infrastructure and Certificates
- Key Exchange
- Transport Level Security (TLS)
- **Secure Shell (SSH)**

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.

Summary

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

Questions?

VM Assignment

- Create two ubuntu virtual machines where one can ping the other.
- Use VirtualBox

Carnegie Mellon University

Relevant web sites

- https://www.virtualbox.org/wiki/
- http:/osboxes.org

institute for SOFTWARE RESEARCH

Carnegie Mellon University

Discussion

- Let's talk about IP addresses
- Where do they come from
- What is the difference between public and private IP addresses
- What is the difference between IP addresses and ports?

Carnegie Mellon

institute for SOFTWARE RESEARCH

Basic Tools

Overview

- **Version Control**
- Configuration Management Tools
- Provisioning

Version Control System (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 or other checks on check-in

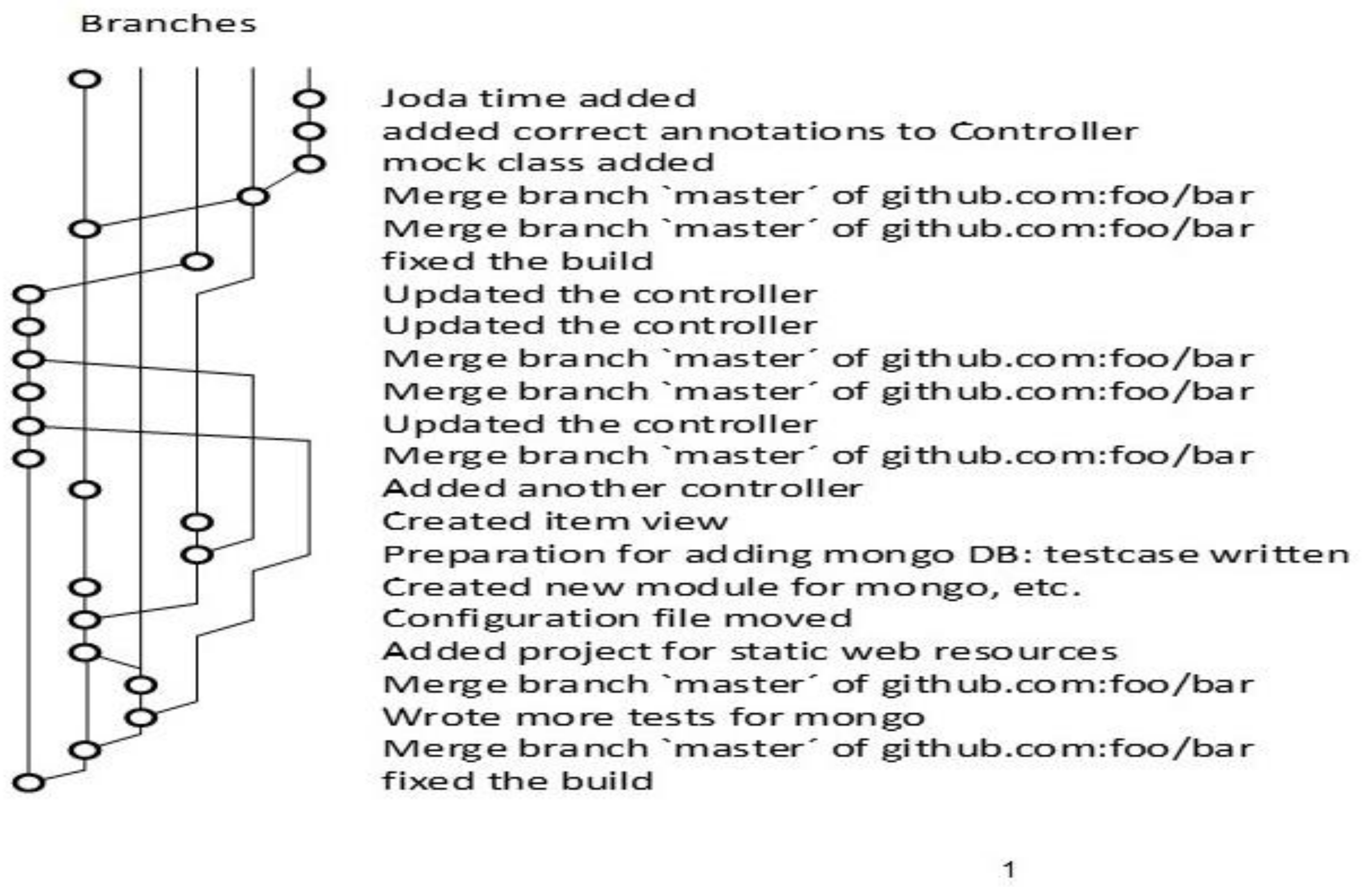
Branch process

- 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


Branches
Joda time added
added correct annotations to Controller
mock class added
Merge branch `master´ of github.com:foo/bar
Merge branch `master´ of github.com:foo/bar
fixed the build
Updated the controller
Updated the controller
Merge branch `master´ of github.com:foo/bar
Merge branch `master´ of github.com:foo/bar
Updated the controller
Merge branch `master´ of github.com:foo/bar
Added another controller
Created item view
Preparation for adding mongo DB: testcase written
Created new module for mongo, etc.
Configuration file moved
Added project for static web resources
Merge branch `master´ of github.com:foo/bar
Wrote more tests for mongo
Merge branch `master´ of github.com:foo/bar
fixed the build
1

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**
- Provisioning

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 (or TLS) 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
- **Provisioning**

Manual Provisioning

- Given a virtual machine or a container, you can provision (add software) manually by using package manager, e.g. apt-get in Ubuntu.
- Problems with manual provisioning are
  - Errors in specification
  - Inconsistencies among team members
  - Repeatability
  - No history

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 (configuration management tools)
  - Replicas across data centers should have same software installed down to version and patch numbers.

Provisioning tools

- A provisioning tool such as Vagrant or Cloud Formation (for AWS) provides a scripting basis for establishing environment
- Allows one command provisioning. E.g. Vagrant up
- Maintains consistency among team members
- Can be version controlled

Summary

- Version control systems enable controlled sharing and modification
- Configuration management systems maintain consistency of software and versions across classes of machines
- Provisioning tools simplify the creation of an environment

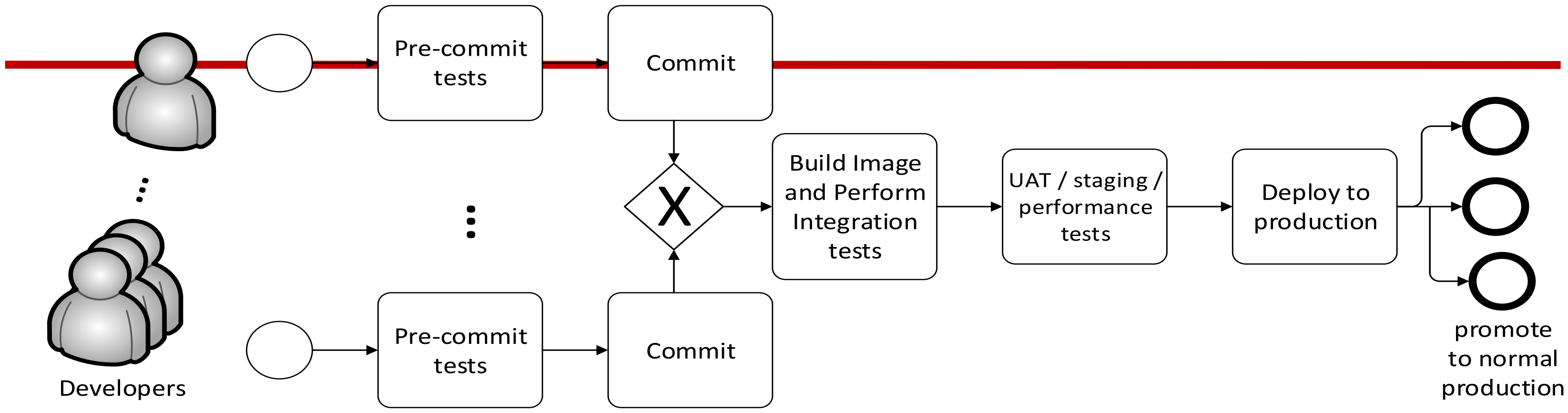
Questions?

Carnegie Mellon

institute for SOFTWARE RESEARCH

Deployment Pipeline

Deployment pipeline


Pre-commit tests
Commit
Developers
Pre-commit tests
Commit
X
Build Image and Perform Integration tests
UAT / staging / performance tests
Deploy to production
promote to normal production

- 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.

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.

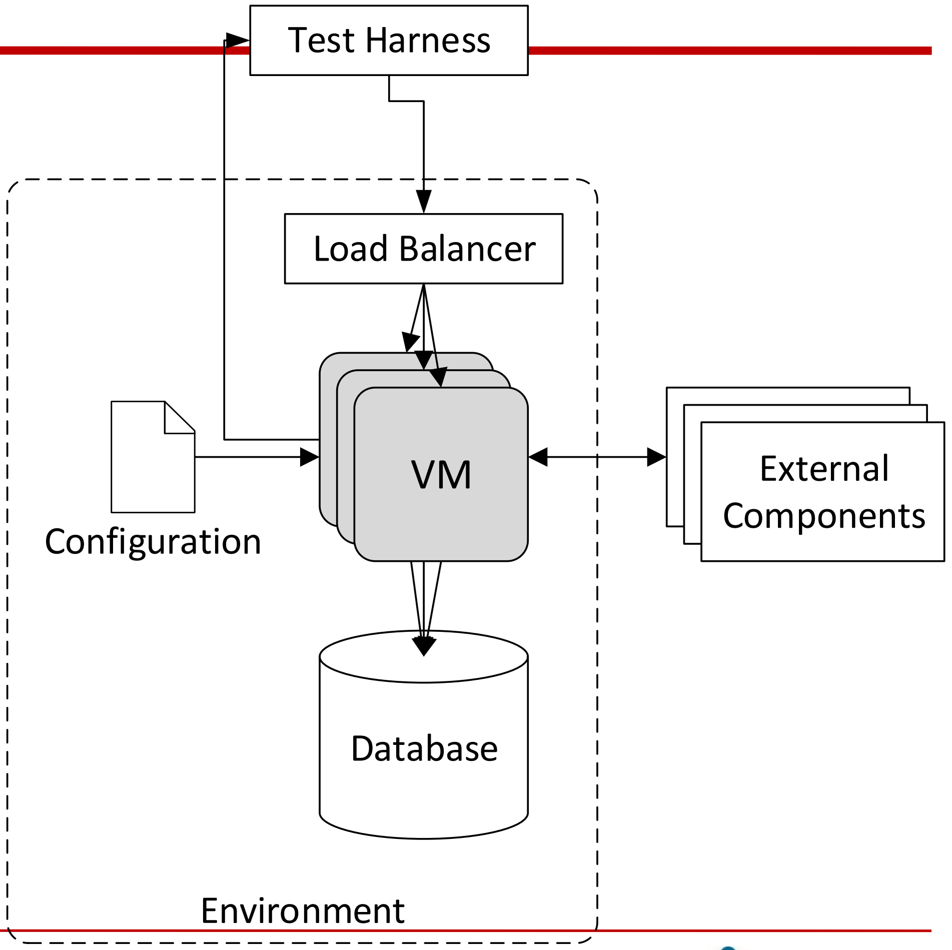
Testing

- 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

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



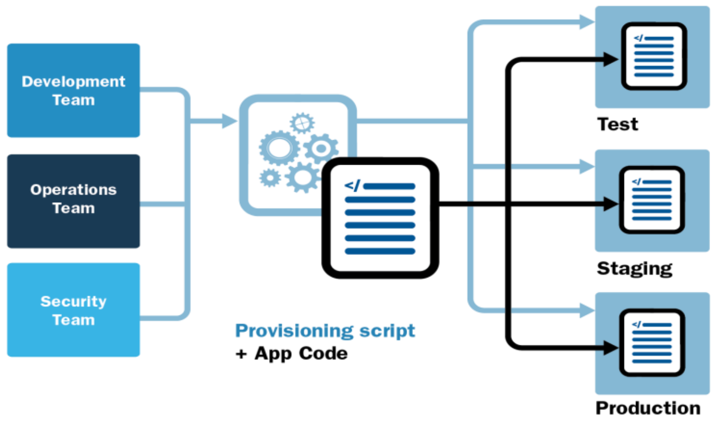
A test harness generates inputs and compares outputs to verify the correctness of the code.

Tooling

- 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

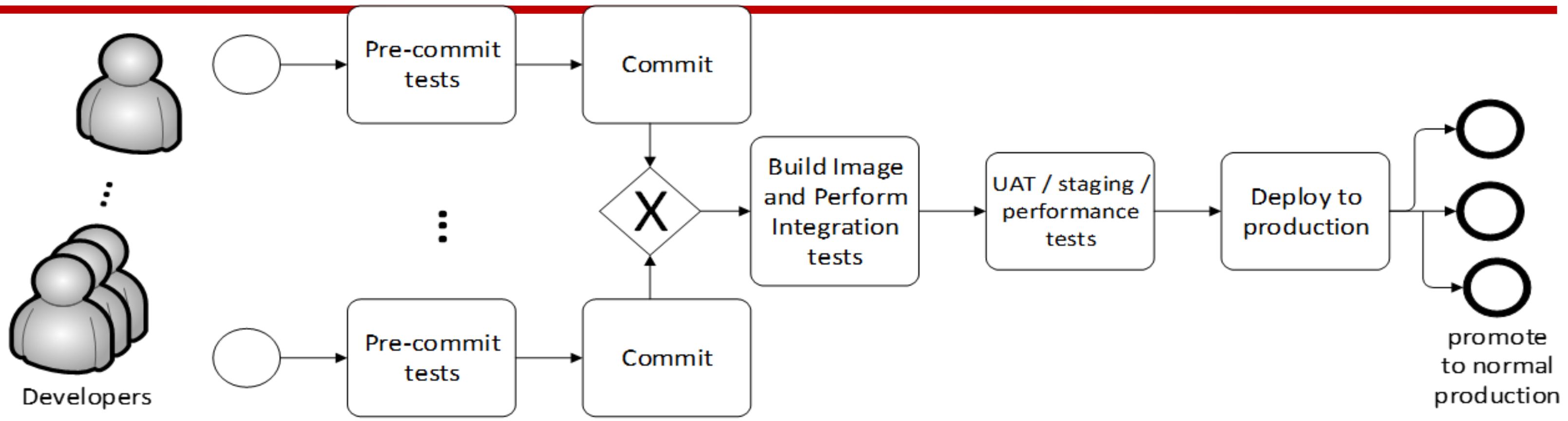
Scripts for pipeline tools are an example of infrastructure as code (IaC). They should be version controlled and tested just like application code.



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


Pre-commit tests
Commit
Build Image and Perform Integration tests
UAT / staging / performance tests
Deploy to production
promote to normal production
Developers
Pre-commit tests
Commit

- 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

Summary

- 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

Assignment

- Use Vagrant to create two ubuntu virtual machines in VirtualBox where one can ping the other.

Relevant web sites

- https://docs.vagrantup.com/v2/getting-started/index.html
- http://www.sitepoint.com/getting-started-vagrant-windows/

Vagrant instructions

Inhttps://github.com/cmudevops/workshop-instructionsstructions can be found at

Discussion of Assignment

- Compare doing assignment with Vagrant vs doing it manually (assignment 1)

Carnegie Mellon

institute for SOFTWARE RESEARCH

Containers

Overview

- **Container definition**
- Layers and deployment
- Container repositories
- Clusters and orchestration
- Serverless Architecture

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

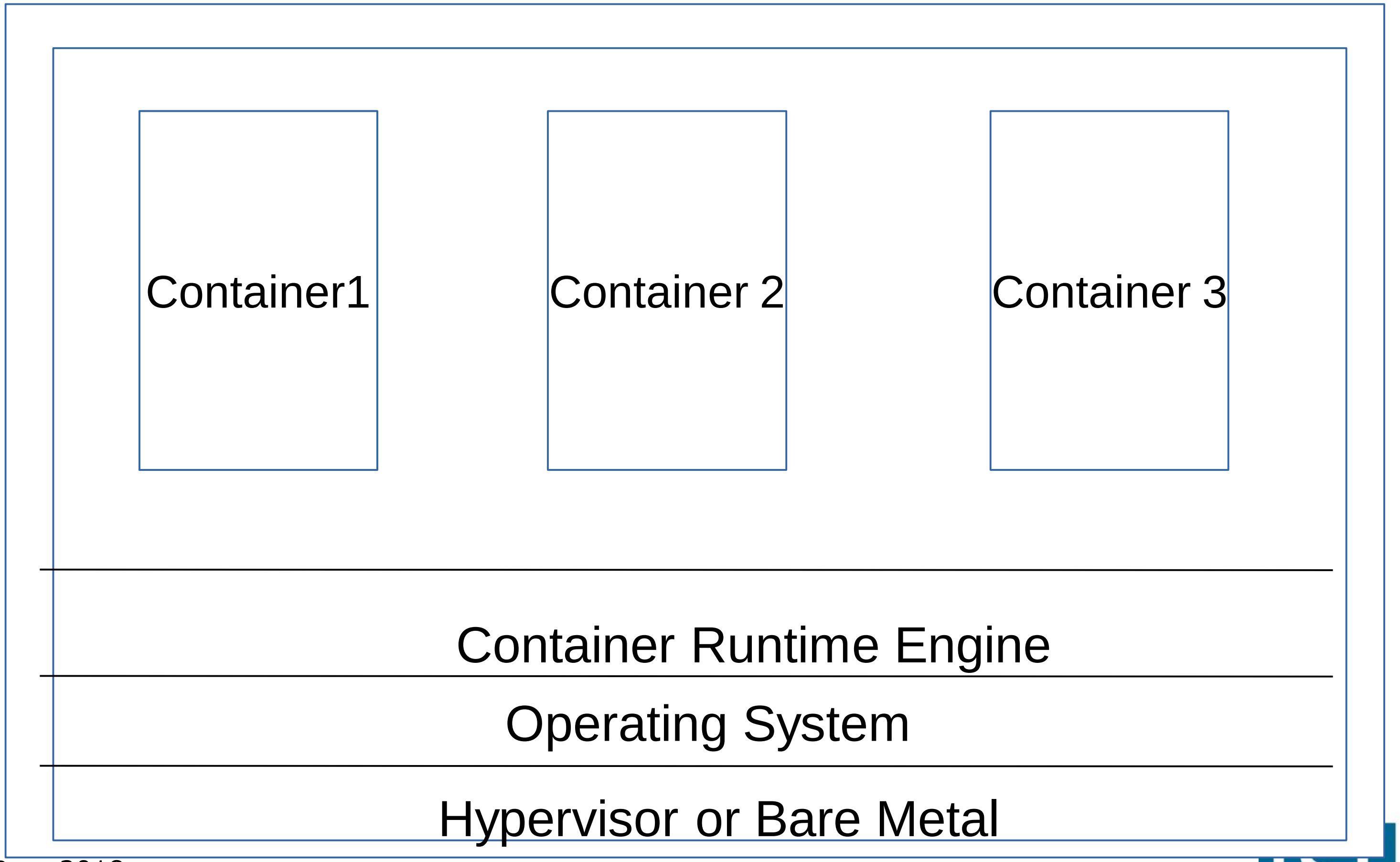
Virtual Machine
- Isolate address space
- isolate files and networks
- Heavyweight

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

Carnegie Mellon University

Container hierarchy



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

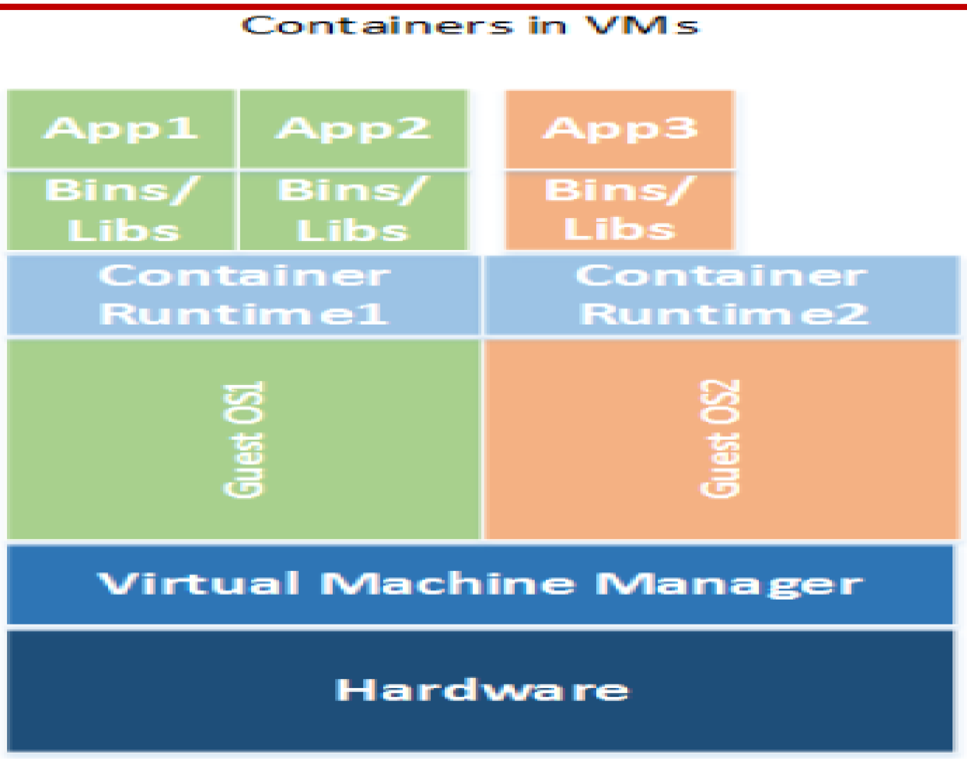
Persistence

- 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



Overview

- Container definition
- **Layers and deployment**
- Container repositories
- Clusters and orchestration
- Serverless Architecture

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

- 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

Overview

- Container definition
- Layers and deployment
- **Container repositories**
- Clusters and orchestration
- Serverless Architecture

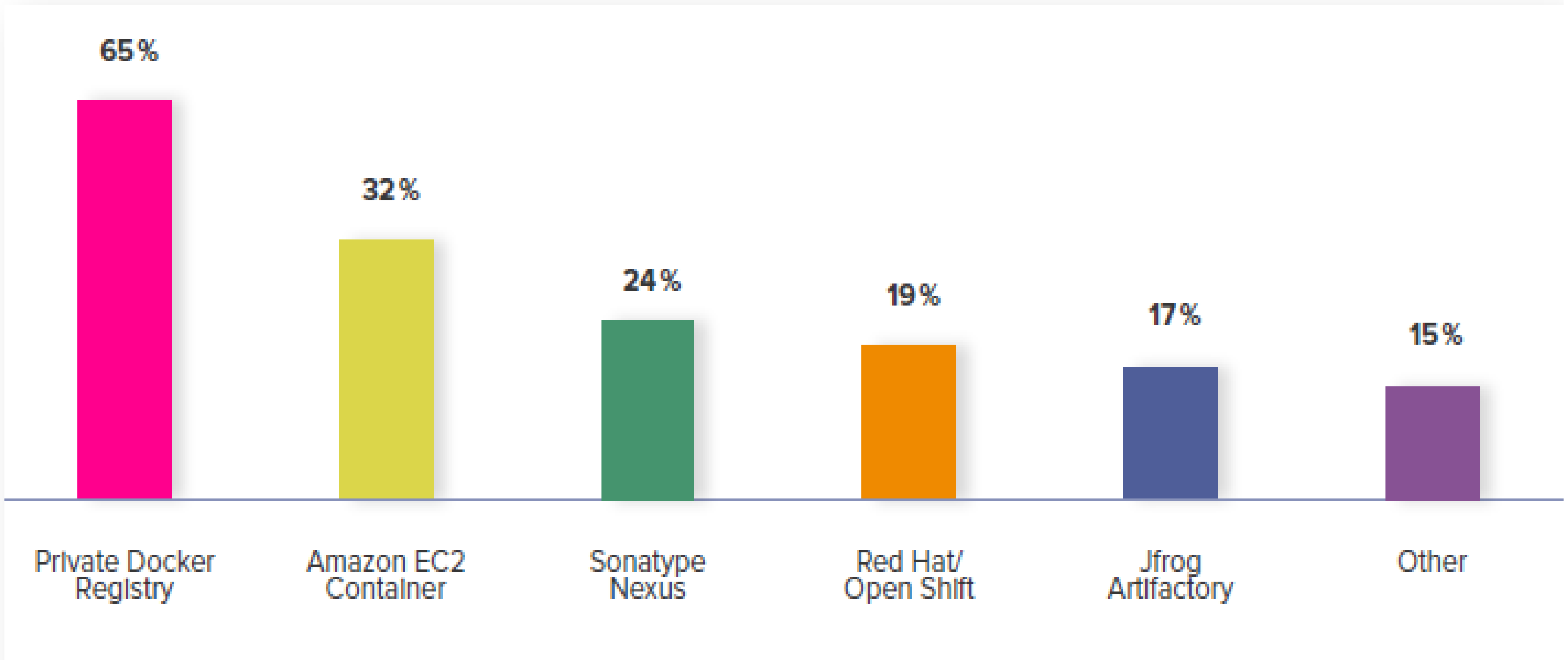
Container repositories

- 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

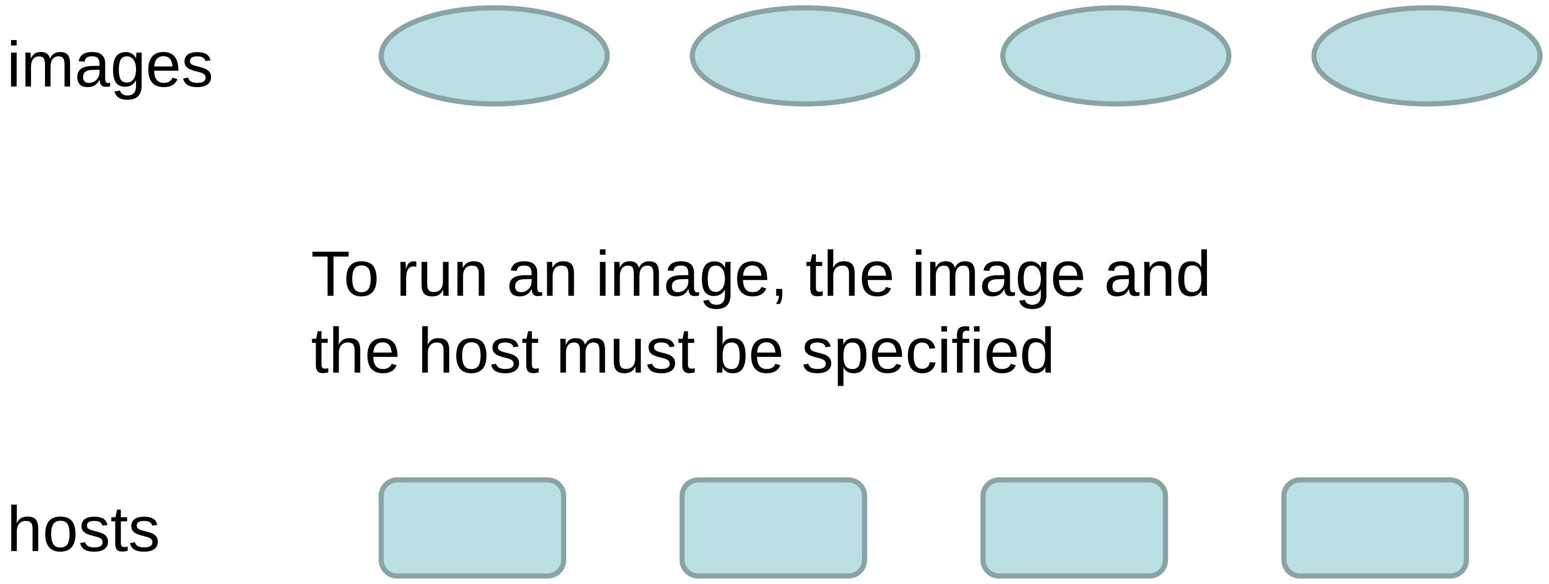


DevOps Community Survey 2018 - Sonatype

Overview

- Container definition
- Layers and deployment
- Container repositories
- **Clusters and orchestration**
- Serverless Architecture

Allocation of images to hosts



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

With basic Docker this allocation must be done manually

Container orchestrator

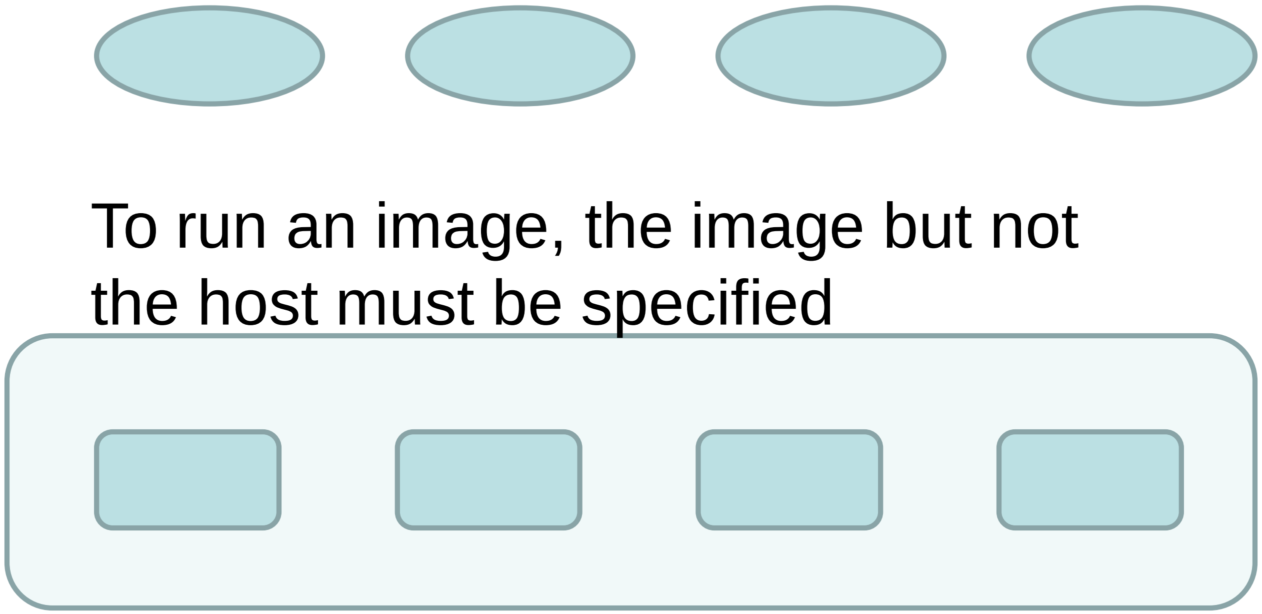


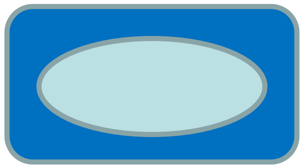
image
To run an image, the image but not the host must be specified
Orchestrator encapsulates hosts into a collection

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

Collection Master

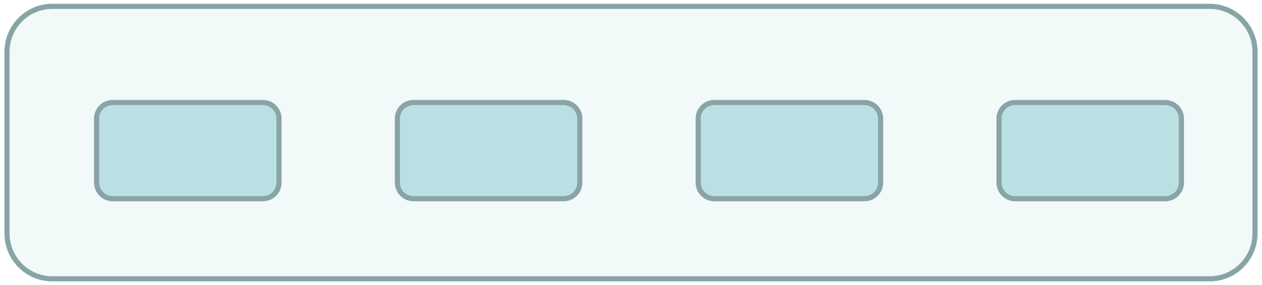
image

Run request is sent to collection master which selects host



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

Collection

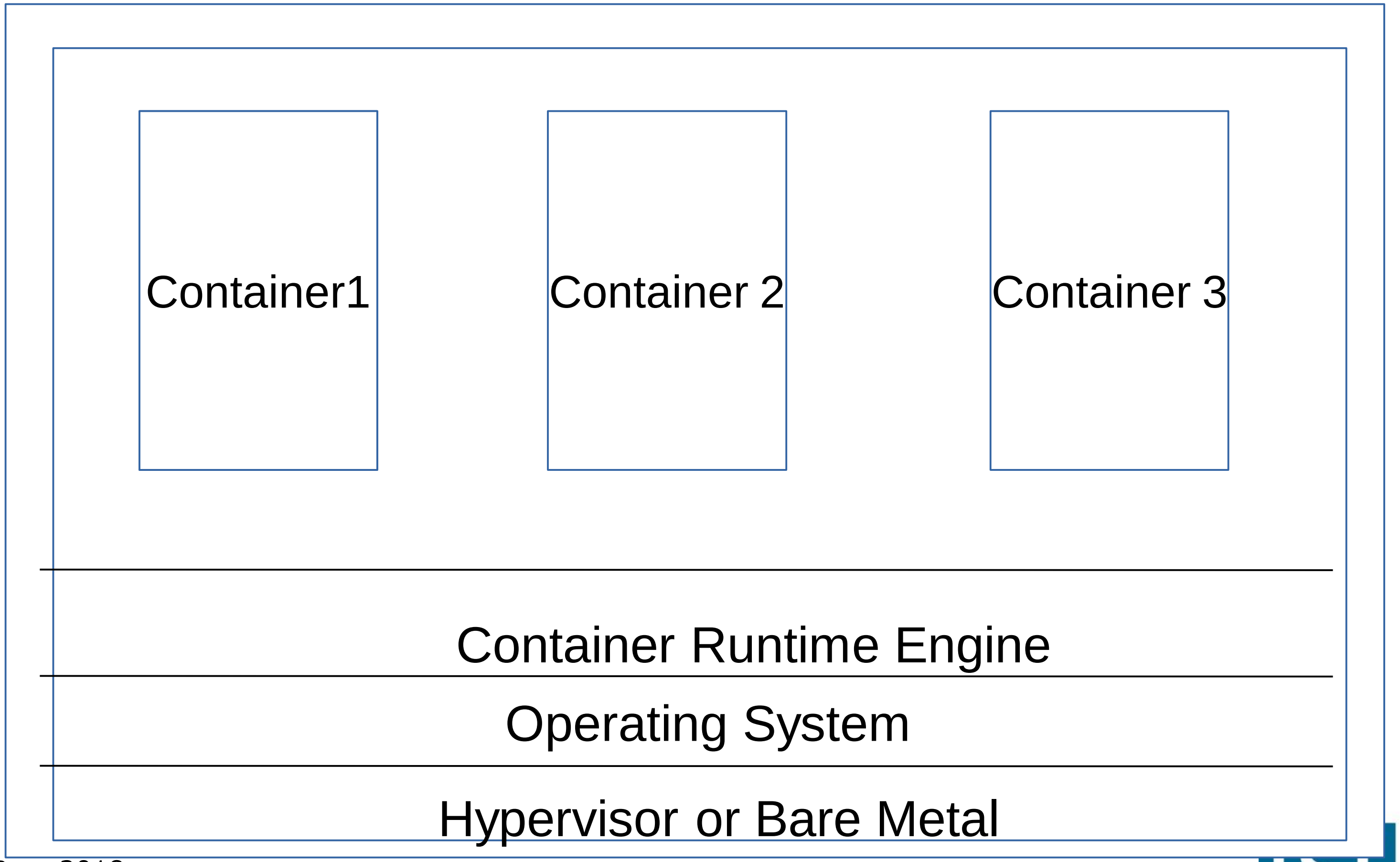


Grouping containers

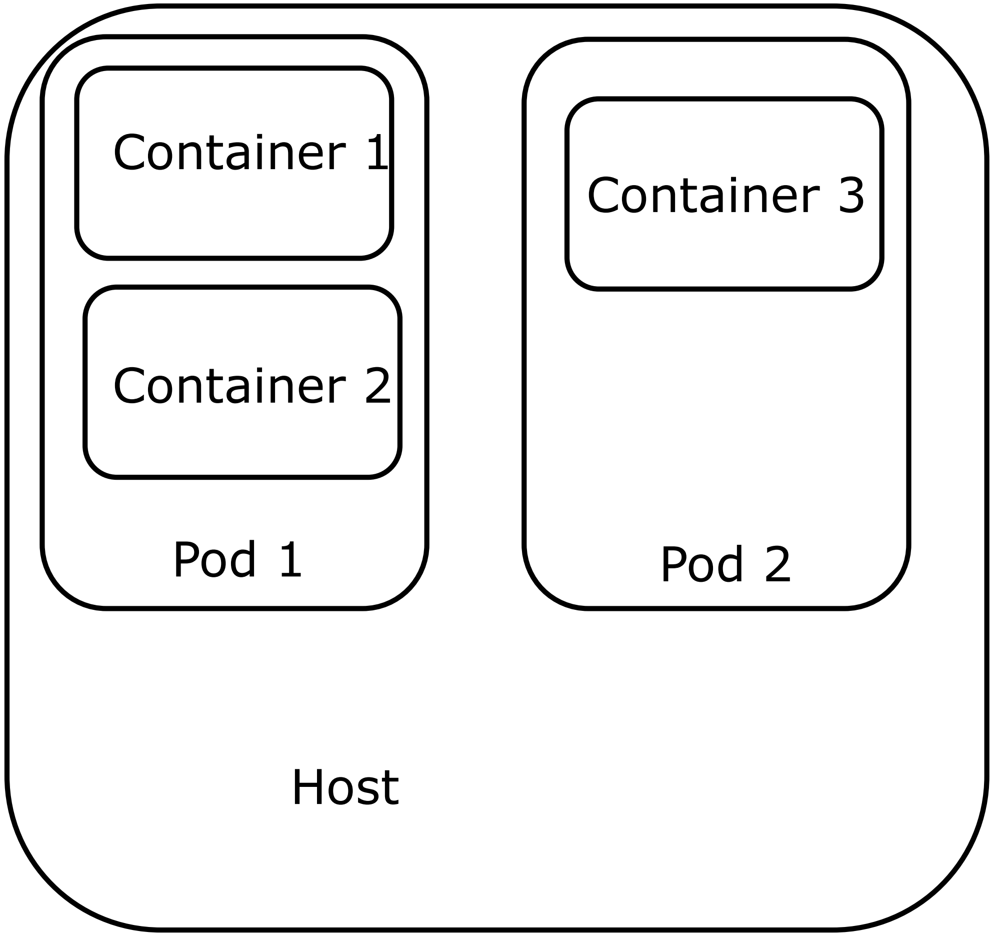
- Up to this point, containers are arbitrarily assigned to available host without regard for communication among containers.
- Suppose two containers communicate frequently. E.g. app container and logging containers. Then you would like them to be allocated into same host to reduce communication time.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

Carnegie Mellon University

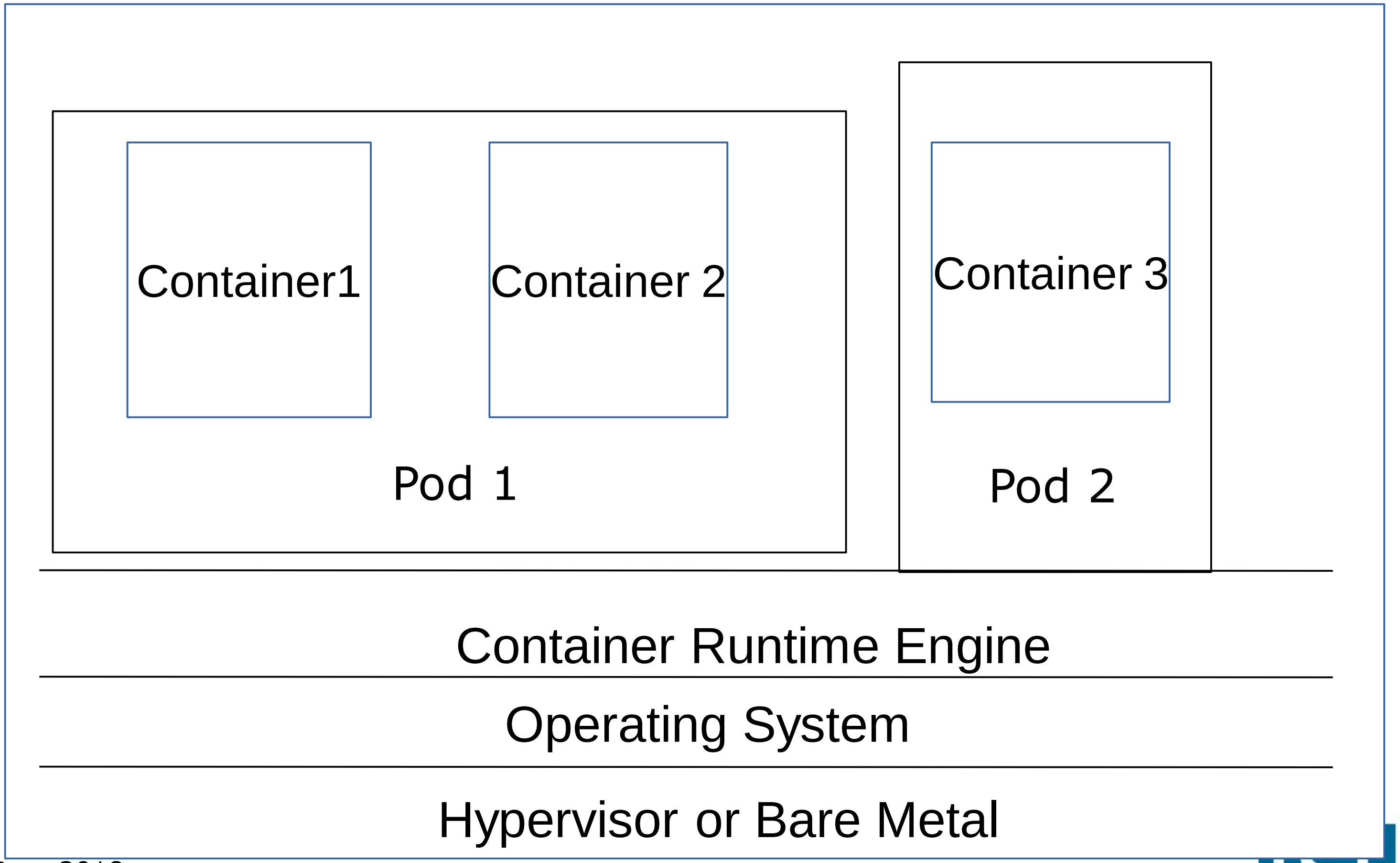
Container hierarchy (again)



Adding pods to hierarchy



Revisiting container hierarchy



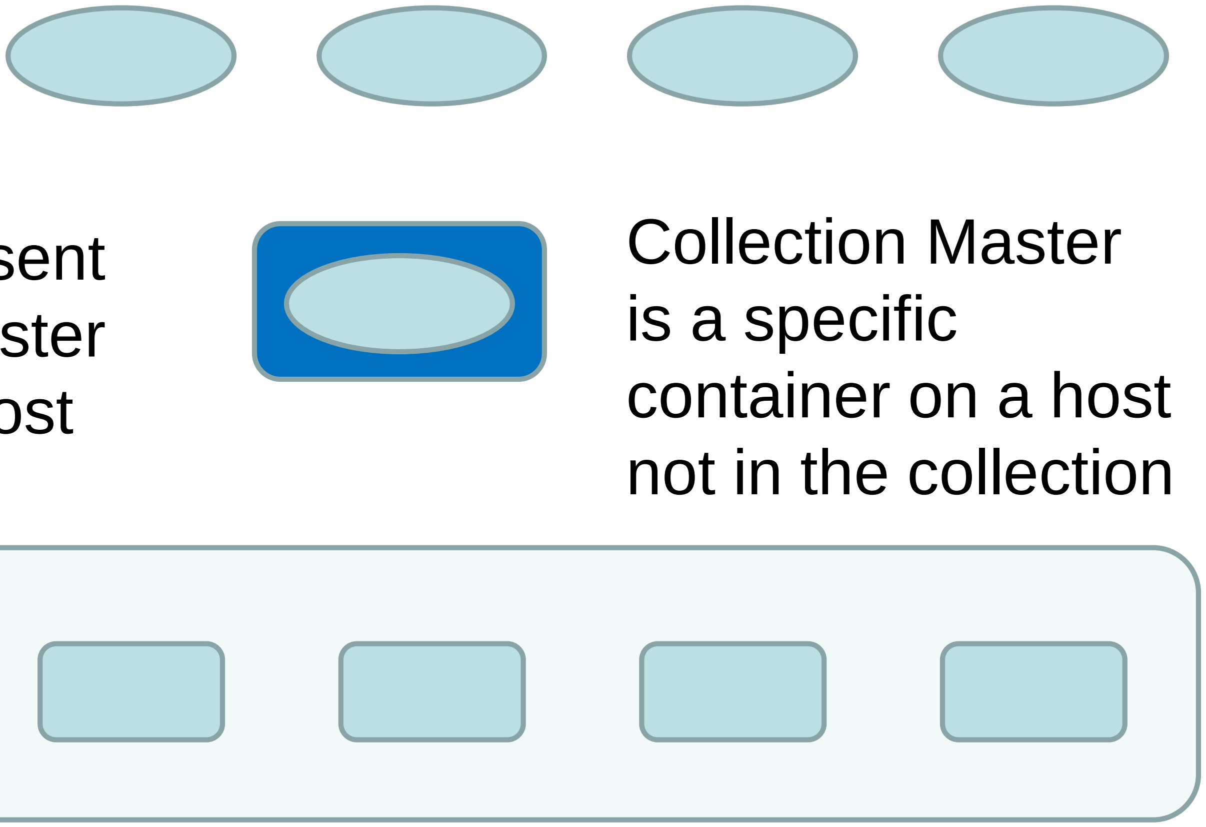
Collection Master revisited with pods

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

- A container is a lightweight virtual machine
- Docker allows building images in layers and deployment of a new version just requires deploying layers that have changed.
- 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.

Carnegie Mellon University

Docker Assignment

Instructions can be found at

https://github.com/cmudevops/workshop-instructions

institute for SOFTWARE RESEARCH

Carnegie Mellon

isr institute for SOFTWARE RESEARCH

Microservice Architecture

DevOps processes can be divided into three categories

1. Reduce errors during deployment
2. Reduce time to deploy
3. Reduce time to resolve discovered errors

- Microservice architecture helps with all three categories

Overview

- **Microservice architecture definition**
- Containers and microservices
- Reducing errors during deployment
- Reducing time to deploy
- Speeding up incident handling

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 [services] 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)

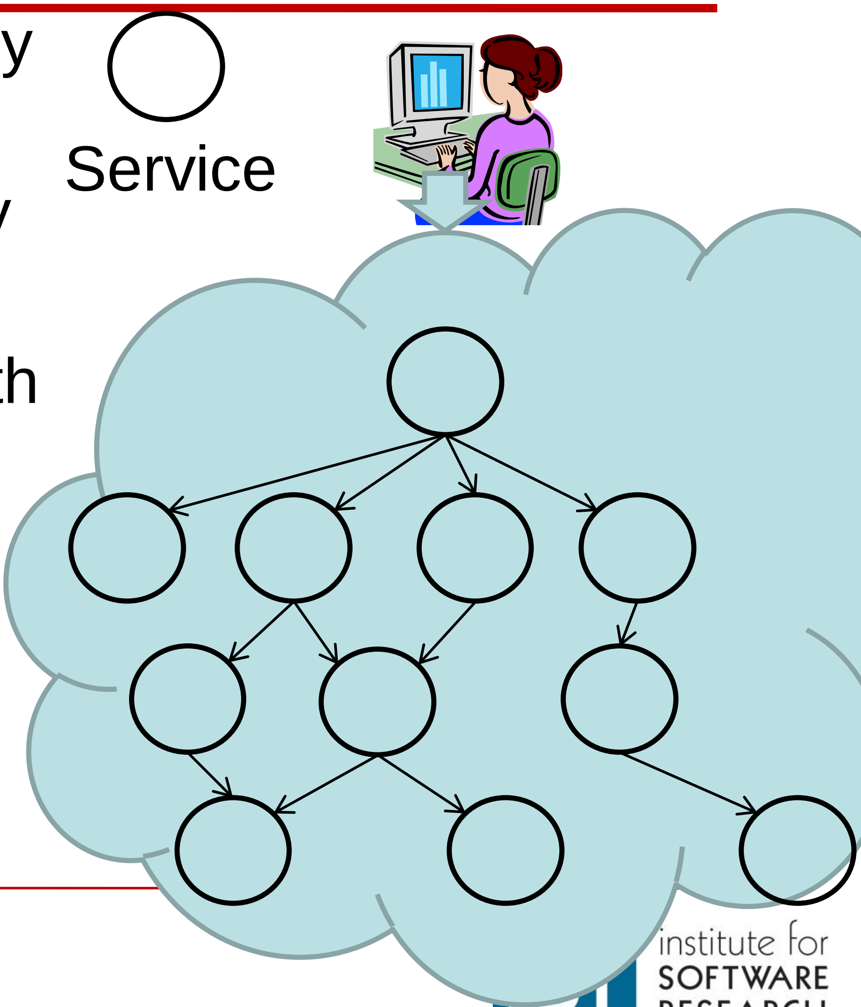
- The definition of microservice architecture sounds a lot like SOA.
- What is the difference?
- Amazon did not use the term “microservice architecture” when they introduced their rules. They said “this is SOA done right”

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)



Quality attributes for microservice architectures

- Availability (+)
- Modifiability (+/-)
- Performance (-)
- Reusability (-)
- Scalabilty (_+)

Carnegie Mellon University

Overview

- Microservice architecture definition
- **Containers and microservices**
- Reducing errors during deployment
- Reducing time to deploy
- Speeding up incident handling

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 container.
- Containers can be grouped in pods
- Also called service mesh
- Allows for deploying and scaling together

Overview

- Microservice architecture definition
- Containers and microservices
- **Reducing errors during deployment**
- Reducing time to deploy
- Speeding up incident handling

Carnegie Mellon University

Microservice architecture reduces errors during deployment

- One common source of errors during integration is inconsistency in technology choices
- E. g. Team A uses version 2.1 of a library, team B uses version 2.2. Two versions are incompatible.
- With microservice architecture,
  - each team makes their own technology choices.
  - Services communicate only via messages
  - No requirement for teams to use the same version of a library. Teams can even use different languages.
- Fewer errors

isr institute for SOFTWARE RESEARCH

Also

- No requirement for teams to coordinate to choose language, libraries, or dependent technologies
- Fewer meetings, shortens time to deployment.

Overview

- Microservice architecture definition
- Containers and microservices
- Reducing errors during deployment
- **Reducing time to deploy**
- Speeding up incident handling

Continuous Deployment

- Continuous deployment is process whereby after commit, software is tested and placed in production without human intervention
- Continuous development goes through the same steps except a human has to sign off on the actual deployment.
- In either case, time to deployment is shortened.

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.
- Must design to allow for these possibilities.

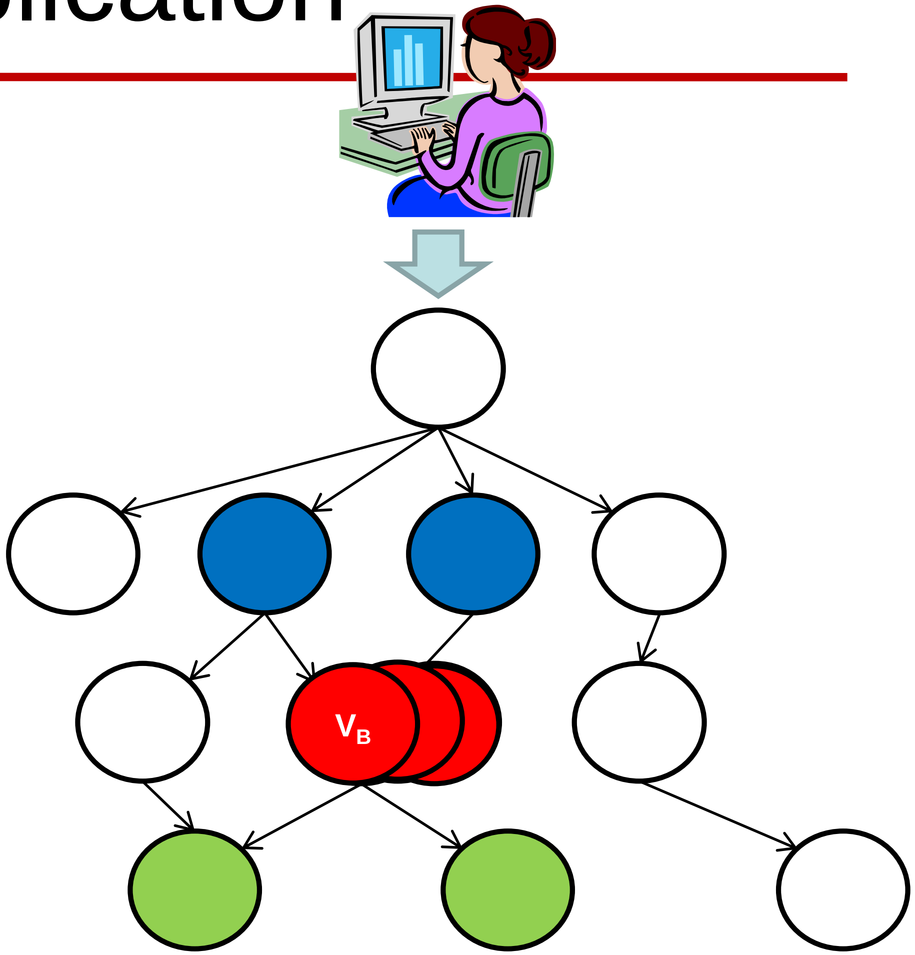
Situation

- 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.

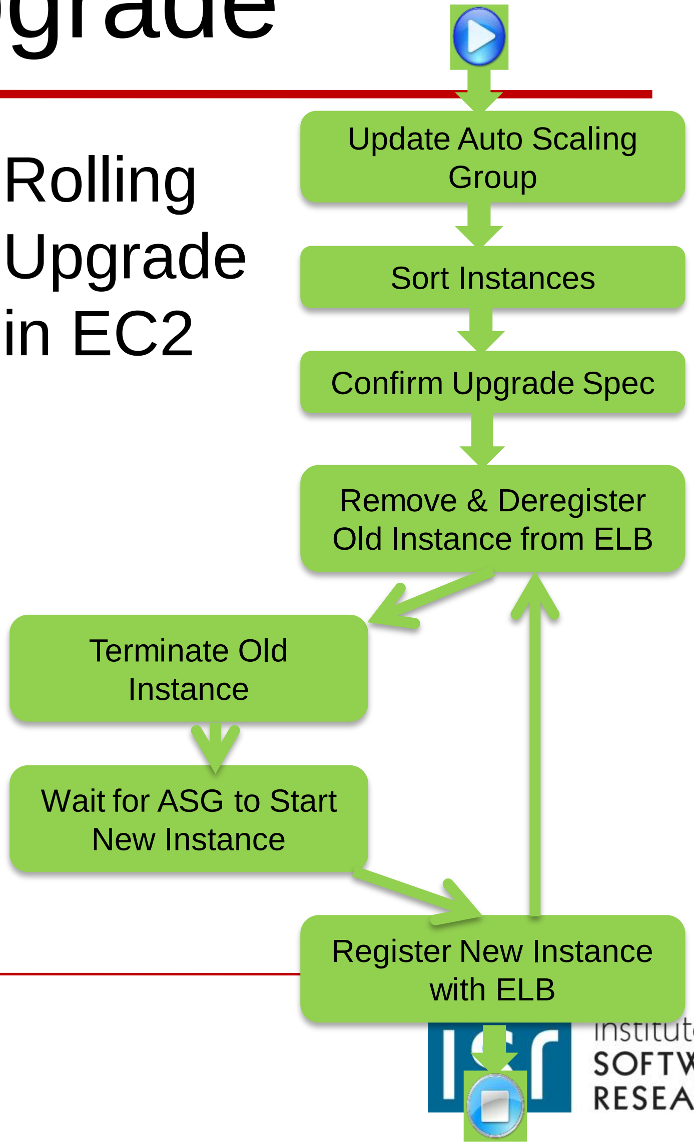
Deployment strategies

- Two basic all of nothing strategies
  - Blue/Green (also called Red/Black) – 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 – Blue/Green and Rolling Upgrade

- Blue/Green
  - 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


Update Auto Scaling Group
Sort Instances
Confirm Upgrade Spec
Remove & Deregister Old Instance from ELB
Terminate Old Instance
Wait for ASG to Start New Instance
Register New Instance with ELB

Problems to be dealt with

- Temporal inconsistency
- Interface mismatch

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 2.

Temporal inconsistency

- Can occur with either Blue/Green or rolling upgrade
- Prevented by using feature toggles.
- Feature toggle puts new code under control of if statement keyed to toggle.

```
If toggle then
        new code
    else
        old code
```

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 and forward compatible.

Overview

- Microservice architecture definition
- Containers and microservices
- Reducing errors during deployment
- Reducing time to deploy
- **Speeding up incident handling**

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."

-*Wener 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.

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

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
  - …

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)

Logs on Entry/Exit

- Protocol Buffer compilers 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

Repository

- Logs and metrics are placed in 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.

Summary

- Time to market is driver for processes to
  - Reduce errors during deployment
  - Reduce time to deployment
  - Respond to incidents more quickly
- Microservice architecture helps all three goals
- Continuous deployment/development requires designing to avoid various kinds of inconsistencies
- Developers carry pagers as a means to speed up incident handling

More Information

Deployment and Operations for Software Engineers

by

Len Bass and John Klein