Carnegie Mellon

Don’t forget about deployment and operations

Len Bass

Overview

- **Motivation**
- Microservice architecture
- Reducing errors during deployment
- Reducing time to deploy
- Speeding up incident handling

What is software engineering?

- "the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software"—The U.S Bureau of Labor Statistics—IEEE *Systems and software engineering*

- By this definition software engineers are NOT responsible for deployment and operations

NOT TRUE any more

- Time to market pressures have made software engineers responsible for portions of deployment and operations.

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

Traditional 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
- Each team completes its code and their job is finished.

8

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 errors that occur.

9

What else is wrong?

- Incident handling is too slow. Traditional process
  - Incident reported to operations
  - Ticket is generated
  - Ticket goes (eventually) to developers
  - Developers fix problem and schedule fix for release.

Dealing with errors

- Errors can be
  - Prevented
    - Coordination among teams
    - Architecture design
    - Process
  - Repaired
- Coordination (meetings) is time consuming
- Repairing errors is time consuming

Leads to 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

- Microservice architecture helps with all three categories

Overview

- Motivation
- **Microservice architecture**
- 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)

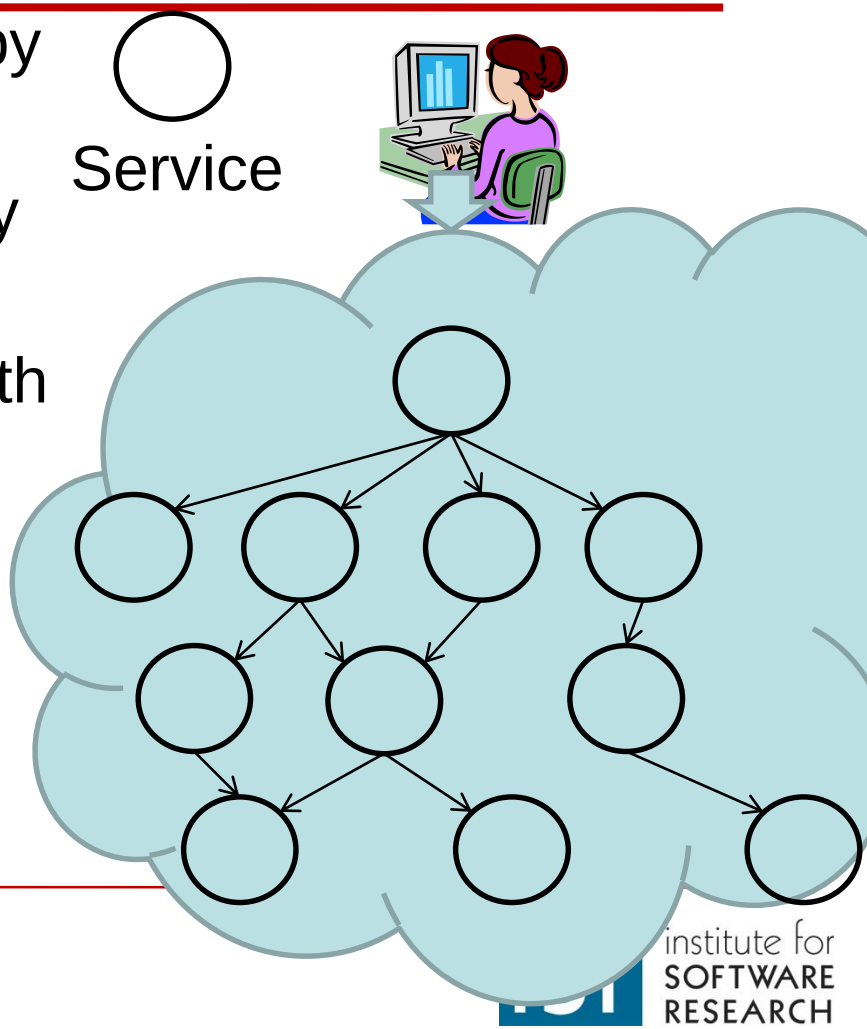
- The definition of microservice architecture sounds a lot like SOA.
- What is the difference?
- Amazon did not use the term “microservice architecture” when they introduced their rules. They said “this is SOA done right”

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 (-)

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

Overview

- Motivation
- Microservice architecture
- **Reducing errors during deployment**
- Reducing time to deploy
- Speeding up incident handling

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

Also

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

Overview

- Motivation
- Microservice architecture
- 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

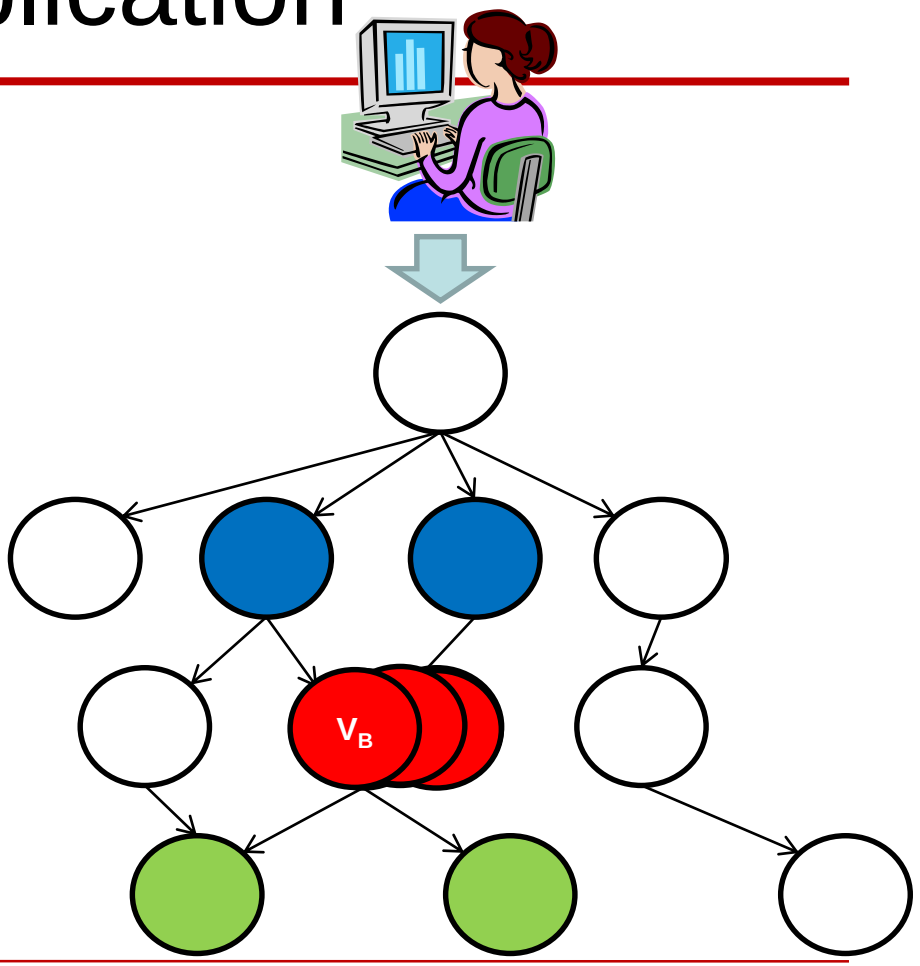
- Your application is executing
  - Multiple independent deployment units
  - Some of these deployment units may have multiple instances serving requests
- You have a new version of one of the deployment units to be placed into production
- An image of the new version is on the staging server or in a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

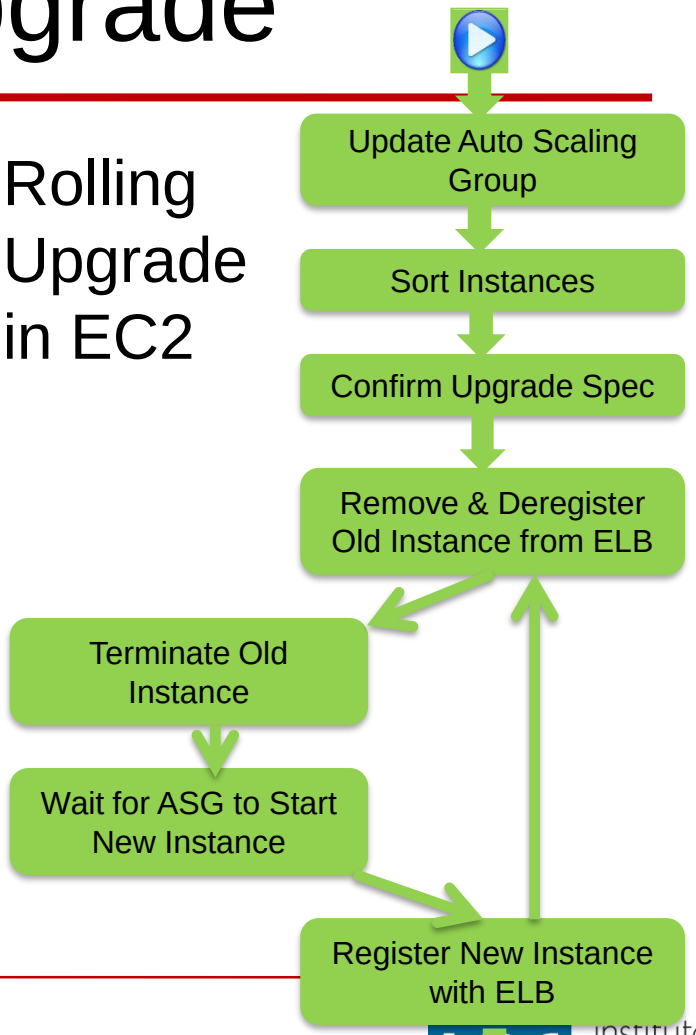
Deployment strategies

- 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



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 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

- Motivation
- Microservice architecture
- 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

Feedback/questions

- Deployment and Operations for Software Engineers
- Available from Amazon