DevOps
Part I – what is DevOps?

Len Bass

"This project is vital to our company. How long will it take?"

Day 1

"Its taking too long!!! "

Day 30

"You Are Fired!"

Day 60

Where Does the Time Go?

- As Software Architects our view is that there are the following activities in software development
  - Concept
  - Requirements
  - Design
  - Implementation
  - Test
- Code Complete
- Different methodologies will organize these activities in different ways.
- Agile focuses on getting to Code Complete faster than with other methods.

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
- DevOps is a movement intended to reduce the time between code complete and code in production.

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 process, architecture, tools, and business.

*DevOps: A Software Architect's Perspective

5 Categories of DevOps processes

- Treat operators as first class citizens
- Make Dev more responsible for incident handling
- Enforce deployment practices uniformly across both dev and ops
- Use continuous deployment
- Develop infrastructure code using same processes as application code

- We are focusing on number 4 in this session

Why is Deployment so Time Consuming?

- Errors in deployed code are a major source of outages.
- So much so that organizations have formal release plans.
- There is a position called a "Release Engineer" that has responsibility for managing releases.

Release plan

1. Define and agree release and deployment plans with customers/stakeholders.

2. Ensure that each release package consists of a set of related assets and service components that are compatible with each other.

3. Ensure that integrity of a release package and its constituent components is maintained throughout the transition activities and recorded accurately in the configuration management system.

4. Ensure that all release and deployment packages can be tracked, installed, tested, verified, and/or uninstalled or backed out, if appropriate.

5. Ensure that change is managed during the release and deployment activities.

6. Record and manage deviations, risks, issues related to the new or changed service, and take necessary corrective action.

7. Ensure that there is knowledge transfer to enable the customers and users to optimise their use of the service to support their business activities.

8. Ensure that skills and knowledge are transferred to operations and support staff to enable them to effectively and efficiently deliver, support and maintain the service, according to required warranties and service levels

*http://en.wikipedia.org/wiki/Deployment_Plan

Look at one requirement

2. Ensure that each release package consists of a set of related assets and service components that are compatible with each other.
   - Every development team contributing to the release must have completed their code
   - Every development team must have used the same version of every supporting library
   - The development teams must have agreed on a common set of supporting technologies
- Every item requires coordination among developers
  - Meetings
  - Documents
- TIME

How to Speed Up Deployment

- Set up a process and an architecture so that development teams do not need to coordinate with each other

- Code Complete ➡ Code in Production

Three facets of getting code into production quickly

- Process – continuous deployment (Sascha Bates)

- Architecture - Microservice architecture (Sam Newman)

- Deployment issues (Len Bass)

- Book raffle at the end of the session
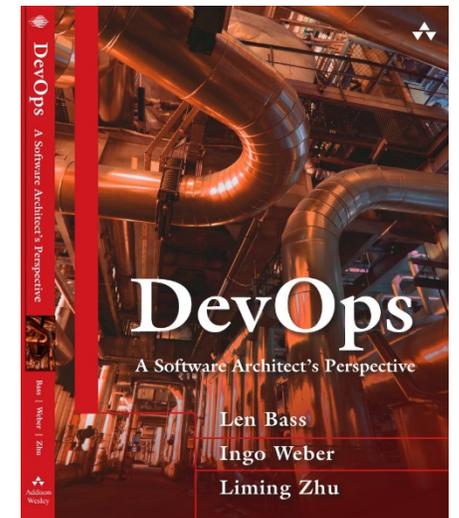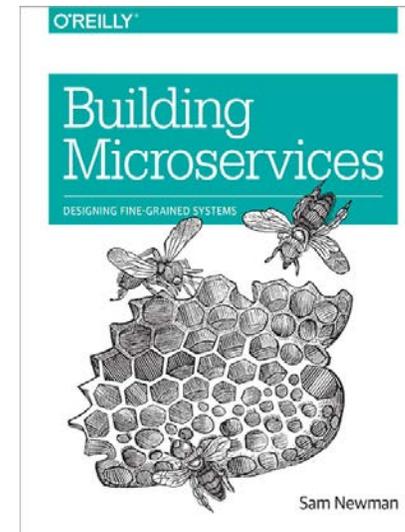
DevOps
# Part 2 – Upgrade Issues

Len Bass

One more thing

- Having a continuous deployment pipeline and a microservice architecture does not end your problems.

- In this section, we will look at issues associated with deployment

Deploying a new version of a service

Multiple instances of a service are executing

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

UAT / staging / performance tests
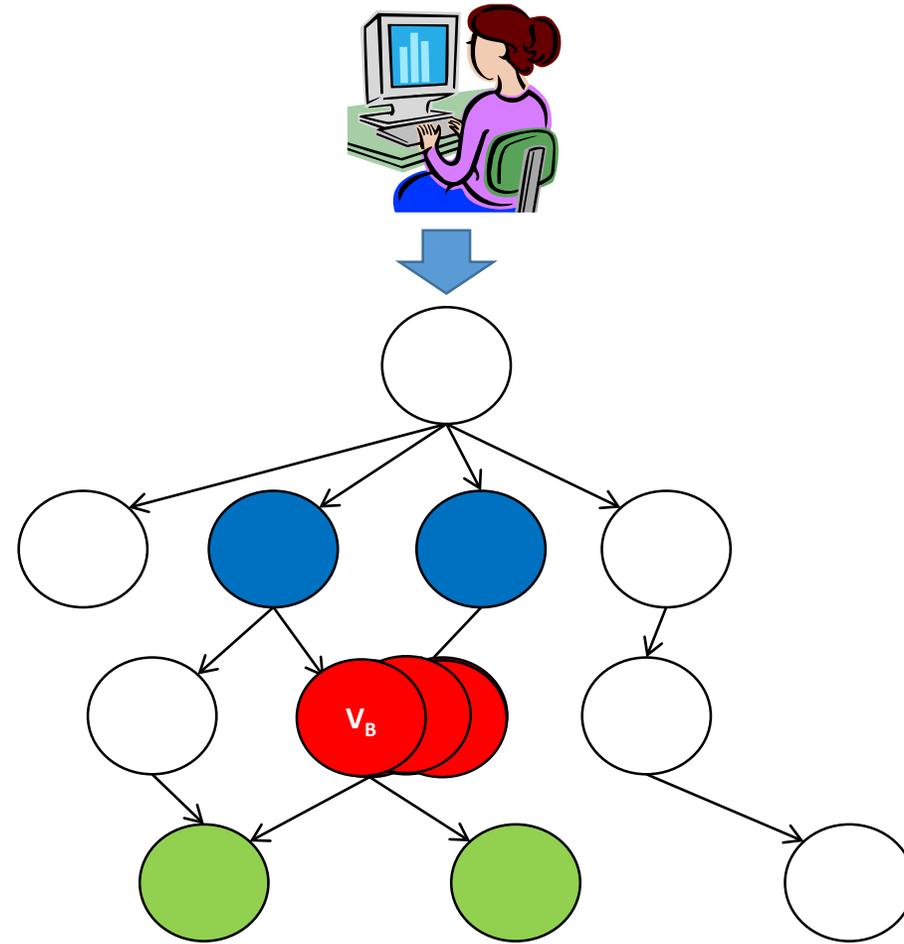
Deployment goal and constraints

- 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)
  - Service to clients must be maintained while the new version is being deployed.
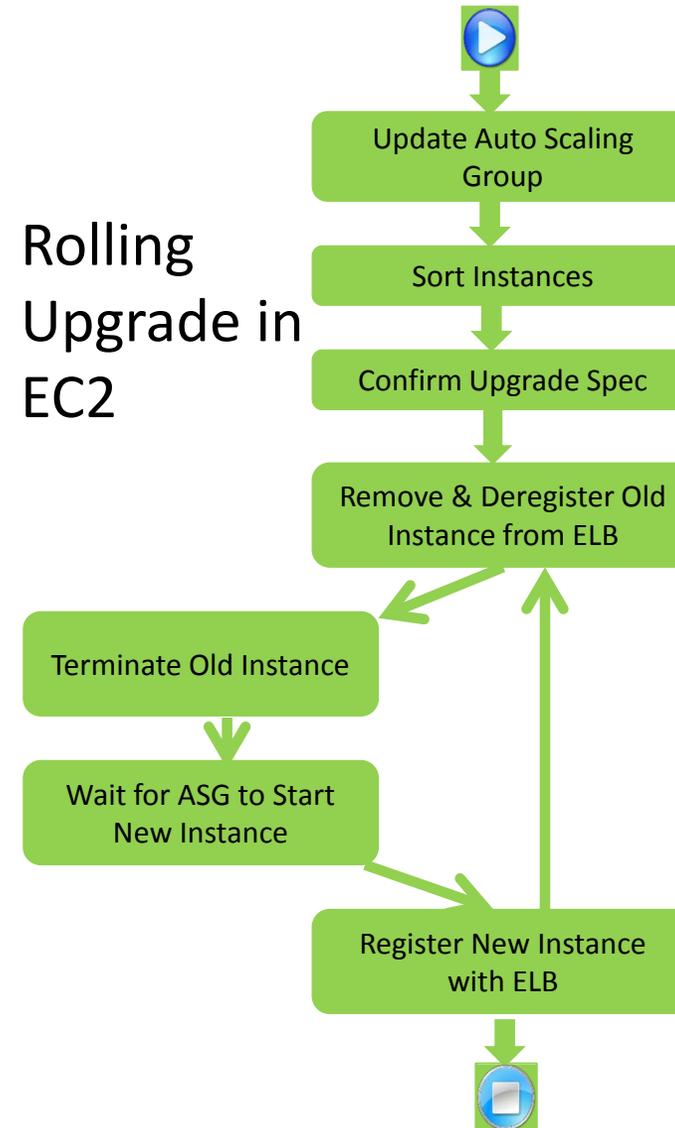
Deployment strategies

- Two basic all of nothing strategies
  - 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 – 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

```
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
```

6

What are the problems with Rolling Upgrade?

- Any development team can deploy their service at any time.
- Four concerns
  - Maintaining consistency between different versions of the same service when performing a rolling upgrade
  - Maintaining consistency among different services
  - Maintaining consistency between a service and persistent data
  - Rollback

Maintaining consistency between different versions of the same service

- Key idea – differentiate between *installing* a new version and *activating* a new version

- Involves "feature toggles" (described momentarily)

- Sequence
  - Develop version B with new code under control of feature toggle
  - Install each instance of version B with the new code toggled off.
  - When all of the instances of version A have been replaced with instances of version B, activate new code through toggling the feature.

Issues

- What is a feature toggle?
- How do I manage features that extend across multiple services?
- How do I activate all relevant instances at once?

Feature toggle

- Place feature dependent new code inside of an "if" statement where the code is executed if an external variable is true. Removed code would be the "else" portion.

- Used to allow developers to check in uncompleted code. Uncompleted code is toggled off.

- During deployment, until new code is activated, it will not be executed.

- Removing feature toggles when a new feature has been committed is important.

Multi servicefeatures

- Most features will involve multiple apps.
- Each app has some code under control of a feature toggle.
- Activate feature when all instances of all apps involved in a feature have been installed.
  - Maintain a catalog with feature vs service version number.
  - A feature toggle manager determines when all old instances of each version have been replaced. This could be done using registry/load balancer.
  - The feature manager activates the feature.
  - **Archaius** is an open source feature toggle manager.

Maintaining consistency among different services

- Use case:
  - Wish to deploy new version of service A without coordinating with development team for clients of service A.
    - I.e. new version of service A should be backward compatible in terms of its interfaces.
    - May also require forward compatibility in certain circumstances, e.g. rollback

Maintaining consistency between a serviceand persistent data

- Assume new version is correct. Rollback discussed in a minute.

- Inconsistency in persistent data can come about because data schema or semantics change.

- 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 apps that access data.

Summary of consistency discussion so far.

- Feature toggles are used to maintain consistency within instances of a service

- Disallowing modification of schema will maintain consistency between services and persistent data.

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.

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 making registry/load balancer canary aware.

A/B testing

- Suppose you wish to test user response to a system variant. E.g. UI difference or marketing effort. A is one variant and B is the other.

- You simultaneously make available both variants to different audiences and compare the responses.

- Implementation is the same as canary testing.

Rollback

- New versions of an app 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 or roll forward is almost never automated because there are multiple factors to consider.
  - Forward or backward recovery
  - Consequences and severity of problem
  - Importance of upgrade

Summary

- Speeding up deployment time will reduce time to market
- Continuous deployment is a technique to speed up deployment time
- Microservice architecture is designed for minimizing coordination needs and allowing independent deployment
- Multiple simultaneous versions managed with feature toggles and backward/forward compatibility.
- Feature toggles support rollback, canary testing, and A/B testing.

Raffle – drop your business card into the container