架构迎接未来变化

IAS2017 • NANJING

Designing for Deployment

Len Bass

Outline

- Release process
- Microservice architecture
- Deployment strategies
- Other issues

Why Does Placing a System into Production Take so Long?

- One reason is that changes are packaged into releases.
- All projects have to be combined to make up a release
- Releases are scheduled

Managing releases

- Releases are stressful.
- Releases take careful management
  - 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.

Suppose releases did not have to be scheduled?

- Release when a code segment is complete
- Small releases
- Less stress
- Less waiting for a release to be complete

⇒Shorter time to market for new features, fixes

- Happier stakeholders!

Ad hoc releases exist

- Many companies now release to production multiple times per day.
  - 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

Replace management discipline over release by engineering discipline

- The management discipline that went into release planning and execution is replaced by
  - Engineering process discipline
  - Architecture techniques
  - Tool support

Deployment

- Much of the current software engineering focus is on completing code
- But … 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
- One purpose of release planning is to deploy code without errors

Modern Deployment Processes

Process	Architecture techniques	Tools
Continuous Deployment	• Microservice architecture • Backward/Forward compatability • Feature toggles	• Management tools • Deployment pipeline tools • Configuration management tools
Post deployment testing	• Reliability tactics • Pedigreed testing • Initialization testing • Log generation	• Fault injection tools • Locality tools • Performance monitors • Janatorial tools

Outline

- Release process
- Microservice architecture
- Deployment strategies
- Other issues

~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 what has come to be called "Microservice Architecture".
- (Its really an architectural style).

In Addition

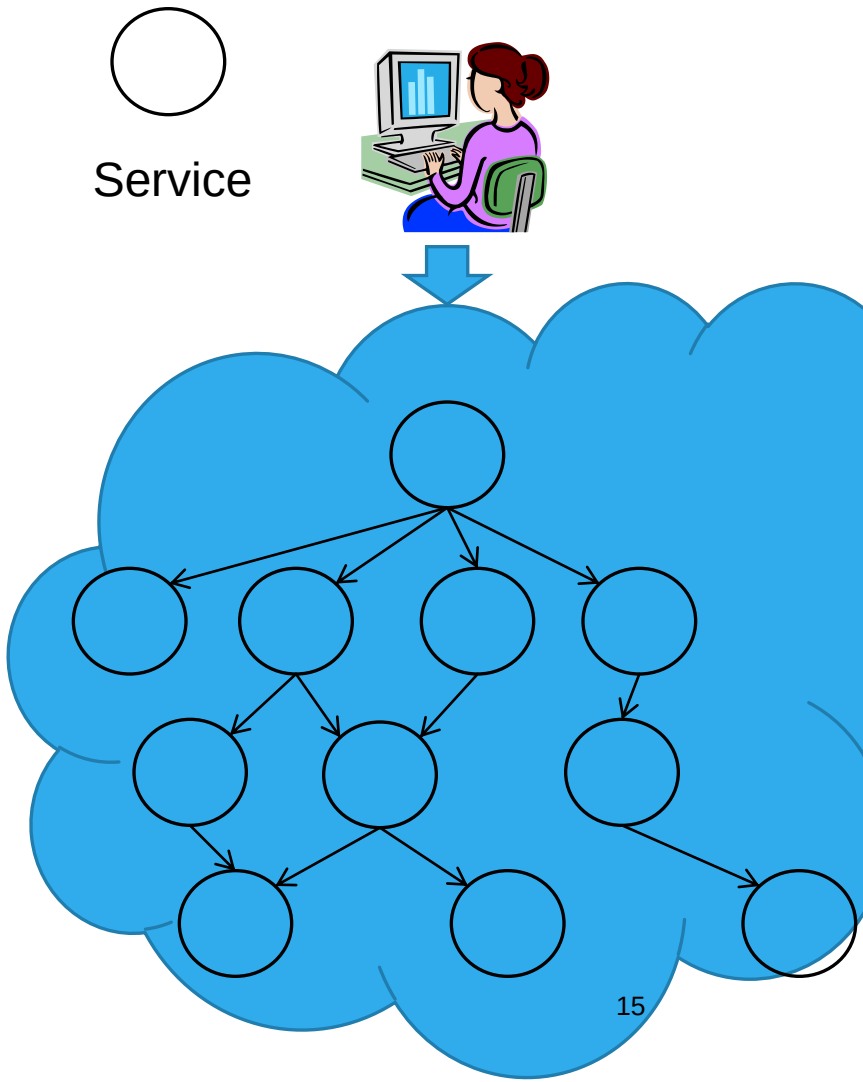
- Amazon has a “two pizza” rule.
- No team should be larger than can be fed with two pizzas (~7 members).
- Each (micro) service is the responsibility of one team
- This means that microservices are small and intra team bandwidth is high
- Large systems are made up of many microservices.
- There may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

Microservice architecture supports continuous deployment

- Two topics:
  - What is microservice architecture?
  - What are the deployment issues and how do I deal with them?

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)



Relation of teams and services

- Each service is the responsibility of a single development team
- Individual developers can deploy new version without coordination with other developers.
- It is possible that a single development team is responsible for multiple services

Questions about Micro SOA

- /Q/ Isn't it possible that different teams will implement the same functionality, likely differently?
- /A/Yes, but so what? Major duplications are avoided through assignment of responsibilities to services. Minor duplications are the price to be paid to avoid necessity for synchronous coordination.
- /Q/ what about transactions?
- /A/ Micro SOA privileges flexibility above reliability and performance. Transactions are recoverable through logging of service interactions. This may introduce some delays if failures occur.

Microservice architecture supports continuous deployment

- Two topics:
  - What is microservice architecture?
  - What are the deployment issues and how do I deal with them?

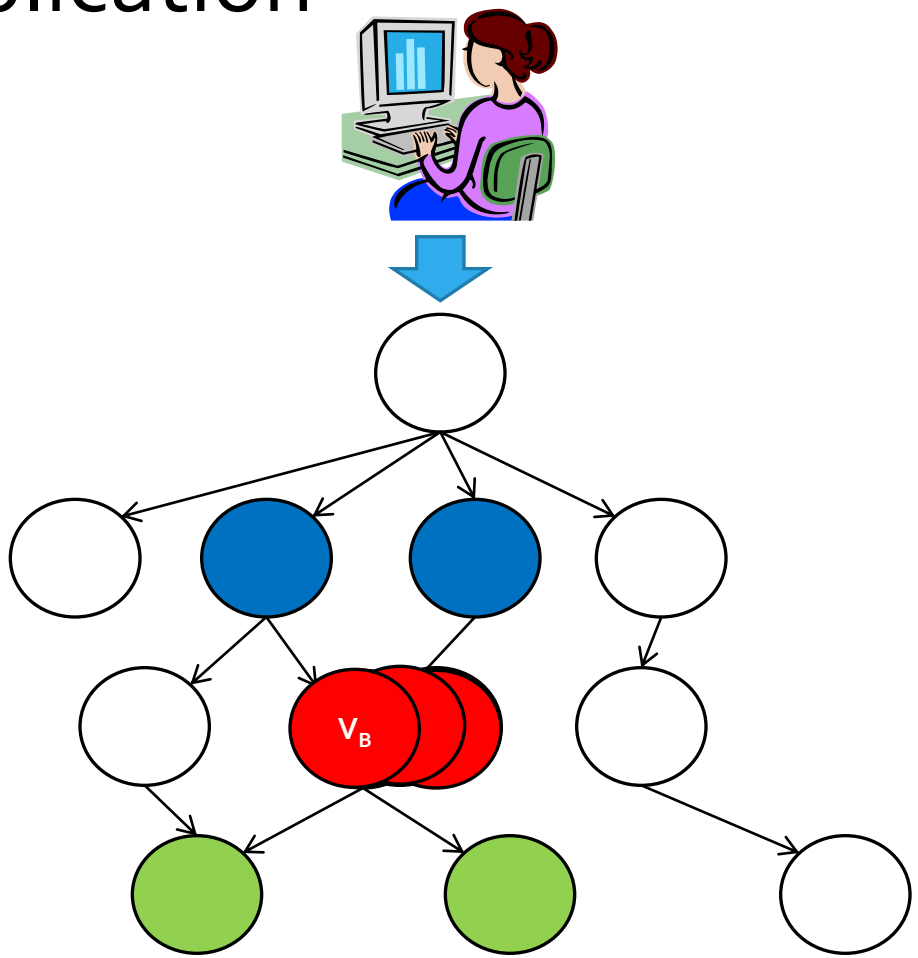
Outline

- Release process
- Microservice architecture
- Deployment strategies
- Other issues

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



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.

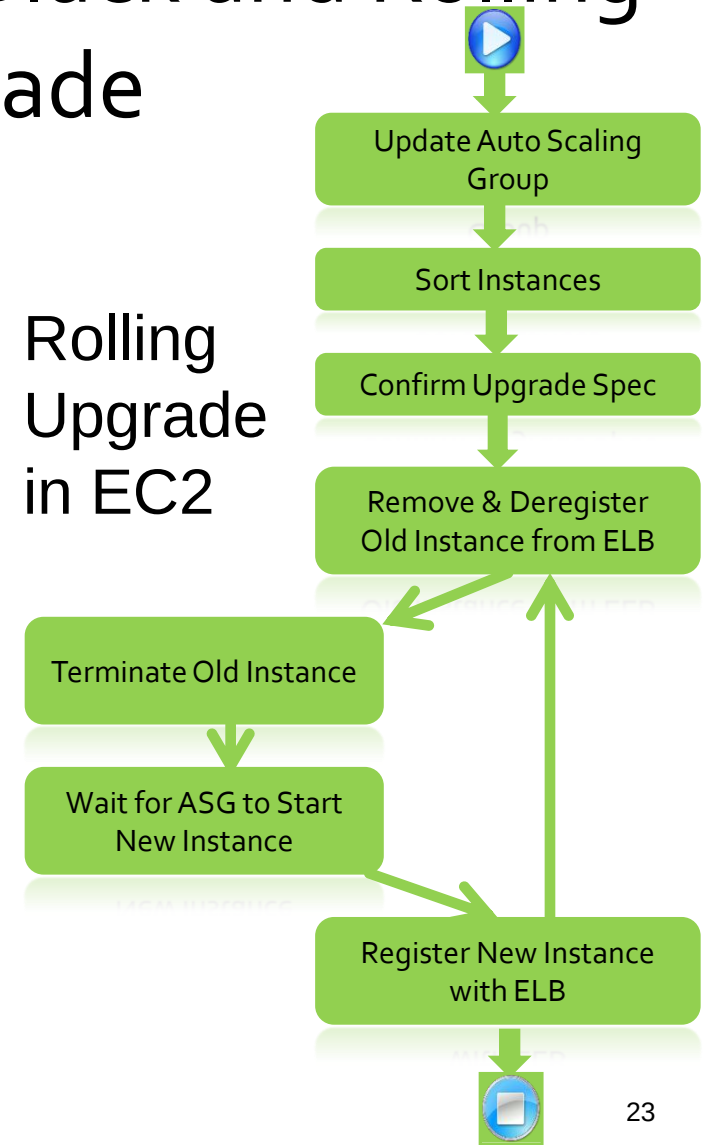
Deployment strategies

- 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



What are the problems with Rolling Upgrade?

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

Maintaining consistency

- 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 apps?
- 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 service features

- Most features will involve multiple services.
- Each service has some code under control of a feature toggle.
- Activate feature when all instances of all services 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.

Activating feature

- The feature toggle manager changes the value of the feature toggle. Two possible techniques to get new value to instances.
  - Push. Broadcasting the new value will instruct each instance to use new code. If a lag of several seconds between the first service to be toggled and the last can be tolerated, there is no problem. Otherwise synchronizing value across network must be done.
  - Pull. Querying the manager by each instance to get latest value may cause performance problems.
- A coordination mechanism such as Zookeeper will overcome both problems.

Outline

- Release process
- Microservice architecture
- Deployment strategies
- Other issues

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.
- Similar in concept 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 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 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 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.
- Feature toggles support rollback, canary testing, and A/B testing.

More Information

Contact lenbass@cmu.edu

DevOps: A Software Architect's Perspective is available from your favorite bookseller