Carnegie Mellon

institute for SOFTWARE RESEARCH

DevOps – Continuous Integration

Deployment Strategies

1

Carnegie Mellon University

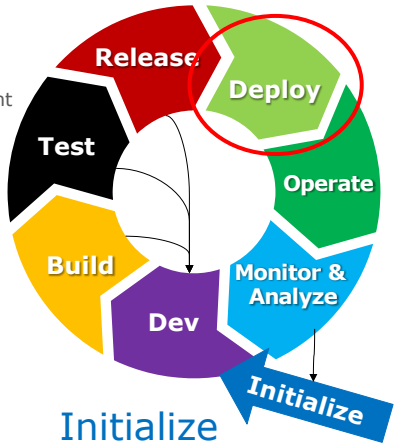
DEVOPS PROCESSES

Release
Approve for deployment

Test
Automate tests as much as possible

Build
Create an executable artifact

Dev
Perform normal development activities



Initialize
Select architecture to support other activities
Create scripts for other activities

Deploy
Move into production environment

Operate
Execute system and gather measurements about its operation

Monitor & Analyze
Display measurements taken during operation & analyze the data

institute for SOFTWARE RESEARCH

2

Carnegie Mellon University

Overview

- **Deployment strategies**
- Temporal inconsistency
- Interface mismatch
- Data inconsistency
- Rollback
- Partial Deployments

Carnegie Mellon University

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
- You do not want to interrupt service

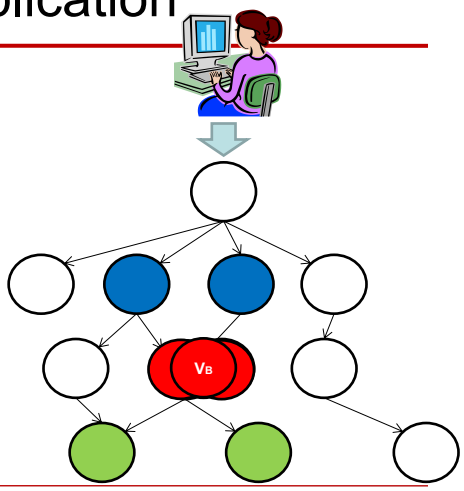
Carnegie Mellon University

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



Carnegie Mellon University

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 that 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.

Carnegie Mellon University

Deployment strategies

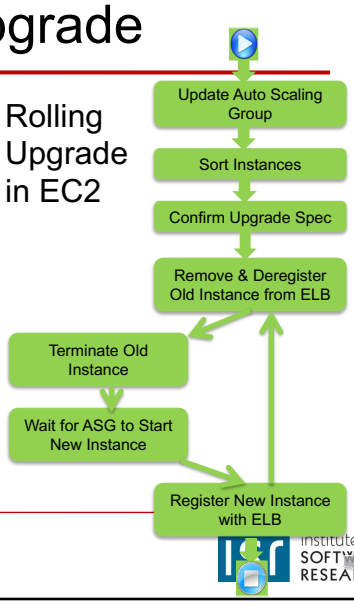
- Two basic all of nothing strategies
  - Red/Black (also called Blue/Green) – leave N instances with version A as they are, allocate and provision N instances with version B and then switch to version B and release instances with version A.
  - Rolling Upgrade – allocate one instance, provision it with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

Carnegie Mellon University

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



Carnegie Mellon University

Overview

- Deployment strategies
- **Temporal inconsistency**
- Interface mismatch
- Data inconsistency
- Rollback
- Partial Deployments

Carnegie Mellon University

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 B does discounts per shopping cart.
- Client C's first call goes to version A and its second call goes to version B.
- Results in inconsistent discounts being calculated.
- Caused by update occurring between call 1 and call 2.

Carnegie Mellon University

Temporal inconsistency

- Can occur with either Blue/Green or rolling upgrade
- Prevented by using feature toggles or service mesh.
- We will discuss feature toggles now and service mesh later.

Carnegie Mellon University

Feature Toggle

- A feature toggle (also call feature flag) is a configuration parameter.
- Coded as:

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

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

Carnegie Mellon University

Preventing Temporal Inconsistency

- Write new code for version B under control of a feature toggle – initially toggled off
- Install N instances of version B 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 are running version B, activate the new code by turning the feature toggle on.

Carnegie Mellon University

Feature toggle manager

- There will be many different feature toggles
  - One for each feature across multiple services
- 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).

Carnegie Mellon University

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.

Carnegie Mellon University

Overview

- Deployment strategies
- Temporal inconsistency
- **Interface mismatch**
- Data inconsistency
- Rollback
- Partial Deployments

Carnegie Mellon University

Interface mismatch

- Suppose version B has a different interface from version A
- Then if Service C calls version B with an Interface designed for version A an interface mismatch occurs.
- Recall that Service A can be upgraded either before or after Service C.

Carnegie Mellon University

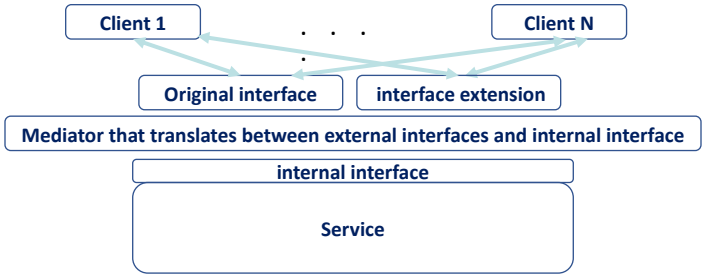
Forward and Backward Compatibility

- Services 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.

Carnegie Mellon University

Achieving Backwards Compatibility

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



Carnegie Mellon University

Overview

- Deployment strategies
- Temporal inconsistency
- Interface mismatch
- **Data inconsistency**
- Rollback
- Partial Deployments

Carnegie Mellon University

Two types of data consistency problems during upgrade

1. Persistent data
2. Transient data

Carnegie Mellon University

Maintaining consistency between a service and persistent data

- Assume new version is correct
- Inconsistency in persistent data can occur 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.

Carnegie Mellon University

I *really* must change the schema

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

Carnegie Mellon University

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.

Carnegie Mellon University

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.

Carnegie Mellon University

Overview

- Deployment strategies
- Temporal inconsistency
- Interface mismatch
- Data inconsistency
- **Rollback**
- Partial Deployments

Carnegie Mellon University

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).

Carnegie Mellon University

Automating rollback

- 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

Carnegie Mellon University

States of upgrade.

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

Carnegie Mellon University

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.

Carnegie Mellon University

Overview

- Deployment strategies
- Temporal inconsistency
- Interface mismatch
- Data inconsistency
- Rollback
- **Partial Deployments**

Carnegie Mellon University

Partial deployments

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

Carnegie Mellon University

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 more instances.
- Named after canaries in coal mines.
- Equivalent to beta testing for shrink wrapped software

Carnegie Mellon University

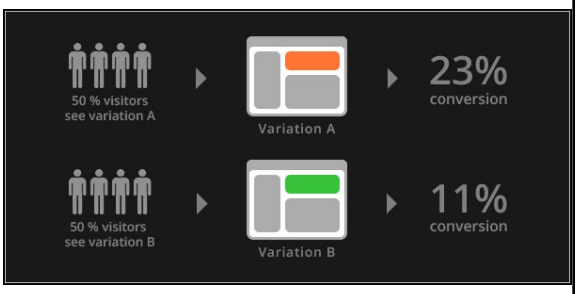
Implementation of canaries

- 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 or service mesh
  - Route messages from canary customers to canaries. Can be done through load balancer or DNS server.
- Measure performance metrics

Carnegie Mellon University

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



Carnegie Mellon University

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?

Carnegie Mellon University

Implementation

- The same as canary testing.
  - Use feature toggles or service mesh
- Measure business measure responses.

Carnegie Mellon University

Summary

- 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.
- Partial deployment strategies can be used for quality or marketing purposes.