Desmistificando Microsserviços e DevOps: Projetando Arquiteturas Efetivamente Escaláveis

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

[IF1004] - Seminários em SI 3
https://github.com/vinicius3w/if1004-DevOps

Licença do material

Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-CompartilhaIgual 3.0 Não Adaptada

Mais informações visite

http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt

AssertLab
Advanced Software and Systems Engineering Research Technologies

Centro de Informática

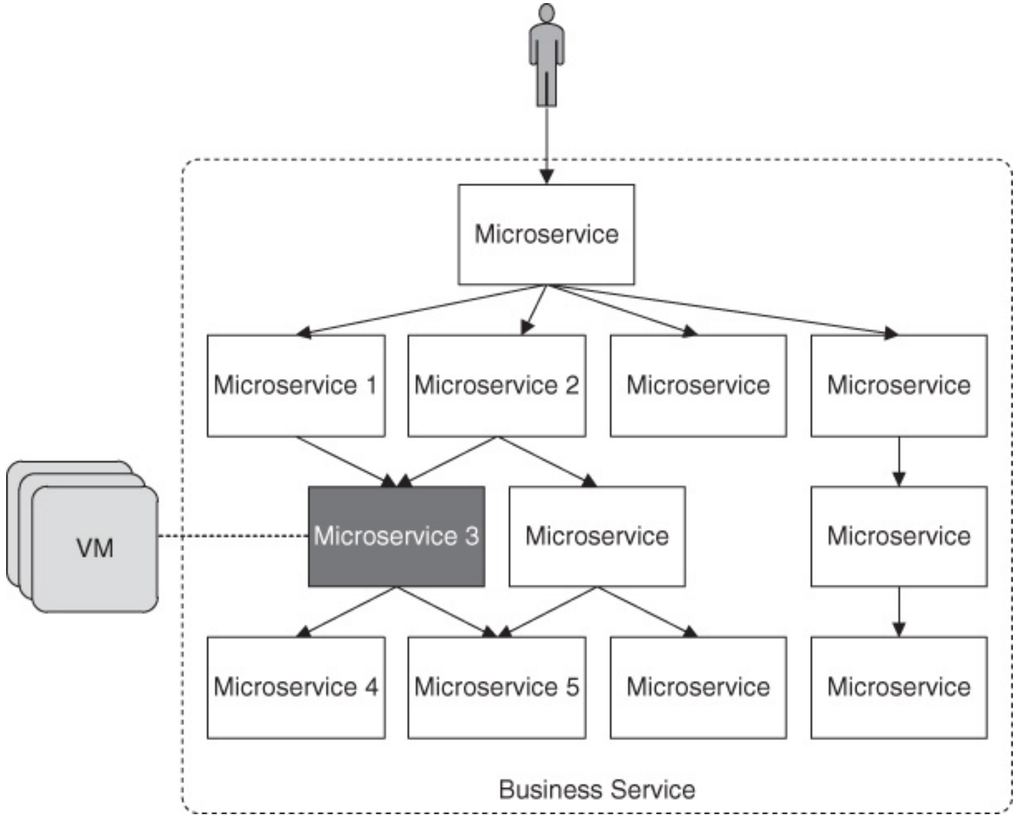
The Deployment Pipeline

Deployment

Error Code 725: It works on my machine.
—RFC for HTTP Status Code 7XX: Developer Errors

Introduction

- The overall goal of a deployment is to place an upgraded version of the service into production with minimal impact to the users of the system, be it through failures or downtime
- There are three reasons for changing a service
  - to fix an error, to improve some quality of the service, or to add a new feature
- The goal of a deployment is to move from the current state that has N VMs of the old version, A, of a service executing, to a new state where there are N VMs of the new version, B, of the same service in execution



Strategies for Managing a Deployment

Strategies for Managing a Deployment

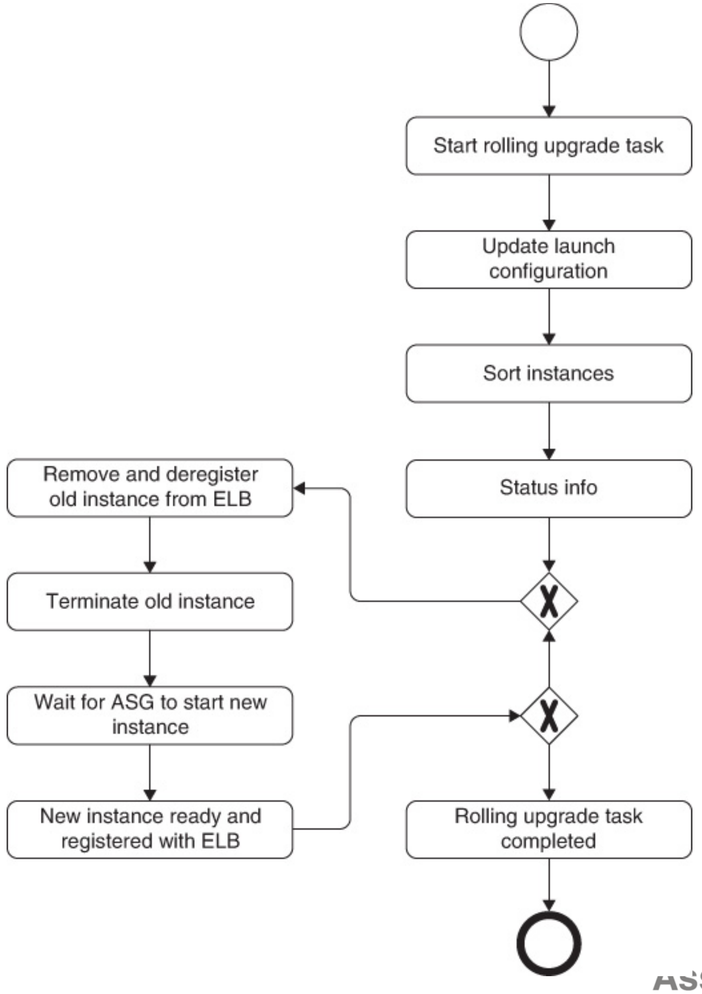
- There are two popular strategies for managing a deployment: blue/green deployment and rolling upgrade
  - They differ in terms of costs and complexity
  - The cost may include both that of the VM and the licensing of the software running inside the VM

Blue/Green Deployment

- (sometimes called big flip or red/black deployment) consists of maintaining the N VMs containing version A in service while provisioning N VMs of virtual machines containing version B
- Once N VMs have been provisioned with version B and are ready to service requests, then client requests can be routed to version B

Rolling Upgrade

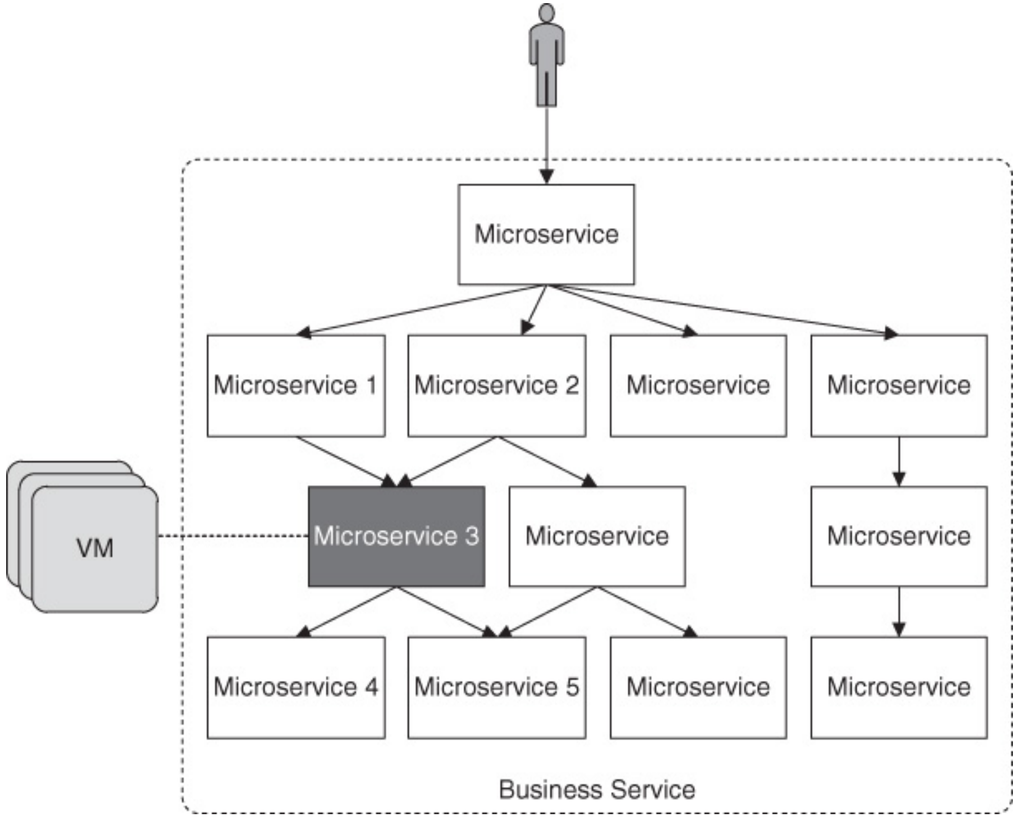
- Consists of deploying a small number of version B VMs at a time directly to the current production environment, while switching off the same number of VMs running version A
- During a rolling upgrade, one subset of the VMs is providing service with version A, and the remainder of the VMs are providing service with version B
  - This creates the possibility of failures as a result of mixed versions



Logical Consistency

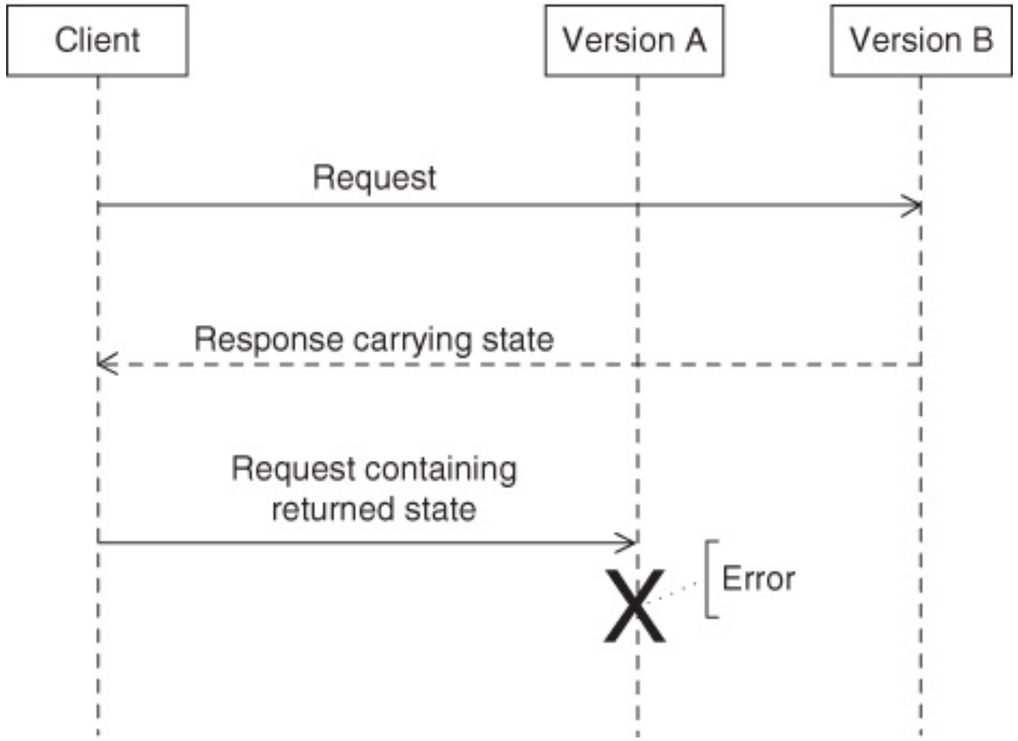
Logical Consistency

- Assuming that the deployment is done using a rolling upgrade introduces one type of logical inconsistency—multiple versions of the same service will be simultaneously active
- This may also happen with those variants of the blue/green deployment that put new versions into service prior to the completion of the deployment



Multiple Versions of the Same Service Simultaneously Active

- Two components are shown—the client and two versions (versions A and B) of a service
- The client sends a message that is routed to version B
- Version B performs its actions and returns some state to the client
- The client then includes that state in its next request to the service
- The second request is routed to version A, and this version does not know what to make of the state, because the state assumes version B
- Therefore, an error occurs
- This problem is called a mixed-version race condition



Multiple Versions of the Same Service Simultaneously Active

- Make the client version aware so that it knows that its initial request was serviced by a version B VM. Then it can require its second request to be serviced by a version B VM
- Toggle the new features contained in version B and the client so that only one version is offering the service at any given time
- Make the services forward and backward compatible, and enable the clients to recognize when a particular request has not been satisfied

Feature Toggling

- Can be used to control whether a feature is activated
  - Is a piece of code within an if statement where the if condition is based on an externally settable feature variable
- Using this technique means that the problems associated with activating a feature are
  a) determining that all services involved in implementing a feature have been sufficiently upgraded and
  b) activating the feature in all of the VMs of these services at the same time

Feature Toggling

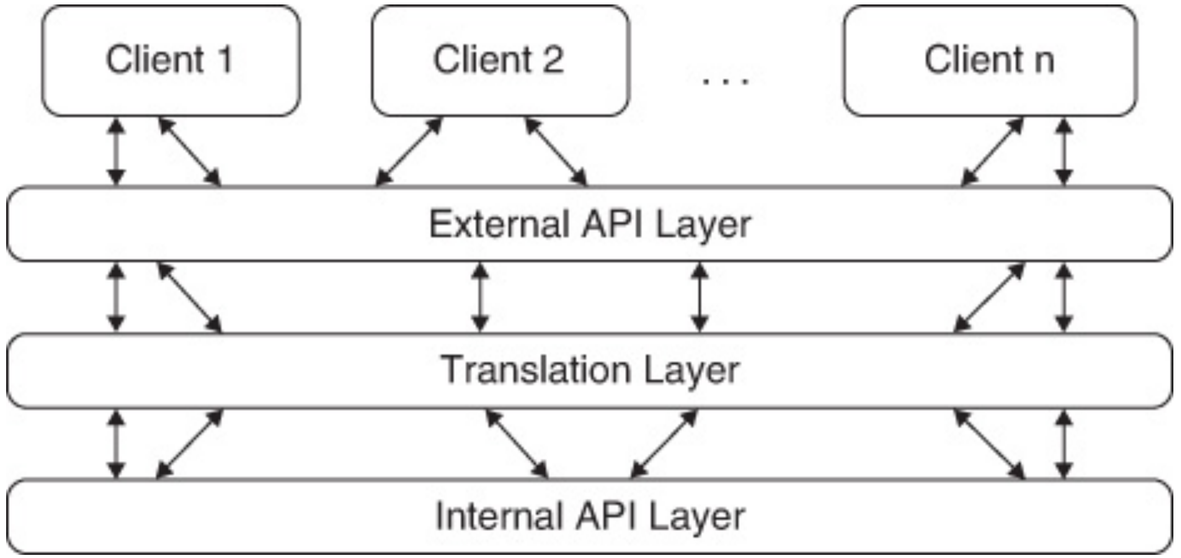
- One complication comes from deciding when the VMs have been “sufficiently upgraded”
  - VMs may fail or become unavailable
  - Waiting for these VMs to be upgraded before activating the feature is not desirable
- The use of a registry/load balancer enables the activation agent to avoid these problems

Backward and Forward Compatibility

- A service is backward compatible if the new version of the service behaves as the old version
  - The external interfaces provided by version B of a service are a superset of the external interfaces provided by version A of that service
- Forward compatibility means that a client deals gracefully with error responses indicating an incorrect method call

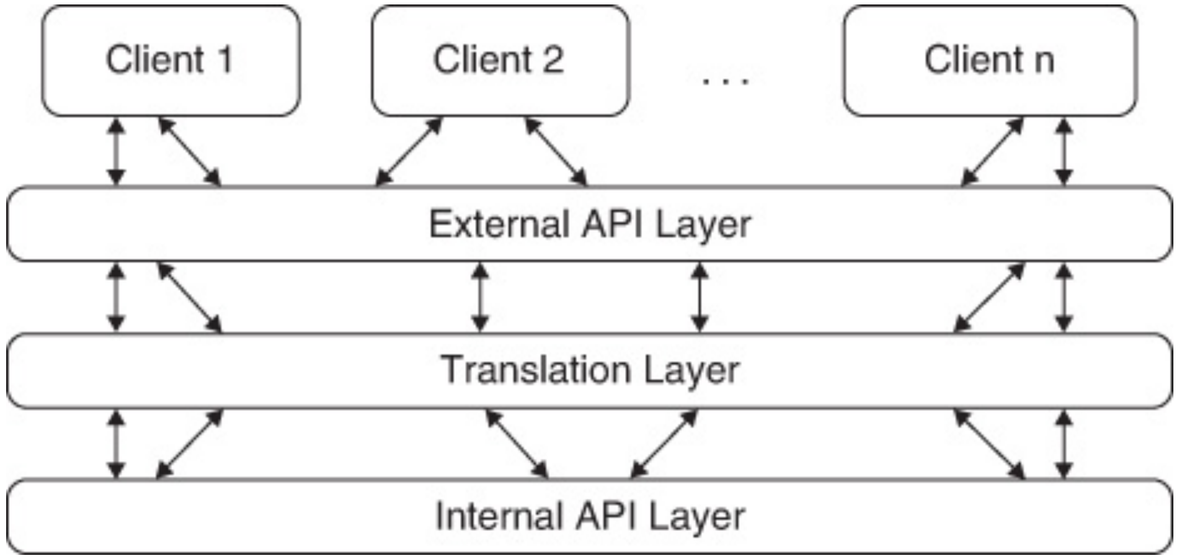
Maintaining backward compatibility for service interfaces

- Requiring backward compatibility might seem at first to preclude many changes to a service
- If you cannot change an interface, how can you add new features or, for example, refactor your service?
- A: The service being upgraded makes a distinction between internal and external interfaces



Maintaining backward compatibility for service interfaces

- External interfaces include all of the existing interfaces from prior versions as well as, possibly, new ones added with this version
- Internal interfaces can be restructured with every version
- In-between the external interfaces and the internal interfaces is a translation layer that maps the old interfaces to the new ones



Compatibility with Data Kept in a Database

- In addition to maintaining compatibility among the various services, some services must also be able to read and write to a database in a consistent fashion
- The most basic solution to such a schema change is not to modify existing fields but only to add new fields or tables [new features in a release]
- If, however, a change to the schema is absolutely required you have two options:
  1. Convert the persistent data from the old schema to the new one
  2. Convert data into the appropriate form during reads and writes. This could be done either by the service or by the database management system

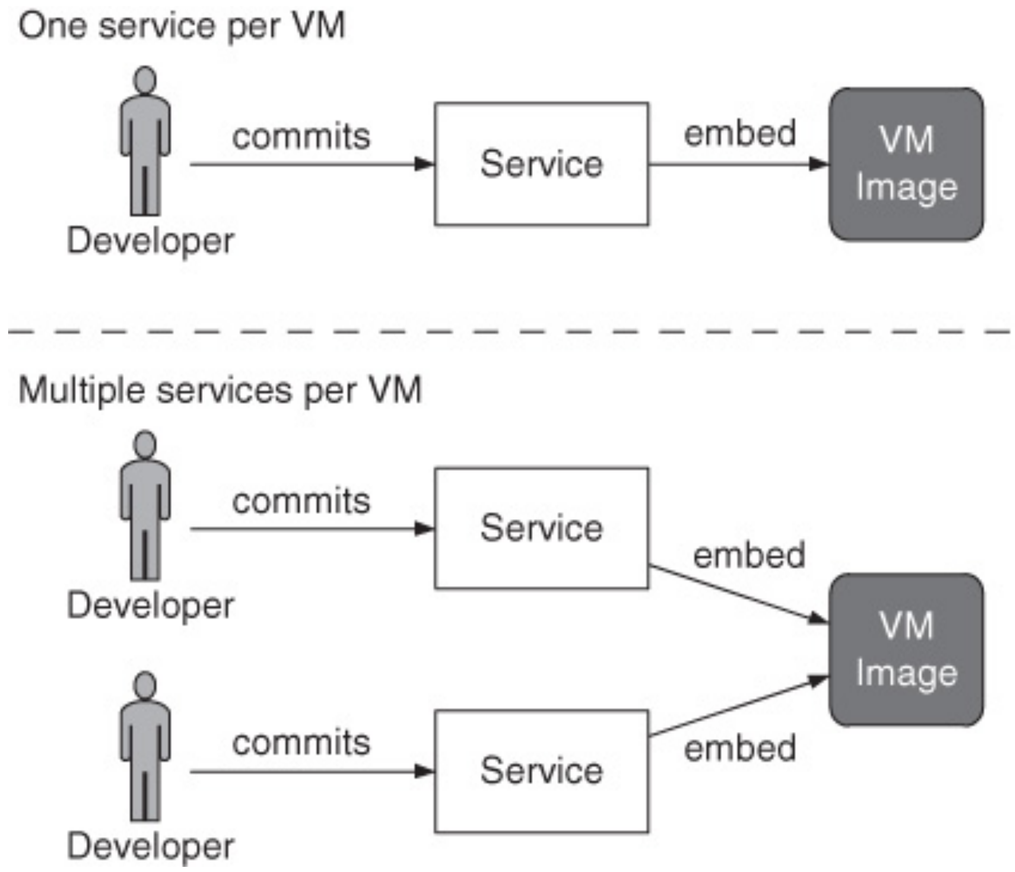
Packaging

Packaging

- We now turn from consistency of services during runtime to consistency of the build process in terms of getting the latest versions into the services
- Deciding that components package services and that each service is packaged as exactly one component does not end your packaging decisions
- You must decide on the binding time among components residing on the same VM and a strategy for placing services into VMs

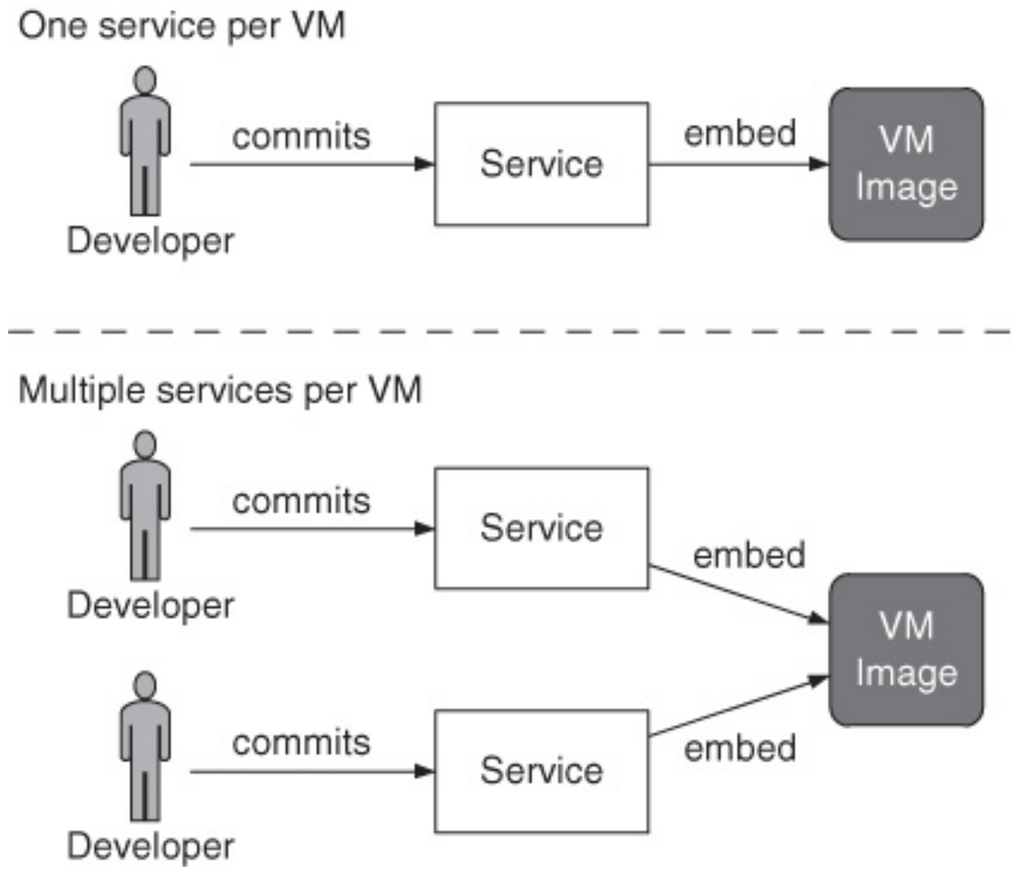
Packaging

- Packaging components onto a VM image is called baking and the options range from lightly baked to heavily baked
- A VM image could include multiple independent processes - each a service
- The question then is: Should multiple services be placed in a single VM image?
- The emergence of lightweight containers often assumes one service per container, but with the possibility to have multiple containers per VM



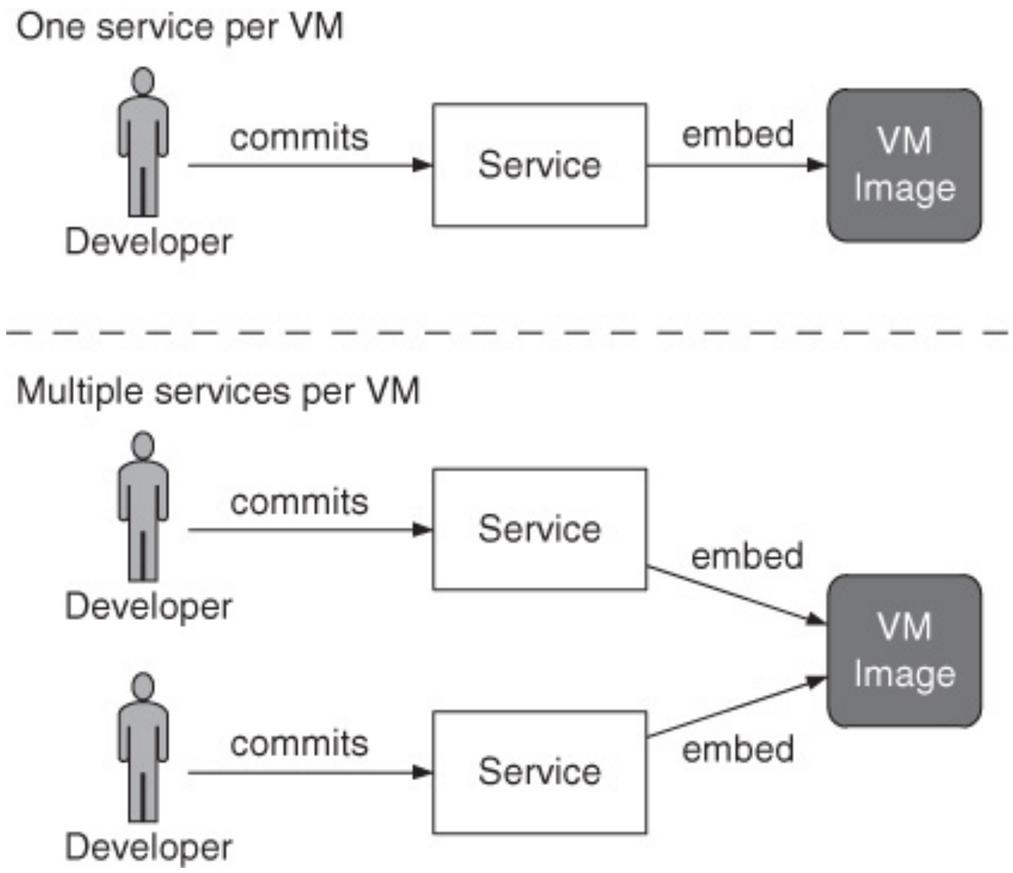
Packaging

- A more important difference occurs when service 1 sends a message to service 2
- If the two are in the same VM, then the message does not need to leave the VM to be delivered
- If they are in different VMs, then more handling and, potentially, network communication are involved
- Hence, the latency for messages will be higher when each service is packaged into a single VM



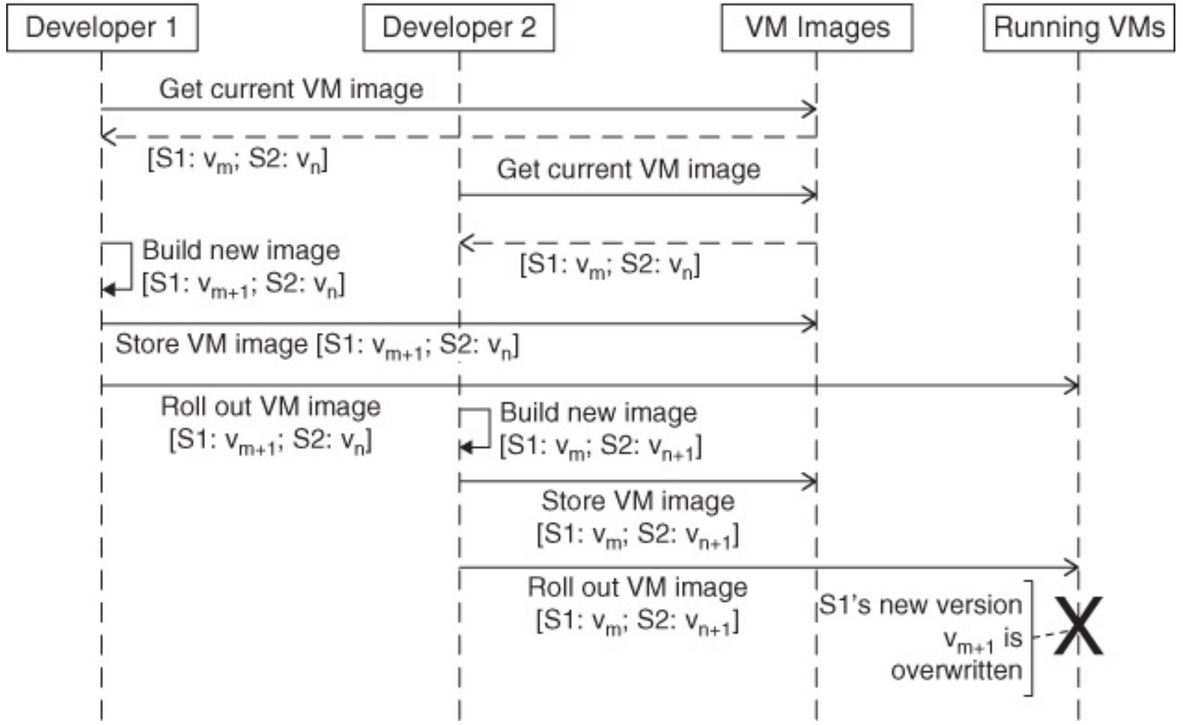
Packaging

- On the other hand, packaging multiple services into the same VM image opens up the possibility of deployment race conditions
- The race conditions arise because different development teams do not coordinate over their deployment schedules
- This means that they may be deploying their upgrades at (roughly) the same time



Packaging

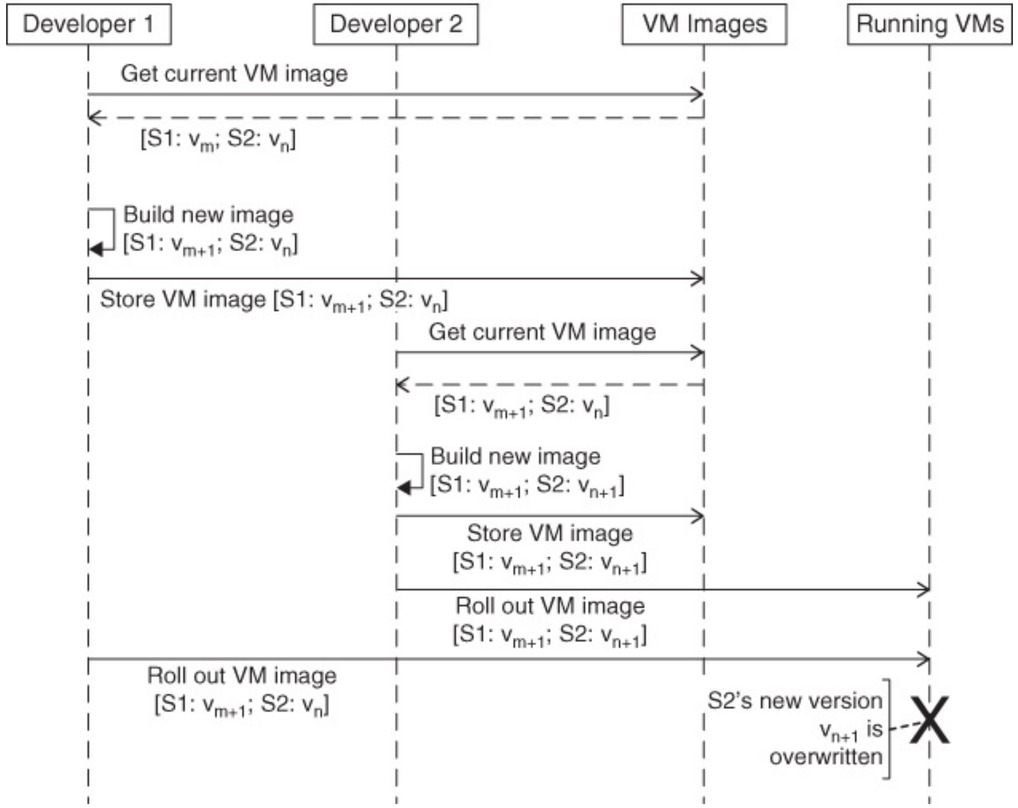
- Development team 1 creates a new image with a new version (vm+1) of service 1 (S1) and an old version of service 2 (S2)
- Development team 2 creates a new image with an old version of service 1 and a new version (vn+1) of service 2
- The provisioning processes of the two teams overlap, which causes a deployment race condition.



AssertLab
Advanced Software and Systems Engineering Research Technologies

Packaging

- Development team 1 builds their image after development team 2 has committed their changes
- The result is similar in that the final version that is deployed does not have the latest version of both service 1 and service 2.



Packaging

- The tradeoff for including multiple services into the same VM is between reduced latency and the possibility of deployment race conditions

Deploying to Multiple Environments

Deploying to Multiple Environments

- As long as services are independent and communicate only through messages, such a deployment is possible basically with the design we have presented
  - The registry/load balancer needs to be able to direct messages to different environments

Deploying to Multiple Environments

- Business Continuity
- Public Cloud
  - Failures do occur
  - Different availability zones or regions
    - State management [making services stateless]
    - Latency
      - Sending messages from one availability zone to another adds a bit of latency; messages sent from one region to another adds more latency to your system
- Private Cloud

Partial Deployment

Canary Testing

- One question is to whom to expose the canary servers
  - organization a user belongs to, for example, the employees of the developing organization, or particular customers
- A new feature cannot be fully tested in production until all of the services involved in delivering the feature have been partially deployed

A/B Testing

- It is the behavior of the user when presented with these two different versions that is being tested
- Implementing A/B testing is similar to implementing canaries
- The registry/load balancer must be made aware of A/B testing and ensure that a single customer is served by VMs with either the A behavior or the B behavior but not both

Rollback

Rollback

- Rolling back means reverting to a prior release
  - rarely triggered automatically
- It is also possible to roll forward - that is, correct the error and generate a new release with the error fixed
- Rolling forward is essentially just an instance of upgrading, so we do not further discuss rolling forward

Tools

Tools

- A large number of tools exist to manage deployment
- One method for categorizing tools is to determine whether they directly affect the internals of the entity being deployed

For Further Reading

- To learn more about the peril of doing an upgrade, you can find an empirical study on the topic at [Dumitras 09]
- To read more about the pros and cons of the heavily baked and the lightly baked approach for VM images, see [InformationWeek 13]
- You can find more about latency between services involving multiple regions/VMs at the links:
  - http://www.smart421.com/cloud-computing/amazon-web-services-inter-az-latency-measurements/
  - http://www.smart421.com/cloud-computing/which-amazon-web-services-region-should-you-use-for-your-service/

Homework #12.1

- Forward and backward compatibility allows for independent upgrade for services under your control. Not all services will be under your control. In particular, third-party services, libraries, or legacy services may not be backward compatible. In this case, there are several techniques you can use, although none of them are foolproof.
  - Discovery
  - Exploration
  - Portability layer
- How do they work? What is the differences among them? And the similarities?

Homework #12.2

- Research and discuss your understanding of the following rolling back strategies
  - Not using feature toggles
  - Using feature toggles

Homework #12.3

- Search for some tools/solutions for the following deployment approaches
  - heavily baked
  - lightly baked