Class 1 – Spring Boot Microservice Enterprise Applications with Java

Contents

Introduction – Instructor and Students

I graduated from Virginia Tech where I got my Master's in Computer Science and with an undergraduate degree from the National University of Singapore. I also completed a study abroad as an exchange student at the University of Nottingham in the United Kingdom. I have certificates as a Scrum Master and Sun Java Programmer and over 12 years of experience as a professional software engineer working in large and small companies. I have also worked as a Teaching Assistant at the National University of

Singapore and a Graduate Research Assistant at Virginia Tech. My published Master's research focused on use of Adaptive Learning in Education (ACM SIGCSE) and Student Evaluations using Mobile Devices (ICDE).

Please connect on LinkedIn: http://www.linkedin.com/in/rahulaga

Class Overview

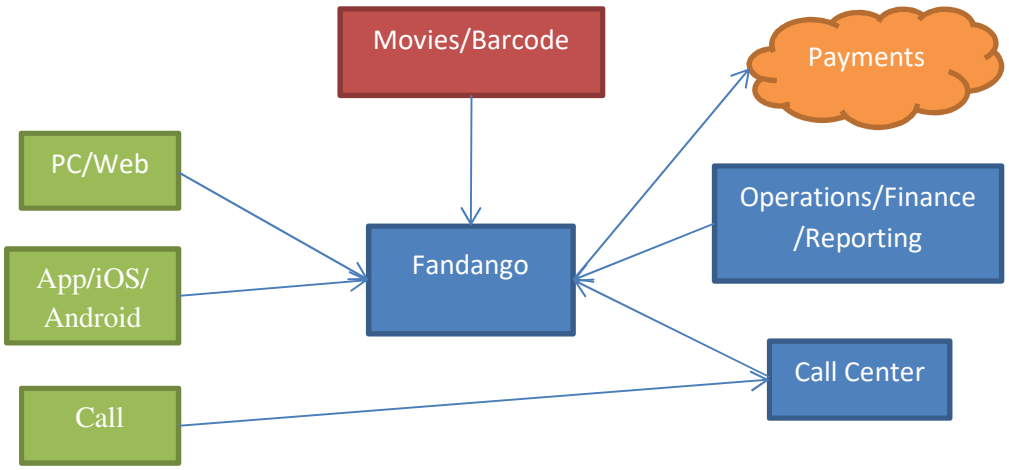
- No Powerpoint!
- Please feel free to ask questions at any time
- After class you will likely need 5-10 hours per week for reading, homework and assignments
- Assumes very good Java knowledge and advanced Object Oriented concepts
- Each class has a handout with links to online resources
- 3 hour class with approximately 3 sections and two 10min breaks after each hour
    - Section 1 - review of previous class homework, assignment and any Q&A and discussions
    - Section 2 and 3 – new topics
- Please let me know any typos/errors in the handouts

Help/Support

- Meaningful contributions to the online discussion forums – this includes asking and answering questions
- Asking questions
    - Research your problem (Google is your friend)
    - Ask specific questions, vague questions will have vague answers. Do you have a stack trace? Can you explain how to reproduce the problem? Can you show the piece of code having the issue?
    - Keep an open mind – the answer may not be what you wanted to hear
- Answering questions
    - Cite your source and explain how/what you learned
    - Be clear and concise – spelling, grammar matters
    - Be polite and have fun, we are all here to learn
    - It is ok to have multiple solutions for the same problem – if you think one is better than the other then explain why

Final Project

We will be working on creating one big application with the last class used for presentations and demo to everyone in class. Consider Fandango – you can find details about movies, theaters and show-times. You can also purchase tickets and take them right to the door. You can also call them for help with your order. We will try to build something along those lines! There will be a final where you will present in class to your fellow students and each assignment will serve as a milestone. Each assignment builds on top of the previous one.



What happens behind the scenes?

Break

Evolution of Web Services

There have always been "processes" in computer sciences as a way to modularize complex applications. For example consider the Unix kernel and the various process and daemons it runs. To facilitate coordination among various processes "inter-process communication" (IPC) was the solution and is an old concept.

Before 1990 – generally the entire application ran within a single physical machine (Java not invented yet).

- "Pipes" were a common way of communication. Eg: $cat file.log | more
- "Shared Memory" was another way. Processes shared some memory space. Pitfalls?
- COM – DLLs etc as a way to share functionality

After 1990 – generally intranet systems, multiple computers on a LAN

- DCOM – distributed COM. Libraries sit on different machines
- RPC – remote procedure calls. Remote Method Invocation (RMI) in Java
- Sockets – TCP/UDP and IP address plus Port number

Different computers could have different architectures (eg: 16bit PC and 32bit mainframe). This resulted in incompatibilities for data transfer so industry created XDR (binary representation) to represent data being passed around.

Late 1990s came the Internet – computers now distributed over a WAN

- With firewalls above methods became obsolete

- Port 80 was always open so created a way in (HTTP technologies)
- XDR changed to XML
- SOAP along with WSDL, XSD and other increasingly complex standards to address needs

Complexity leads to next round – REST in early 2000s

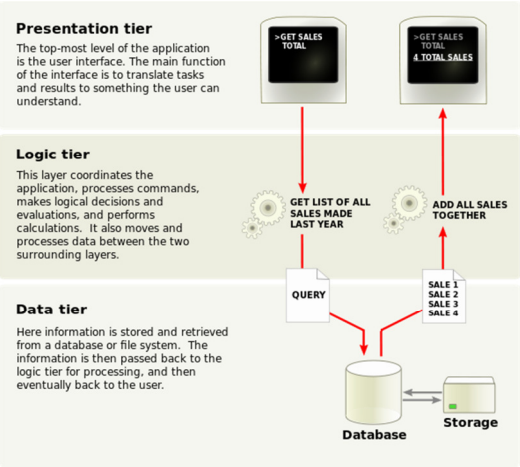
- The web is a network of "**resources**"
- HTTP for CRUD operations – POST, GET, PUT, DELETE

These days' web applications must scale as most rely on a freemium model so require a large customer base with small number of paying customers. Most consumer applications follow this model - think email, iOS/Android apps, games, tax filing services etc. Even large enterprise software companies have a "free" tier like BitBucket, Box.net etc. Another common theme is availability of all kinds of APIs from the smallest startup to giants like Google.

Service Oriented Architecture (SOA)

Wikipedia: http://en.wikipedia.org/wiki/Service-oriented_architecture, http://en.wikipedia.org/wiki/Multitier_architecture

A service is usually implemented in the "n-tier" architecture. At its *simplest* it has 3 layers – the data, logic and presentation tiers. This may be further broken up, or more layers may be added as required.



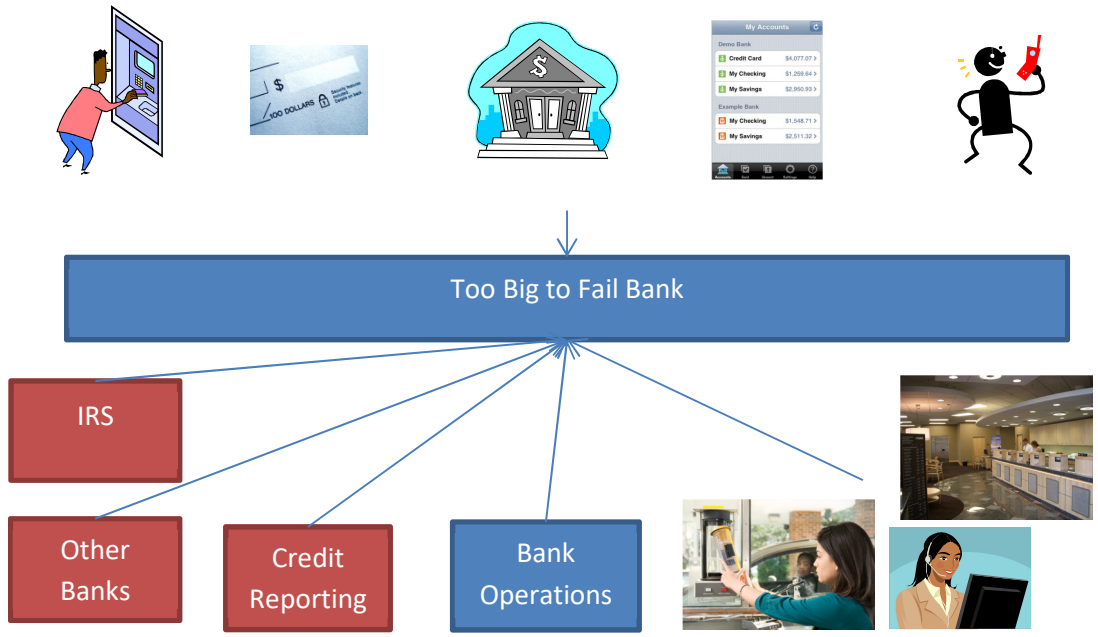
So what is SOA?

- It is an evolution of distributed computing paradigms and NOT a replacement
- The decomposition of a system into autonomous or nearly autonomous units of responsibility or modules
- Loosely coupled services
- Use industry standard protocols to **communicate** among these services (these days REST over JSON is popular)
- Services themselves can be heterogeneous – Java, .NET, Ruby, PHP, Mainframe etc

- Coarse - each service should ideally be a small “standalone” function
- Services publish new functionality or orchestrate other services into new more powerful services
- The service can be thought of as the API for other services

Example – Too Big to Fail Bank

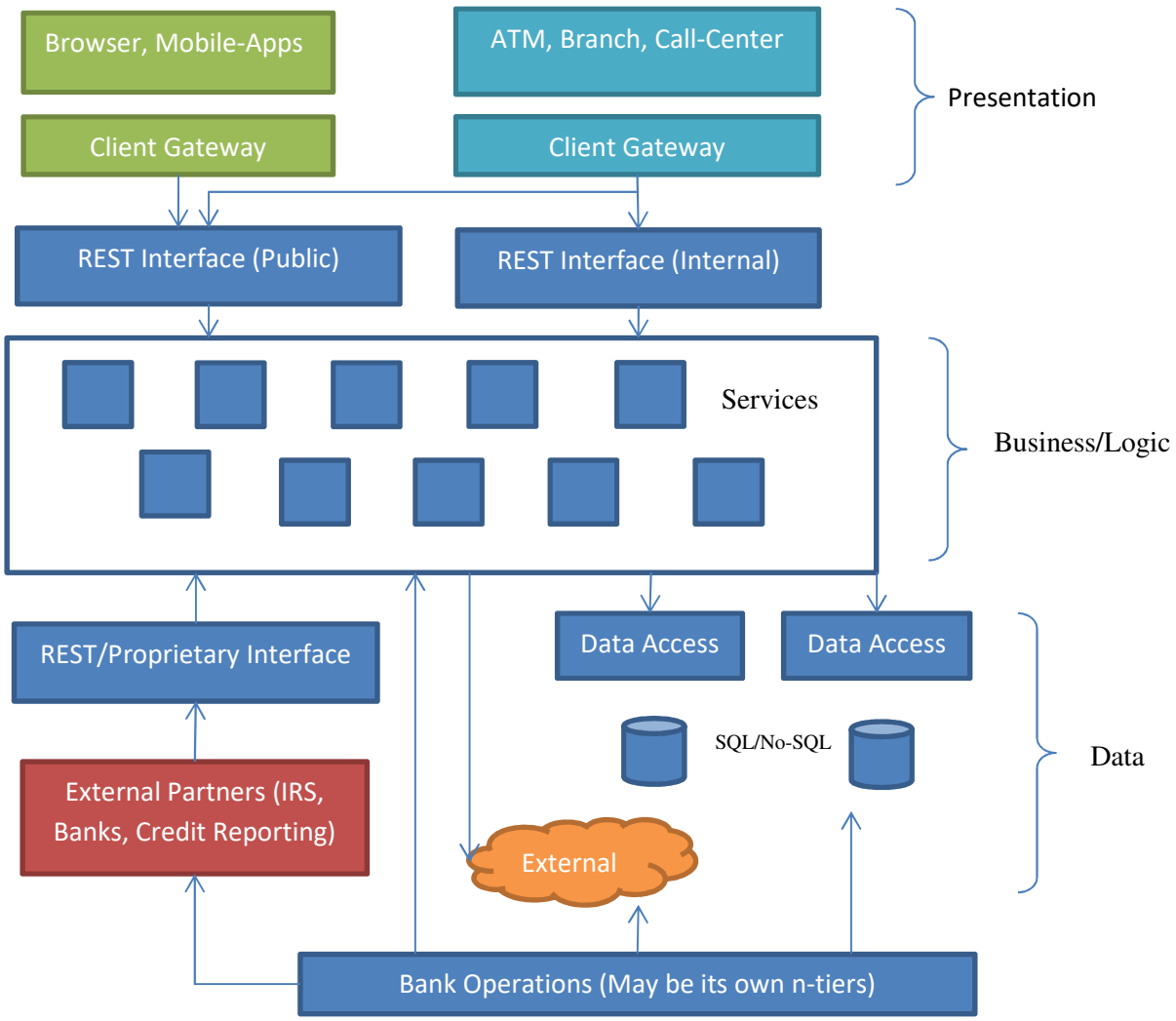
Let us look our Too Big to Fail Bank: as a customer you have a checking account with them. You can access their services via and ATM, on mobile apps, in person at a branch, through a call-center, online or write a check.



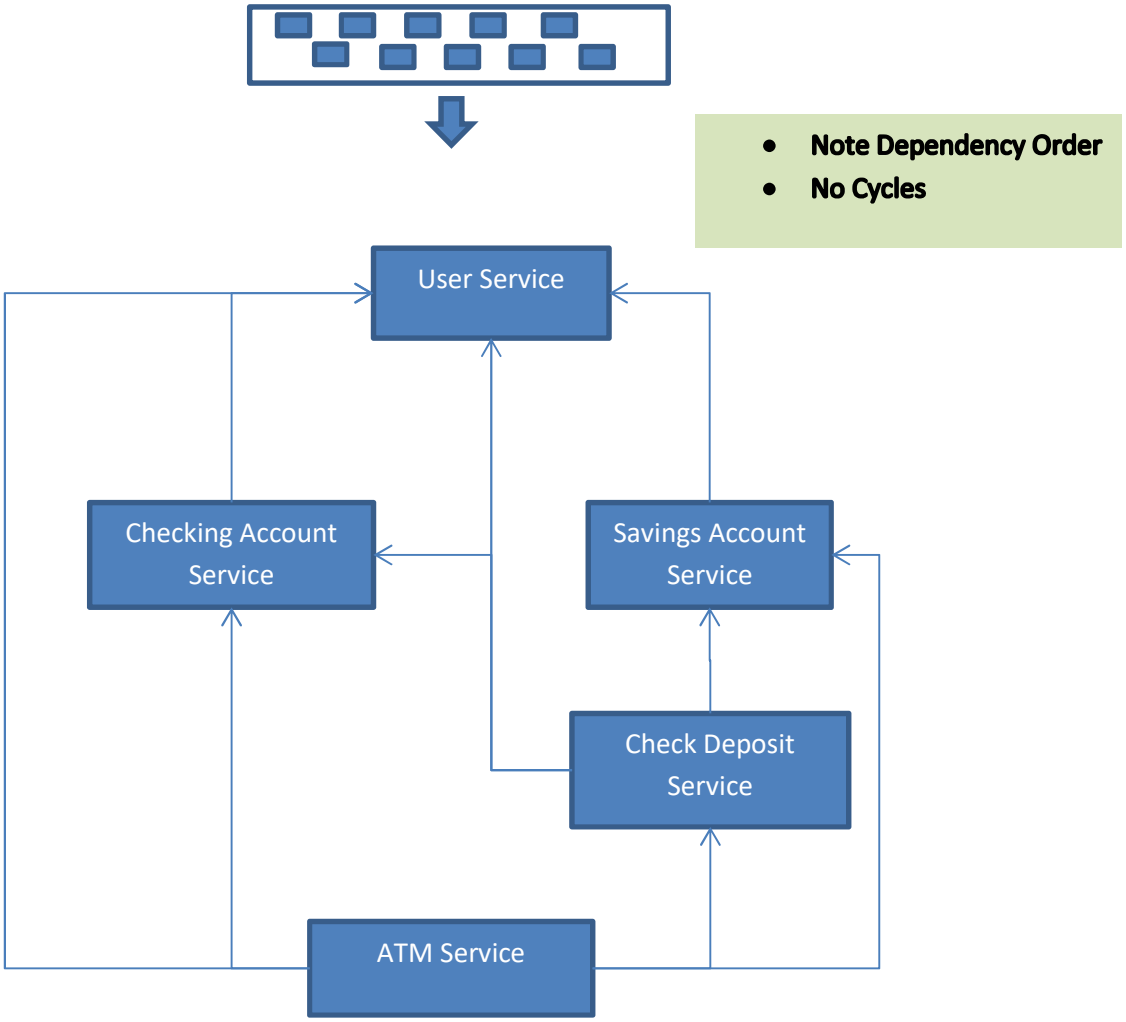
Break

Example Continued

Let us look at how this system maybe decomposed into an n-tier, SOA model. This is but one example and by no means the only way. You may define it differently, so feel free to question it.



Let us look at the details of the Services with a few example services.



Source Control

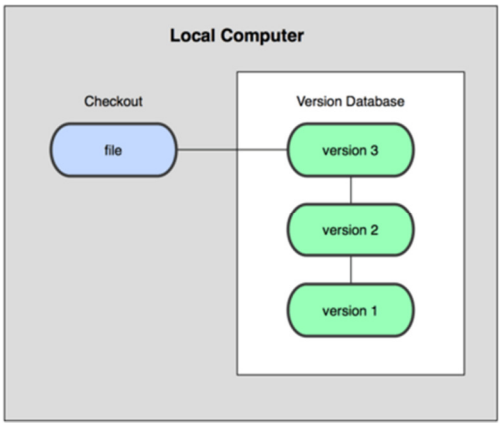
Wikipedia entry: http://en.wikipedia.org/wiki/Revision_control

There are numerous source code control systems also called VCS (version control systems). These have evolved from single file systems to large distributed systems that manage millions of files. Even if you are single programmer working on a small project, source control is still highly recommended! Why?

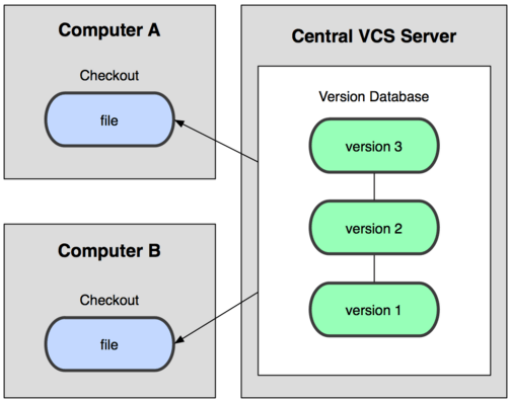
History[1]

1st gen: Single file with multiple versions

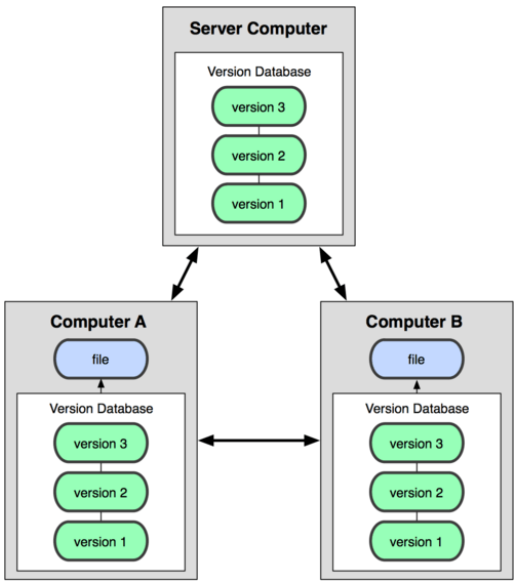
[1] http://www.ericsink.com/vcbe/html/history_of_version_control.html, http://git-scm.com/book/en/Getting-Started-About-Version-Control



2nd gen: Centralized. Eg: CVS, SVN, Perforce



3rd gen: Distributed (DVCS). Eg: Git, Mercurial



Git

There are many Git providers. We will use Bit Bucket which lets you create a free account and private repositories. This will also be where you will present your work for grading.

History

Created by Linus Torvalds (of Linux fame) when the company providing the source control system being used for Linux kernel decided to stop offering free licenses. Some of the goals of the new system were as follows:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

Concept

In Git you have a full copy of the repository locally. All your operations are therefore local (speed, parallel development and distribution goals).

There are 3 states of any file:

- Modified – changed file in your working directory
- Staged – a modified file marked for next commit
- Committed – file submitted to the Git repository (local)

You can work with a central remote repository as well (Bit Bucket in our case). You can "pull" and "push" files.

Homework for Class 1

You **do not need to submit** anything for the homework.

Part 1

1. Create a bit bucket account - https://bitbucket.org
2. Create a private repository and name it anything you want
3. Follow on-screen guide. Here is an **example**.
    a. Install Git – (see Versions Appendix)
    b. Clone repo to your local system. Eg:

```
mkdir /path/to/your/project
cd /path/to/your/project
git init
git remote add origin
https://rahulaga@bitbucket.org/rahulaga/test-repo.git
```

    c. Add a test file
    d. Play around with adding/modifying files and seeing changes locally/remotely

Bit Bucket 101: https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+101

Spend some time taking a look at the Git cheat sheet http://ndpsoftware.com/git-cheatsheet.html

Get an idea of what is available. It is ok if you do not understand everything. We will use Git throughout the class so you will learn more overtime!

Part 2

Class bandwidth maybe insufficient, so before the next class please download and install the following on your laptop:

1. Java **SE** – latest version (see Versions Appendix) for either Windows or Mac depending on your laptop (**Make sure its SE**). Do NOT download with NetBeans options.
   When installing pick a location like C:\jdk8 and create the JAVA_HOME system variable
2. Apache Maven – (see Versions Appendix) for your platform. Again pick a location like C:\apache-maven. Create the M2_HOME system variable
3. **Spring Tool Suite OR Eclipse for Java EE** Developers (see Versions Appendix) for your platform (Make sure you get the Java EE version
4. You may use IntelliJ or other IDE of your choice if you prefer

Part 3

Take a look at Fandango.com and MovieTickets.com. Take note of the various features they have. Create a list of features that stand out to you and if you see anything exclusive offered by one site but not the other.

Appendix – Versions

- Git – Latest version, http://git-scm.com/
- Java SE 8u112 (or later)
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Maven 3.3.9 (earlier 3.x or later versions are ok too), http://maven.apache.org/download.cgi
- Eclipse for Java EE Mars (newer/older version is fine as well), http://www.eclipse.org
- STS - https://spring.io/tools