Taking apart the Monolith: An introduction to Microservices

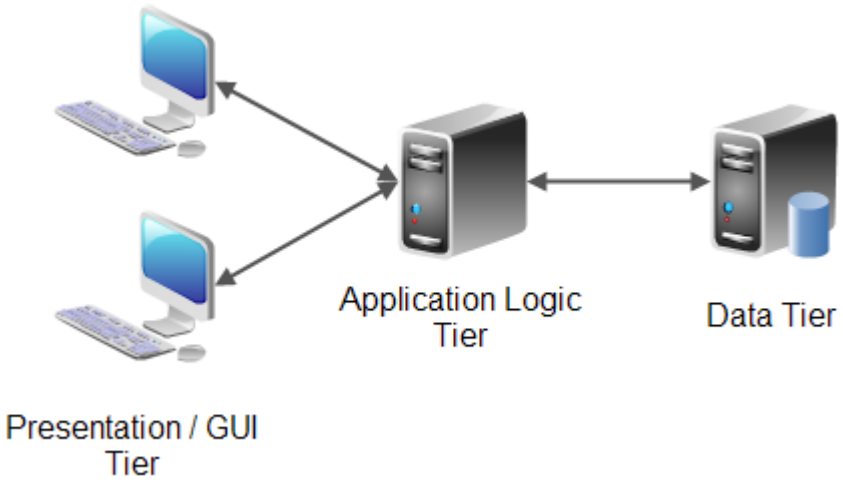
by Shankhadeep Ghoshal
IT2015
Lobackevsky State University of Nizhny Novgorod
ghoshalshankhadeep@hotmail.com

Table of Contents:

Introduction

> The N-tier architecture is more or less monolithic in nature.

The most basic form of an N-tier architecture design is the 3-tier architecture design. It comprises of a storage end popularly known as the *Data Tier*, the middle level tier *Application Logic Tier* and the front end layer which the user interacts with known as the *Presentation Tier*.



> No matter how decentralized we want to make it, there always exist some kind of tight coupling between different architecture levels.

This often causes delays when a change occurs and more often than not, there is a very high chance of entire system breakdown in case of an entire system overhaul.
Also, massive common codebases emerge which delays fault detection and error tracking and thereby makes debugging a big problem which will reduce the quality of performance for the user - not a good trait for business.

And very often we can have problems with the way the *application logic tier* and the *data tier* are couple. For example, there can be multiple application servers connecting directly to the same database server(sometimes it's even the same database) which is a very bad coupling situation. This actually puts the database server at high stress and makes it very likely to break down and as a result there is a denial in services. Additionally, this brings about some core problems like read and write problems(This can be viewed as a single objected being operated on by multiple threads).

Problem Statement

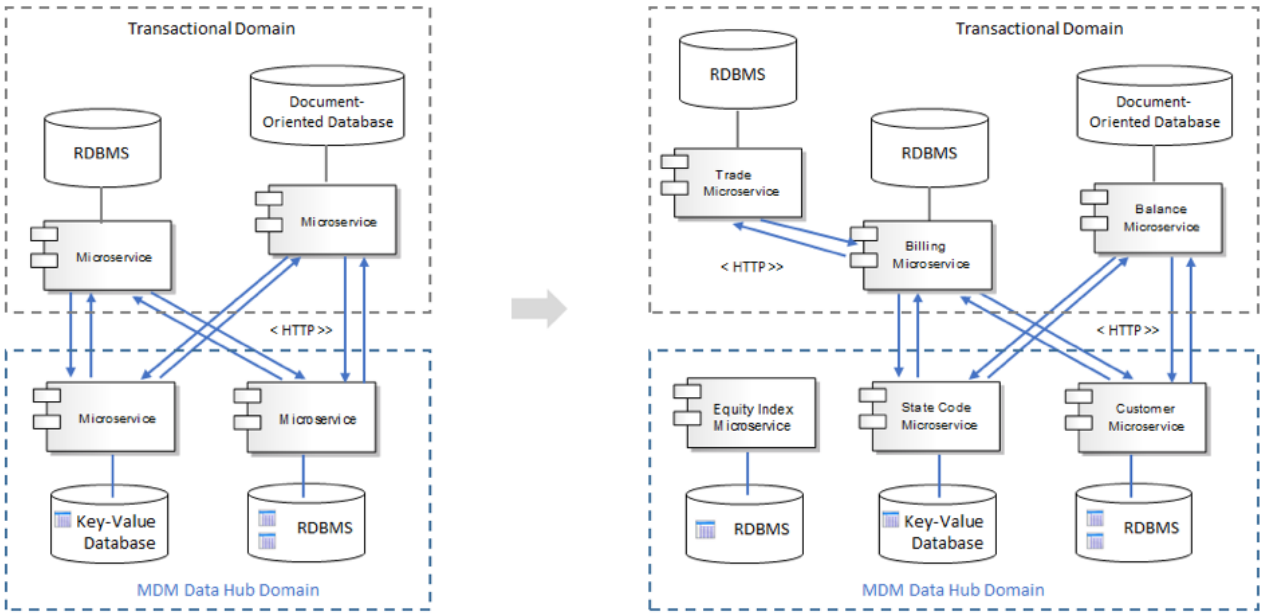
Bottomline: This type of architecture has two major issues:

- Adaptability
- Managing Faults

How to address these above mentioned issues?

Introduction to Microservices

> The problems stated in the problem statement can be solved if we disassemble the various components of the application and execute them as separate **services** with their own execution environments and their own dedicated persistence environments.



This is the basis of **Microservice architecture design**.
To illustrate how all of this happens, we will take an example of a dummy IT services company named **Inglorious Bus Inc.** which will be our test subject.

Overview

Our company **Inglorious Bus Inc** is an IT service providing company which specializes in Big Data and Cloud Computing. They work in a highly challenging environment where adaptability and speedy delivery of results are the biggest keys to success. With the evolution of technology every day, there is a constant need to upgrade and update their information systems and code bases.

Working Methodology

- ### Componentization via Services

  Inglorious Bus Inc. builds components separately called services and then stitches them up to make their IT systems. This means that their main application is made up of numerous sub-applications which functions autonomously.

  For example, they provide a solution called:

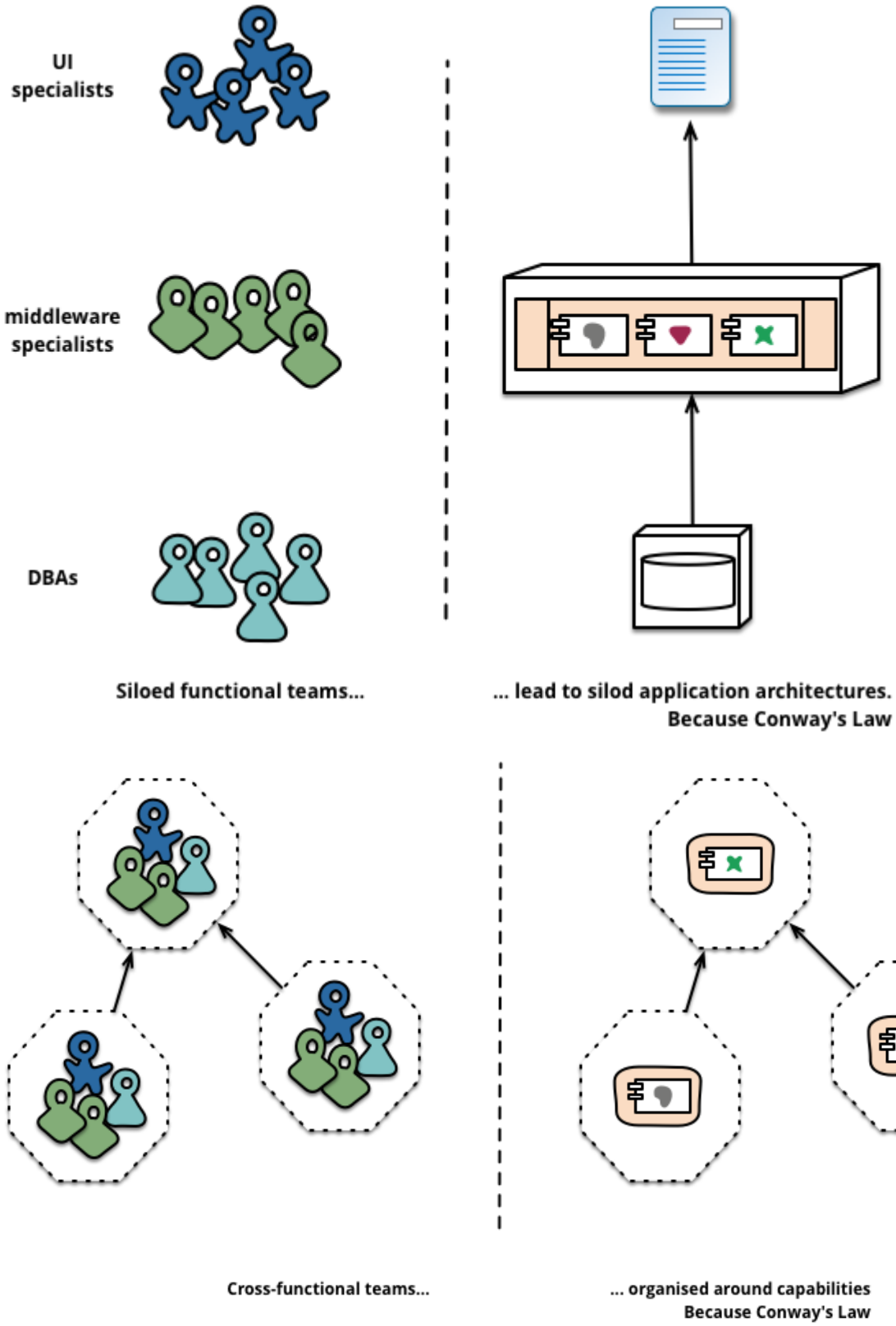
  Financial Assessment Service

  It takes in a ledger from a registered customer and provides a financial assessment of his business which includes various reports. This entire system is run by a `registration service`, a `file manager service`, `front controller service`, `back controller service`, `a classifier service` (part of a bigger big data system), `regression service` (part of a bigger big data system).

  Noteworthy point here is that failure of one component doesn't affect the working of other components i.e. all the components are able to work autonomously.

- ### Organized around Business Capabilities

Inglorious Bus Inc. organizes teams based on individual business services rather than distinct technological domain. This basically means that the team for `registration service` which comprises of various members with UI skills or database management skills or back-end development skills or testers unlike 3-Tier architecture methodology where similarly skilled personnel are grouped together.



- **Products, Not Projects**

> You build, you run it
> -Amazon

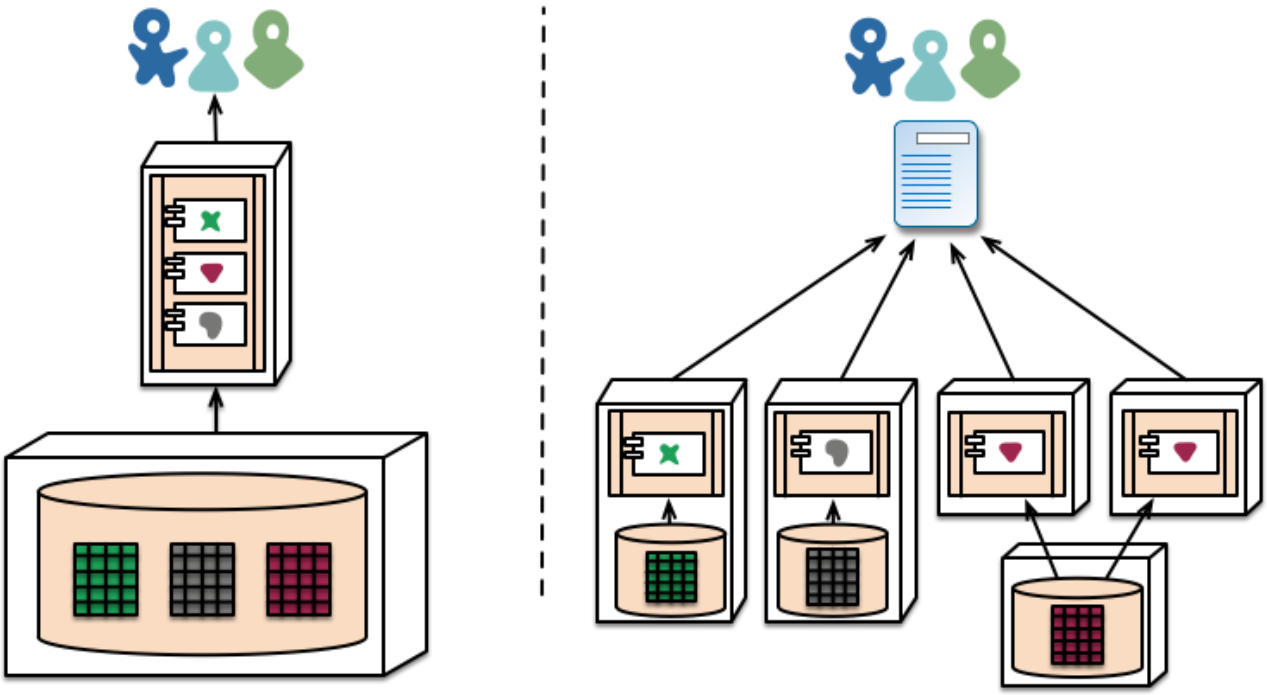
Our company follows this motto of Amazon. This basically mean that the team which is involved with a service, have to do everything from planning to development and testing and maintaining the service unlike 3-Tier architecture where different phases of development are done by different teams.

- ## Smart back-ends and dumb communication protocols

  Our team focuses more on strengthening the back-end more so over using smart communication mediums. The communication mediums are mostly RESTful compliance communication protocols. The two protocols used most commonly are `HTTP request-response with resource API's` and `lightweight messaging`. Thus, it's has more cohesion and is loosely coupled.

- ## Decentralized Governance

  Unlike 3-tier architectural model where standardization of a single technology platform is a tendency, Inglorious Bus Inc. follows a very flexible approach and uses the right tool for the right job and uses **interprocess communication** to combine and come up with a singular solution. This also enable the company to adapt very effectively to change. Like for example the Financial Assessment Service, the `registration service` uses `NodeJS` as the application layer and `MongoDB` as it's persistence layer. The `file manager service` uses `Java` as a scripting language and `HDFS` for persistence. The `classifier` and `regression services` are run by `Scala` with `Apache HBase` as a storage system. A total system overhaul can be done very effectively just by changing each service individually and simultaneously.



- ## Fault Tolerance

  The company is better prepared to deal with customers in case of a breakdown of a single or even multiple entities, as services are distributed(which adds it's own share of complexity). This is achieved as each service runs independent of each other and the entire application can still provide some level of functionality and/or redirect request to backup resources. Also since codebases are separate for

each business, fault diagnostic is a lot quicker compared to monolithic architecture which results in quicker debugging of problems and fast production ready software.

Conclusion

With superior fault tolerance capabilities along with distributed and optimal technology Inglorious Bus Inc.and Continuous Integration and Deployment(CICD) pipeline is extremely loaded(which is good) as a result.

Acknowledgement

- D.Shaposhnikov,Teacher,Software Engineering
- Software Engineering, Ian Sommerville, 9th-Edition
- Martin Fowler
- Martin Fowler KeyNote at ThoughtWorks XCONF 2014