


# MICRO SERVICES ARCHITECTURES AND COMPARISONS

Suzanne Barber

# Terminology

- ❑ Object Oriented Programming (OOP)—a modern programming paradigm
- ❑ Web service / API—a way to expose the functionality of your application to others, without a user interface
- ❑ Service Oriented Architecture (SOA)—a way of structuring many related applications to work together, rather than trying to solve all problems in one application
- ❑ Systems—in the general sense, meaning any collection of parts that can work together for a wider purpose
- ❑ Single Responsibility Principle (SRP)—the idea of code with one focus
- ❑ Interface Segregation Principle (ISP)—the idea of code with defined boundaries.

# Topic Exercise

- 
- What are the quality attributes (goals) driving the developing of microservice architectures? How would you measure each quality?

# MicroServices are the opposite of monolithic architectural styles.

- monolith application is always built as a single, autonomous unit.
  - ▣ In a client-server model, the server-side application is a monolith that handles the HTTP requests, executes logic, and retrieves/updates the data in the underlying database.
  - ▣ monolithic architecture
    - all change cycles usually end up being tied to one another.
    - A modification made to a small section of an application might require building and deploying an entirely new version.
    - To scale specific functions of an application, you may have to scale the entire application instead of just the desired components.

# Micro Services vs. SOA

SOA	MicroServices
more dependent ESBs (Enterprise Service Bus)	faster messaging mechanisms
have an outsized relational database	frequently use NoSQL or micro- SQL databases (which can be connected to conventional databases)

- real difference has to do with the architecture methods used to arrive at an integrated set of services in the first place.

# 1. Software built as microservices can, by definition, be broken down into multiple component services.

## Why?

- Each of these services can be deployed, tweaked, and then redeployed independently without compromising the integrity of an application.
- As a result, you might only need to change one or more distinct services instead of having to redeploy entire applications.
- But this approach does have its downsides, including expensive remote calls (instead of in-process calls), coarser-grained remote APIs, and increased complexity when redistributing responsibilities between components.

## 2. Microservices style is usually organized around business capabilities and priorities.

- microservice architecture utilizes cross-functional teams with specific focus: e.g. UIs, databases, technology layers, or server-side logic.
  - ▣ The responsibilities of each team are to make specific products based on one or more individual services communicating via message bus.
- team owns the product for its lifetime, as in Amazon's oft-quoted maxim "You build it, you run it."

3. Microservices act somewhat like the classical UNIX system: they receive requests, process them, and generate a response accordingly.

- You could say that microservices have smart endpoints that process info and apply logic, and dumb pipes through which the info flows.
- This is opposite to how many other products such as ESBs (Enterprise Service Buses) work, where high-tech systems for message routing, choreography, and applying business rules are utilized.



## 4. Microservices involve a variety of technologies and platforms, old-school methods of centralized governance aren't optimal.

- Decentralized governance is favored to encourage reuse.
  - ▣ A practical example of this is Netflix—the service responsible for about 30% of traffic on the web. The company encourages its developers to save time by always using code libraries established by others, while also giving them the freedom to try with alternative solutions if needed.
- decentralized data management is favored.
  - ▣ each service usually manages its unique database.
  - ▣ Monolithic systems use a single logical database across different applications.

## 5. Microservices are designed to cope with failure.

- Since several unique and diverse services are communicating together, it's quite possible that a service could fail(e.g., supplier isn't available).
  - ▣ client should allow its neighboring services to function while it bows out in as graceful a manner as possible.
- This requirement adds more complexity to microservices as compared to monolithic systems architecture.

## 6. Microservices architecture is an evolutionary design.

- MicroService practitioners see decomposition as a powerful tool that gives them control over application development.
- ▣ A good instance of this scenario could be seen with The Guardian's website (prior to the late 2014 redesign). The core application was initially based on monolithic architecture, but as several unforeseen requirements surfaced, instead of revamping the entire app the developers used microservices that interact over an older monolithic architecture through APIs.

# Summary of Microservice architecture

- Uses services to componentize (services drive component definition)
- Usually organized around business capabilities
- Focuses on products instead of projects
- Has smart end points but not-so-smart info flow mechanisms
- Uses decentralized governance as well as decentralized data management
- Is designed to accommodate service interruptions
- Is an evolutionary model.