Tutorato di Basi di dati e Web

Matthew Rossi
matthew.rossi@unibg.it

Architetture monolitiche

Tutto il codice si trova in un unico sistema

Ha senso quando si crea un sistema di piccole dimensioni, sul quale si lavora autonomamente

Diventa estremamente problematico da gestire quando cresce al di sopra di una certa dimensione e più persone devono collaborare al suo sviluppo

Architetture a microservizi (1)

Naturale evoluzione delle architetture monolitiche

Consiste nel dividere l’architettura monolitica in tanti servizi autonomi di piccole dimensioni che si occupano di realizzare una specifica funzione

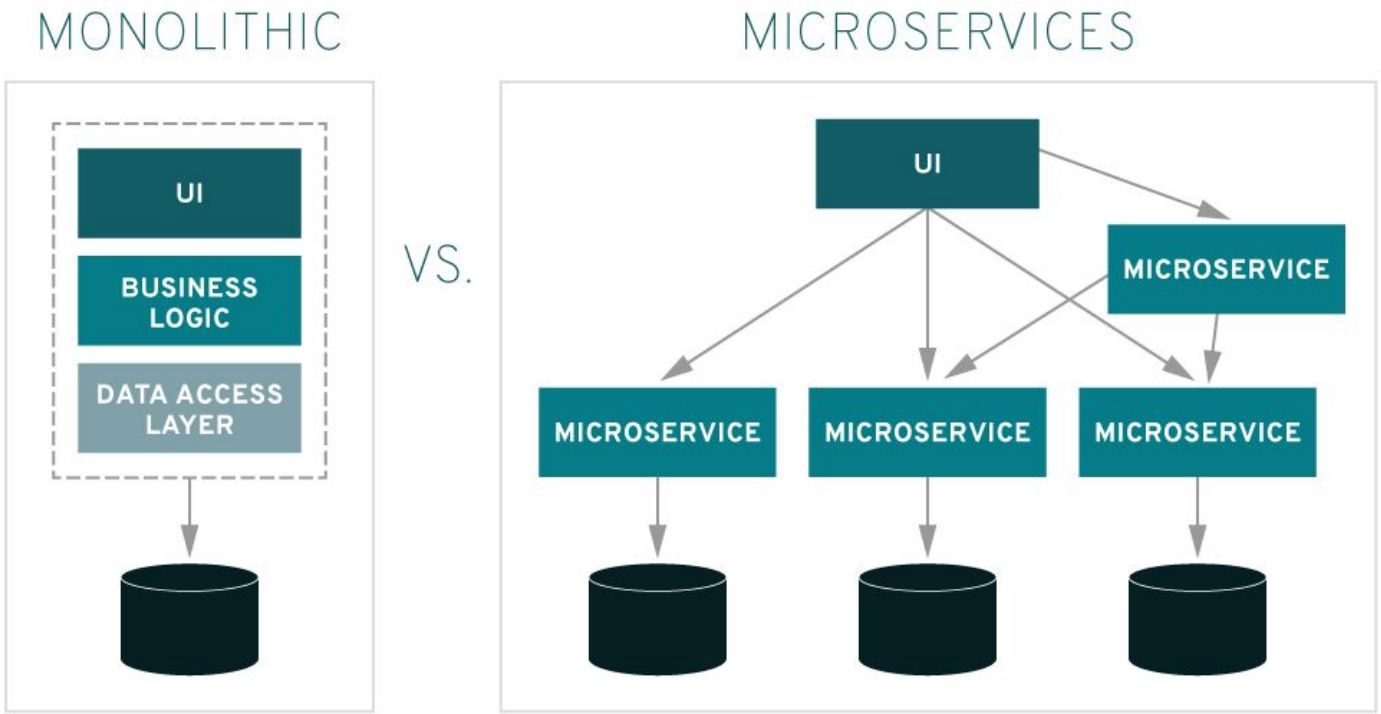
I microservizi di solito interagiscono tra loro al fine di offrire funzioni più ricche all’applicazione

Architetture a microservizi (2)

Rappresentano una buona scelta per realizzare applicazioni complesse di grandi dimensioni con:

- Rapidità
- Affidabilità
- Scalabilità

Architetture a confronto



Vantaggi

- Abilita la Continuous Integration e il Continuous Delivery di applicazioni complesse
  - Manutenibilità
  - Testing
  - Deployment
  - Suddivisione del lavoro tra più team autonomi
- Isolamento dei guasti a singoli microservizi
- Riduce fenomeni di lock-in su uno specifico stack tecnologico

Svantaggi

- Complessità aggiuntiva dovuta alla realizzazione di un sistema distribuito
  - Le richieste possono richiedere il coinvolgimento e il coordinamento di più servizi
  - Gestire guasti parziali dell’architettura
  - Testare le interazione tra i servizi
- Il deployment dell’architettura è più complesso
- Incremento dei consumi di memoria

REST API (1)

REpresentational State Transfer (**REST**) è comunemente usato per l'interazione tra applicazioni e servizi web

Un **API** è un set di definizioni e protocolli per costruire e integrare applicazioni software

- Una architettura client-server gestita mediante **HTTP/HTTPS** (attraverso l'uso dei metodi **GET**, **PUT**, **POST** e **DELETE**)
- Comunicazione client-server priva di stato (**stateless**)
- Le risposte devono dichiarare se possono fare uso di **cache**
- Un'interfaccia comune

REST API (2)

L’interfaccia uniforme consente di semplificare e disaccoppiare l’architettura

- Le risorse richieste sono identificate (**URL**) e la loro rappresentazione interna è indipendente da quella inviata al client
- Manipolazione della risorsa mediante la rappresentazione ottenuta
- I messaggi includo sufficiente informazione per essere processati (**media type**)
- Il server può rispondere con risorse che includo hyperlinks ad altre risorse disponibili