Luca Cabibbo
Architettura dei Sistemi Software

Architettura a microservizi

dispensa asw550
marzo 2019

The future is already here –
it's just not very evenly distributed.
William Gibson

- Fonti

- Newman, S. **Building Microservices**: Designing Fine-grained Systems. O'Reilly, 2015.
- Richardson, C. **Microservices Patterns**: With examples in Java. Manning, 2019.
  - https://microservices.io/
- Wolff, E. **Microservices**: Flexible Software Architectures. Leanpub, 2016.
- Lewis, J. and Fowler, M. Microservices: a definition of this new architectural term. 2014.
  - http://martinfowler.com/articles/microservices.html
- Bass, L., Weber, I., and Zhu, L. **DevOps**: A Software Architect's Perspective. Addison-Wesley, 2015.
  - Chapter 4, Overall Architecture
  - Chapter 6, Deployment
- Evans, E. **Domain-Driven Design**: Tackling Complexity in the Heart of Software. Addison-Wesley, 2004. Referenziato come [DDD]

- Obiettivi e argomenti

- Obiettivi
  - presentare e discutere i microservizi e l'architettura a microservizi
- Argomenti
  - introduzione
  - architettura a microservizi
  - discussione

* Introduzione

- In alcune dispense precedenti abbiamo presentato le capacità fondamentali delle tecnologie a servizi, nonché una nozione piuttosto generale di "architettura a servizi"
  - l'obiettivo fondamentale dell'architettura a servizi è sostenere la costruzione di sistemi informatici in grado di soddisfare gli obiettivi di business, correnti e futuri, delle organizzazioni
  - in effetti, esistono diversi tipi di "architetture a servizi"
  - in una precedente dispensa abbiamo presentato anche l'architettura orientata ai servizi (SOA) – un pattern architetturale molto ambizioso, che però è stato un successo solo parziale
  - questa dispensa presenta l'architettura a microservizi – che è un altro importante caso specifico di "architettura a servizi"

Dai servizi ai microservizi

- La SOA è stata, in pratica, un successo solo parziale – tuttavia, la SOA è basata su idee molto ragionevoli – e, in particolare, sulla nozione di servizio
  - per questo, molti team di sviluppo hanno provato ad applicare alcune idee dell'architettura a servizi, insieme a varie idee e pratiche agili, e sfruttando anche le possibilità offerte dalle tecnologie di virtualizzazione e dal cloud
  - da queste sperimentazioni è emersa, come un trend o un pattern, l'*architettura a microservizi* – o, più semplicemente, i *microservizi*
    - alcuni casi di successo sono Amazon, Ebay, Groupon, Netflix, Spotify, Uber e Zalando
  - è bene osservare che i microservizi sono ancora un argomento in "rapido movimento" – per questo non è ancora possibile dare una definizione consolidata di questo pattern architetturale

* Architettura a microservizi

- Una definizione preliminare
  - l'*architettura a microservizi* è un pattern architetturale per lo sviluppo di una singola applicazione come un insieme di microservizi
  - i *microservizi* sono dei servizi "piccoli" e autonomi, eseguiti come processi distinti, che lavorano insieme comunicando mediante meccanismi leggeri
- Attenzione: talvolta si parla informalmente di "microservizi" anziché di "architettura a microservizi"
  - ad es., "i microservizi sono un pattern architetturale per lo sviluppo di una singola applicazione …"
  - tuttavia, in questo modo è difficile capire se il termine "microservizi" si riferisce al pattern architetturale oppure ad un insieme di microservizi

Architettura a microservizi e SOA

- La definizione preliminare consente di evidenziare alcune similitudini e soprattutto alcune differenze significative tra l'architettura a microservizi e la SOA
  - similitudini
    - sia l'architettura a microservizi che la SOA sono delle architetture a servizi, basate sulla nozione di servizio
  - differenze
    - l'architettura a microservizi è una application architecture (per una singola applicazione) – mentre la SOA è una enterprise architecture (per tutte le applicazioni e i sistemi di un'organizzazione)
    - i microservizi sono "piccoli" – questo termine suggerisce l'utilizzo di metodi e pratiche agili per lo sviluppo del software
    - i microservizi comunicano mediante "meccanismi leggeri" – e non "pesanti" come quelli richiesti dai servizi web SOAP

Influenze sull'architettura a microservizi

- Un'altra considerazione importante per comprendere l'architettura a microservizi è che essa è fortemente influenzata dai metodi per lo sviluppo agile (che si sono sempre più affermati in questi anni), nonché dalla virtualizzazione e dal cloud
  - team piccoli e autonomi, che lavorano in modo iterativo, per sviluppare e rilasciare software con il più alto valore per i clienti, nel più breve tempo possibile
  - DDD (Domain-Driven Design) come guida per la modellazione e la progettazione del software – dunque, sulla base di opportuni modelli di dominio e pattern di progettazione
  - Continuous Delivery, per fare in modo che il software possa essere rilasciato nell'ambiente di produzione in ogni momento
  - la virtualizzazione e l'automazione dell'infrastruttura
  - il cloud, che consente di acquisire risorse computazionali in modo elastico e sulla base di un modello di pagamento a consumo

- Architettura a microservizi

- Il contesto tipico per l'architettura a microservizi
  - il business di un'organizzazione è centrato su una singola applicazione – ad es., Netflix, Spotify, Uber o anche Amazon
- Problemi affrontati dall'architettura a microservizi
  - è richiesta un'elevata scalabilità – l'applicazione ha milioni di utenti in tutto il mondo
  - è richiesta un'alta disponibilità – poiché un'interruzione di servizio dell'applicazione è anche un'interruzione del business
  - è richiesta agilità – l'agilità del business è legata soprattutto alla capacità di offrire (implementare e rilasciare) funzionalità e caratteristiche ("servizi") sempre migliori o innovative nell'ambito di questa applicazione
  - inoltre, l'applicazione potrebbe essere stata inizialmente sviluppata in modo monolitico – e deve essere riingegnerizzata affinché possa soddisfare tutti questi requisiti

Architettura a microservizi

- La soluzione dell'architettura a microservizi
  - l'applicazione viene definita come un insieme di microservizi
  - ogni *microservizio* è un servizio coeso e "piccolo", che rappresenta una specifica capacità di business autocontenuta, in esecuzione in un proprio processo (o VM o contenitore)
  - i microservizi comunicano sulla base di meccanismi leggeri – spesso mediante delle API basate su HTTP
  - i microservizi sono autonomi – ciascun microservizio può essere scritto in un linguaggio di programmazione distinto da quello degli altri microservizi, può essere basato su uno stack software diverso, e può anche utilizzare una tecnologia differente per la gestione dei dati
  - ciascun microservizio può essere rilasciato indipendentemente dagli altri – mediante meccanismi completamente automatizzati
  - c'è un minimo di gestione centralizzata di questi microservizi

I microservizi sono servizi

- Innanzitutto, i microservizi sono servizi – come tali, devono soddisfare i diversi principi per la progettazione dei servizi (anche se talvolta con interpretazioni più rilassate che non nella SOA)
  - contratto e astrazione (incapsulamento) – i microservizi hanno un'interfaccia opportuna, separata dalla loro implementazione, mediante la quale è possibile interagire con essi
  - accoppiamento debole e autonomia – queste caratteristiche si riferiscono sia ai microservizi che ai rispettivi team di sviluppo
  - componibilità e riusabilità – i microservizi possono essere composti in modo agile – anche se non sempre devono poter essere riusati in applicazioni diverse
  - scopribilità – di solito ci sono diversi team di sviluppo separati, ciascuno dei quali deve poter scoprire, comprendere ed invocare correttamente i microservizi sviluppati da altri team
  - i microservizi sono (preferibilmente) stateless

I microservizi sono coesi e "piccoli"

- Ciascun microservizio è coeso e "piccolo" – è focalizzato su una singola capacità di business, che deve svolgere bene
  - i microservizi sono unità di sviluppo fortemente modulari – altamente coesi e debolmente accoppiati tra loro – per favorire la flessibilità e la modificabilità sia dei singoli servizi che dell'intero sistema software
    - la modularità dei microservizi è una caratteristica molto più importante della "dimensione" dei microservizi – anche se, purtroppo, il qualificativo "micro" sembra suggerire che la caratteristica più importante dei microservizi sia la loro "dimensione"

I microservizi sono coesi e “piccoli”

- Ciascun microservizio è coeso e “piccolo” – è focalizzato su una singola capacità di business, che deve svolgere bene
  - non c’è una misura precisa della dimensione “corretta” per i microservizi
    - i microservizi devono essere piccoli abbastanza – ma non più piccoli
    - un microservizio potrebbe corrispondere al software che un team di sviluppo agile (di solito tra 5 e 9 persone, ovvero dei “two-pizza team”) può inizialmente fare (nel senso di poter rilasciare) in una singola iterazione (di 2-4 settimane)
  - poiché ciascun microservizio è “piccolo”, un’intera applicazione comprende in genere una molteplicità di servizi
    - ad es., Spotify è basato su oltre 800 microservizi, sviluppati da circa 700 sviluppatori – ogni team di sviluppo (di al più 8 persone) gestisce una decina di microservizi (*dati: 2016*)

I microservizi sono autonomi

- Ciascun microservizio costituisce un’entità software autonoma e separata dagli altri microservizi
  - i diversi microservizi possono essere sviluppati con tecnologie differenti (linguaggi, middleware e dati) e rilasciati in esecuzione in ambienti distinti
    - ciascun microservizio può essere così realizzato nel modo più opportuno possibile per quel microservizio
    - l’eterogeneità viene risolta mediante l’uso di meccanismi di comunicazione interoperabili

I microservizi sono autonomi

- Ciascun microservizio costituisce un'entità software autonoma e separata dagli altri microservizi
  - l'autonomia dei microservizi consente di
    - sviluppare e verificare ciascun microservizio separatamente dagli altri
    - rilasciare ciascun microservizio separatamente dagli altri
    - replicare, per la scalabilità, ciascun servizio separatamente dagli altri, sulla base delle esigenze effettive di carico
    - sostituire un microservizio con una sua nuova implementazione, quando necessario

I microservizi sono componibili

- Poiché ciascun microservizio è "piccolo", l'intera applicazione viene definita come composizione di una molteplicità di servizi
  - la composizione dei microservizi deve essere flessibile – in modo da poter integrare nuovi microservizi quando vengono sviluppati
  - per questo, viene di solito realizzata come una coreografia di microservizi, in cui la forma preferita di comunicazione tra microservizi è quella basata sulla notifica asincrona di eventi
    - ogni microservizio notifica gli eventi significativi che si sono verificati nel suo contesto, in modo tale che altri microservizi, ricevendo queste notifiche, possano agire di conseguenza
    - in questo modo, è possibile anche aggiungere nuovi microservizi alla coreografia, senza modificare i microservizi già esistenti
  - ove necessario può essere usata anche l'invocazione sincrona di operazioni remote

- Microservizi e agilità

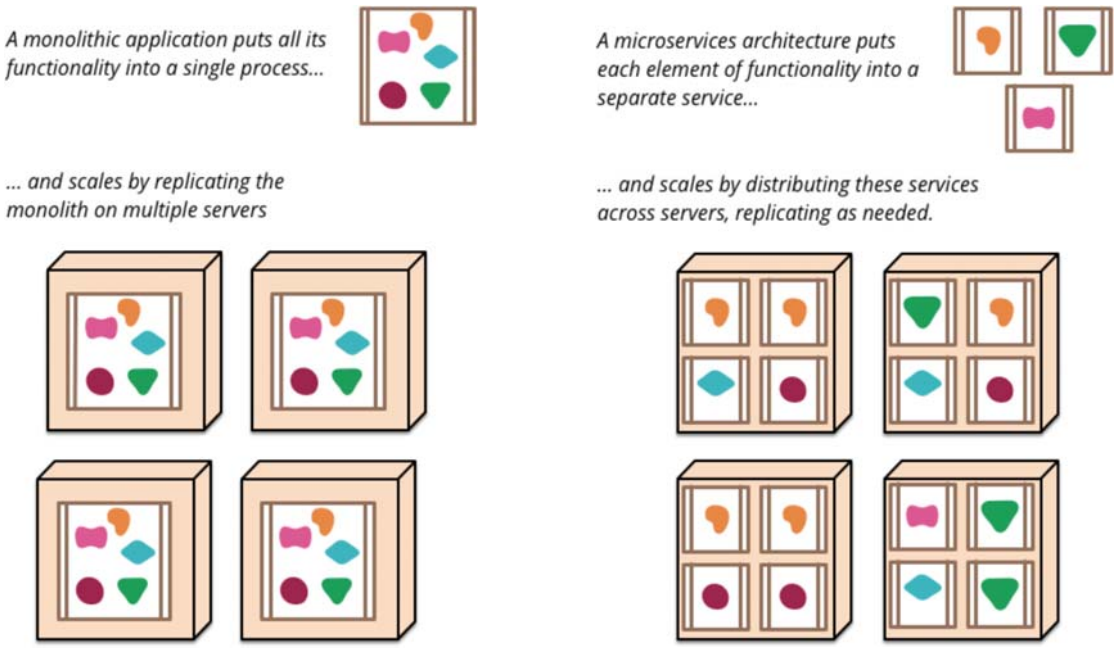
- L'agilità di un'applicazione a microservizi richiede che i diversi team di sviluppo possano lavorare su parti separate dell'applicazione, ciascuno alla propria velocità e con un impatto minimale tra i team
  - i team devono dunque poter operare in modo autonomo, e poter prendere decisioni per implementare e gestire i propri microservizi in modo ottimale
  - i team devono infatti essere liberi di poter effettuare velocemente cambiamenti ai propri microservizi, quando il business lo richiede – e anche di poter rilasciare velocemente questi cambiamenti

- Dal monolito ai microservizi

- Spesso, l'architettura a microservizi viene applicata per ristrutturare un'applicazione che è stata inizialmente realizzata in modo monolitico
  - questa ristrutturazione viene di solito proposta quando si scopre che l'applicazione monolitica è insoddisfacente per uno scarso supporto a una o più qualità importanti – come modificabilità, scalabilità e rilasciabilità – e che una causa principale per lo scarso controllo delle qualità è proprio la monoliticità dell'applicazione
  - in questo caso è necessario effettuare una decomposizione (di solito iterativa) del monolito in microservizi – in modo tale da sostenere le qualità desiderate
  - questo pone il problema di identificare dei criteri opportuni di decomposizione in microservizi di un'applicazione

Dal monolito ai microservizi

- Un motivo comune per l'adozione dell'architettura a microservizi è rendere scalabile un'applicazione monolitica – che per questo non scala facilmente

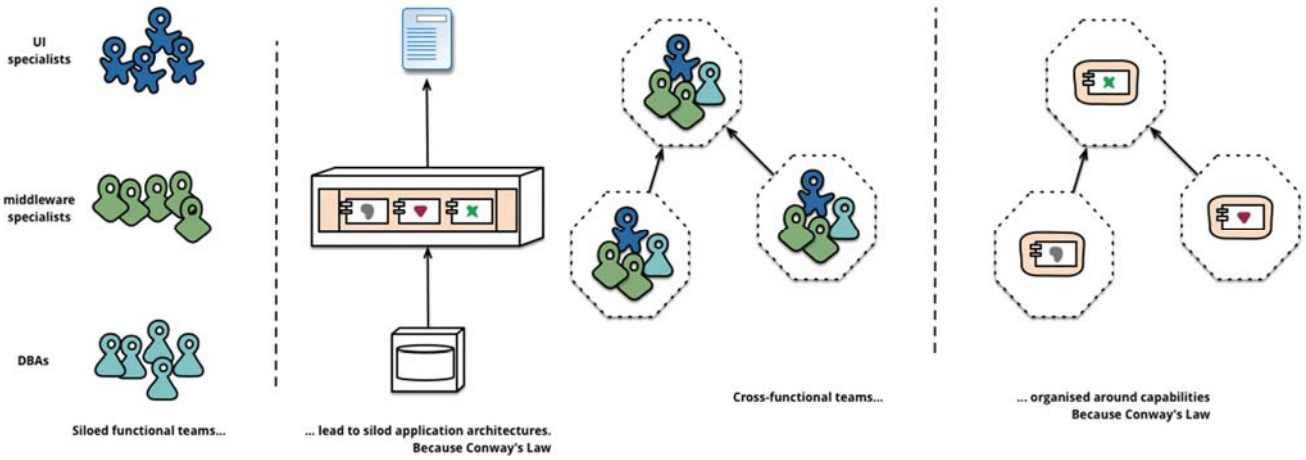


Microservizi e capacità di business

- Un criterio comune di decomposizione dei microservizi è che ciascun microservizio rappresenti una singola capacità di business in modo autocontenuto
  - alcuni team di sviluppo sono specializzati su un singolo aspetto del software – come la UI, la base di dati o la logica lato server
    - questo di solito porta a un'architettura in cui gli elementi principali rappresentano le diverse specializzazioni dei team
  - i team di sviluppo ideali per i microservizi sono invece team cross-funzionali (o full stack) – che comprendono capacità relative ai diversi aspetti del software
    - ciascun microservizio si occupa di tutti gli aspetti del software – UI, logica applicativa e dati – relativi a una specifica capacità di business (anziché tecnica)
  - una decomposizione basata su capacità di business favorisce meglio la scalabilità che non una decomposizione basata su capacità tecniche

Microservizi e capacità di business

- Le precedenti osservazioni fanno riferimento alla legge di Conway
  - ogni organizzazione che progetta sistemi produrrà inevitabilmente dei progetti che sono una copia della struttura di comunicazione dell'organizzazione



- Microservizi e DDD

- La decomposizione di un'applicazione in microservizi può essere guidata dall'applicazione di *Domain-Driven Design* (DDD)
  - DDD è un approccio allo sviluppo di sistemi software complessi
  - DDD definisce due categorie principali di pattern – pattern tattici e pattern strategici – per affrontare problemi di modellazione e progettazione tattici e strategici
  - intuitivamente, la progettazione strategica ha a che fare con l'architettura del software (ovvero, riguarda gli elementi architetturali e le loro interazioni), e la progettazione tattica con la progettazione interna dei singoli elementi architetturali
  - nel contesto della decomposizione in microservizi sono rilevanti soprattutto i pattern strategici
    - un possibile criterio di decomposizione è che ciascun microservizio rappresenti un piccolo dominio autonomo e limitato – ovvero, un *Bounded Context* [DDD] – che può essere poi utilmente implementato come un "esagono"

Bounded Context [DDD]

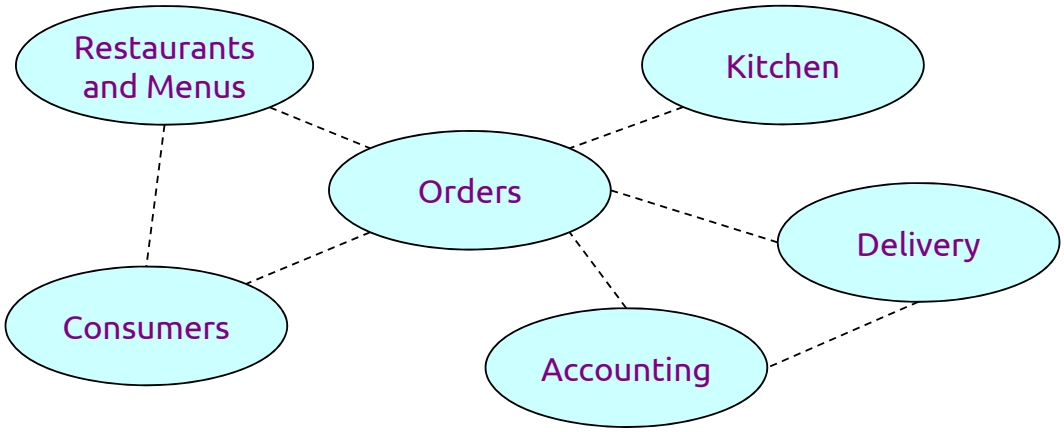
- Il pattern *Bounded Context* rappresenta il principio di progettazione strategica fondamentale di DDD
  - in ogni grande progetto esistono diversi modelli di dominio distinti – ad es., relativi a comunità di utenti o compiti diversi
  - pertanto è opportuno ricorrere a modelli multipli – poiché un singolo modello potrebbe essere errato, inaffidabile e difficile da comprendere – e pertanto anche da gestire e da far evolvere
    - piuttosto, ogni concetto o termine di un modello ha un significato preciso solo in un certo contesto specifico (limitato)
  - pertanto, definisci in modo esplicito il contesto entro cui si applica ciascun modello, nonché i confini tra modelli
    - i concetti e i termini di ciascun modello/contesto vanno mantenuti coerenti entro i confini del modello – e bisogna occuparsi separatamente delle relazioni esterne tra contesti

Context Map [DDD]

- Un altro pattern strategico fondamentale è *Context Map* [DDD]
  - ragionare in termini di singoli modelli/contesti limitati (Bounded Context) è insufficiente – è infatti necessaria anche una visione globale del sistema, come base per una strategia di progettazione complessiva
  - pertanto, descrivi anche le relazioni e i punti di contatto tra i modelli/contesti limitati – che rappresentano le necessità di comunicazione e di integrazione richieste tra i diversi modelli, anche sulla base di alcune tipologie di relazioni comuni tra contesti limitati (rappresentate da ulteriori pattern)
  - questa mappa non rappresenta solo le dipendenze (ovvero, le necessità di comunicazione) tra i diversi contesti limitati – ma anche (e soprattutto) le relazioni tra i relativi team di sviluppo

Bounded Context e Context Map

- Ad esempio, nell'ambito di un'applicazione per la gestione di un servizio di ordinazione e spedizione a domicilio di pasti da ristoranti, su scala nazionale, si potrebbero identificare Bounded Context come Consumers, Orders e Delivery – con la seguente Context Map (che ne mostra anche le dipendenze)



  - questa mappa dei contesti potrebbe guidare la decomposizione in microservizi dell'applicazione

- Decomposizione iterativa del monolito

- La decomposizione di un'applicazione monolitica in microservizi va effettuata in modo iterativo, e non mediante un approccio "big bang" – si può utilizzare il pattern *Strangler* ("strangolatore", dal nome di una pianta tropicale), che suggerisce di
  - aggiungere un API gateway (una facade) di fronte all'applicazione (inizialmente monolitica)
  - sostituire (in modo iterativo e incrementale) pezzi specifici di funzionalità dell'applicazione con nuovi microservizi
    - l'API gateway viene utilizzato per ridirigere le richieste degli utenti finali dell'applicazione verso i nuovi microservizi oppure verso la vecchia applicazione
    - contestualmente, queste funzionalità possono essere "spente" dall'applicazione originale
  - la migrazione delle funzionalità avviene gradualmente – quando è completa, la vecchia applicazione può essere spenta

- Integrazione di microservizi

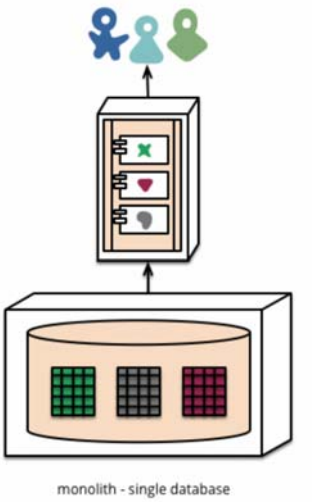
- L'applicazione complessiva è basata sulla composizione dei diversi microservizi – l'integrazione tra microservizi può avvenire a livelli diversi (UI, applicativo e della base di dati)
  - a livello di interfaccia utente (UI)
    - ciascun microservizio può avere una propria UI, che può contenere link alle UI di altri microservizi
    - ad es., l'applicazione potrebbe essere integrata tramite la sua pagina principale, con sezioni differenti per i diversi microservizi
  - a livello della base di dati
    - più microservizi accedono a una base di dati condivisa
    - questa modalità di integrazione è tuttavia sconsigliata, per l'accoppiamento alto tra microservizi – in questo caso, infatti, i microservizi sono accoppiati alla struttura di dati interna degli altri microservizi

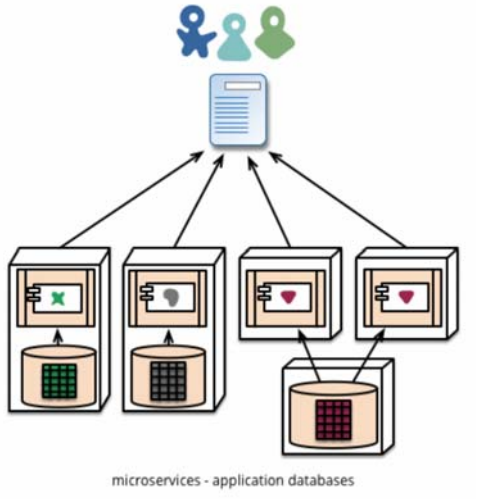
Integrazione di microservizi

- La modalità più comune di integrazione tra microservizi è a livello applicativo – ci sono due approcci principali
  - invocazione di microservizi
    - chiamate remote – basate, ad es., su REST su HTTP – oppure anche su RPC/RMI o SOAP
  - comunicazione asincrona e messaging
    - la comunicazione asincrona offre numerosi vantaggi rispetto all'invocazione remota
      - sostiene un accoppiamento debole
      - sostiene meglio la scalabilità, la disponibilità e l'affidabilità
    - la comunicazione asincrona offre però un supporto solo limitato alla consistenza dei dati (anche se spesso sufficiente in molte applicazioni)

- Microservizi e gestione dei dati

- La decomposizione in microservizi viene operata di solito sia sulle funzionalità che sui dati su cui essi operano – poiché una base di dati monolitica non scala facilmente
  - per questo, in genere ciascun microservizio possiede e gestisce una propria base di dati – realizzata con la tecnologia più opportuna per quel microservizio





Microservizi e gestione dei dati

- Tuttavia, la decomposizione dei dati – in cui ciascun microservizio possiede una propria base di dati – solleva delle sfide
  - transazioni distribuite (per la consistenza dei dati)
    - le transazioni distribuite "tradizionali" sincrone (come il 2PC) sostengono una consistenza forte dei dati
      - tuttavia, possono limitare disponibilità e scalabilità, e per questo vanno usate solo quando è strettamente necessario
    - una soluzione comune è invece usare le saga – una *saga* è una sequenza di transazioni atomiche locali, coordinate mediante lo scambio di messaggi asincroni
      - le saga sostengono solo una consistenza "debole" dei dati – che però è spesso sufficiente per molte applicazioni

Microservizi e gestione dei dati

- Tuttavia, la decomposizione dei dati – in cui ciascun microservizio possiede una propria base di dati – solleva delle sfide
  - interrogazioni distribuite
    - non sono in genere possibili join “globali” tra le basi di dati dei diversi microservizi
    - l’uso di join “applicativi” distribuiti ha diversi svantaggi – e, in ogni caso, non garantisce la consistenza dei risultati
    - una soluzione comune è usare viste materializzate per i pattern di accesso ai dati più importanti (con dati parzialmente replicati in più servizi, replicati in modo asincrono), usando il pattern CQRS

- Microservizi, scalabilità e disponibilità

- Per sostenere scalabilità e disponibilità, è comune avere istanze multiple di ciascun microservizio
  - ciascun microservizio viene replicato separatamente dagli altri
  - in genere è possibile creare e distruggere istanze di microservizi dinamicamente (anche sulla base del carico di lavoro)
- Ciascuna istanza di un microservizio può fallire (per un guasto)
  - ma questo non deve causare un fallimento del microservizio (grazie alle istanze multiple del microservizio)
  - inoltre, il guasto di un microservizio non deve provocare guasti a cascata in altri microservizi – ad es., se un microservizio fa una chiamata ad un’istanza che nel frattempo è fallita
  - pertanto, i microservizi devono essere progettati per sostenere un opportuno isolamento dei guasti

- Infrastruttura per i microservizi

- L'architettura a microservizi richiede un supporto infrastrutturale in diverse aree – tra cui
  - infrastructure automation
    - per il provisioning di VM e contenitori
  - Continuous Delivery
    - per il rilascio dei microservizi
  - message broker
    - per la comunicazione asincrona tra microservizi
  - cache
    - object cache (ad es., per la gestione dello stato delle sessioni) e application cache

Infrastruttura per i microservizi

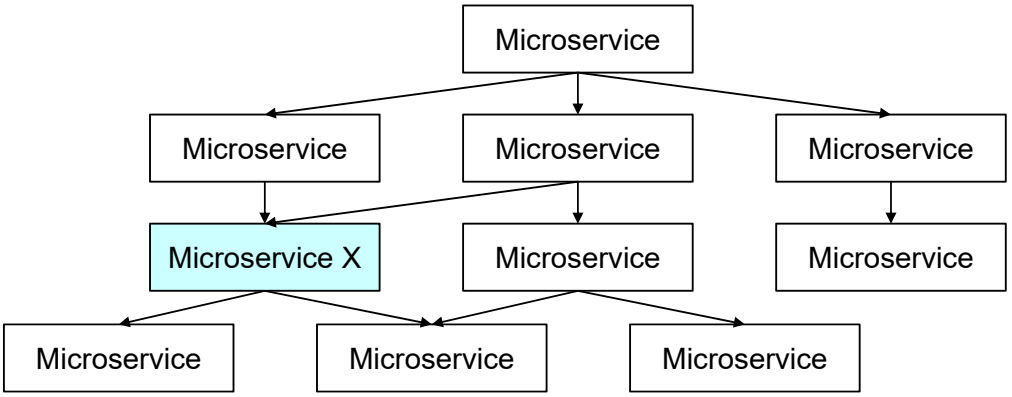
- L'architettura a microservizi richiede un supporto infrastrutturale in diverse aree – tra cui
  - load balancing
    - ci possono essere diverse soluzioni per il bilanciamento del carico – ad es., messaging, load balancer per HTTP, reverse proxy
  - monitoraggio dei microservizi
    - per verificare lo stato di salute dei microservizi ed identificare eventuali problemi

Infrastruttura per i microservizi

- L'architettura a microservizi richiede un supporto infrastrutturale in diverse aree – tra cui
  - service discovery
    - per registrare i microservizi in esecuzione, ciascuno con le sue istanze
    - per consentire ai microservizi (e alle loro istanze) di scoprirsi l'un l'altro
      - questo è importante per consentire le chiamate remote tra microservizi – ma è meno importante quando i microservizi comunicano mediante scambio di messaggi
    - un servizio di service discovery può occuparsi anche di
      - effettuare il monitoraggio dei microservizi
      - supportare il load balancing
      - combinare queste due funzionalità, per evitare ai microservizi di fare chiamate ad un'istanza che è fallita

- Rilascio di microservizi

- Un altro aspetto importante è il rilascio (*deployment* o *delivery* o *release*) di nuove versioni dei microservizi nell'ambiente di produzione
  - i microservizi vengono aggiornati e rilasciati individualmente – dunque i rilasci non sono "monolitici", ma piuttosto riguardano microservizi individuali
  - ad es., un rilascio potrebbe riguardare l'aggiornamento di un microservizio X da una versione A ad una nuova versione B



Rilascio di microservizi

- È importante soprattutto il rilascio di nuove versioni di singoli microservizi nell'ambiente di produzione
  - il rilascio iniziale di un'applicazione a microservizi avviene una sola volta, ed è in genere meno problematico
  - invece il rilascio di nuove versioni dei microservizi può avvenire frequentemente, ed è comunque un'attività più critica
  - in genere, l'obiettivo complessivo è di effettuare un rilascio di una nuova versione in produzione, con un impatto nullo o comunque minimo per gli utenti dell'applicazione, sia in termini di interruzione di servizio che di fallimenti – per minimizzare i rischi del rilascio, che sono non solo tecnici ma anche di "business"
  - il rilascio dei microservizi è basato sulle pratiche della *Continuous Delivery* (*CD*) – in particolare, sulle pratiche della deployment pipeline e sul rilascio senza interruzioni di servizio

- Versionamento dei microservizi

- Ci sono molti motivi per aggiornare un microservizio – dalla correzione di errori, al miglioramento delle funzionalità (ferma restando la sua interfaccia/API), all'introduzione di nuove funzionalità (anche con un'evoluzione della sua interfaccia/API)
  - in alcuni casi è necessaria la coesistenza di più versioni di uno stesso microservizio – ad es., se l'aggiornamento prevede un cambiamento dell'interfaccia del microservizio (che è il caso più problematico), fino a quando non sono stati aggiornati tutti i suoi client
  - ci sono diverse tecniche per gestire questa situazione, ne discutiamo brevemente solo alcune

Versionamento dei microservizi

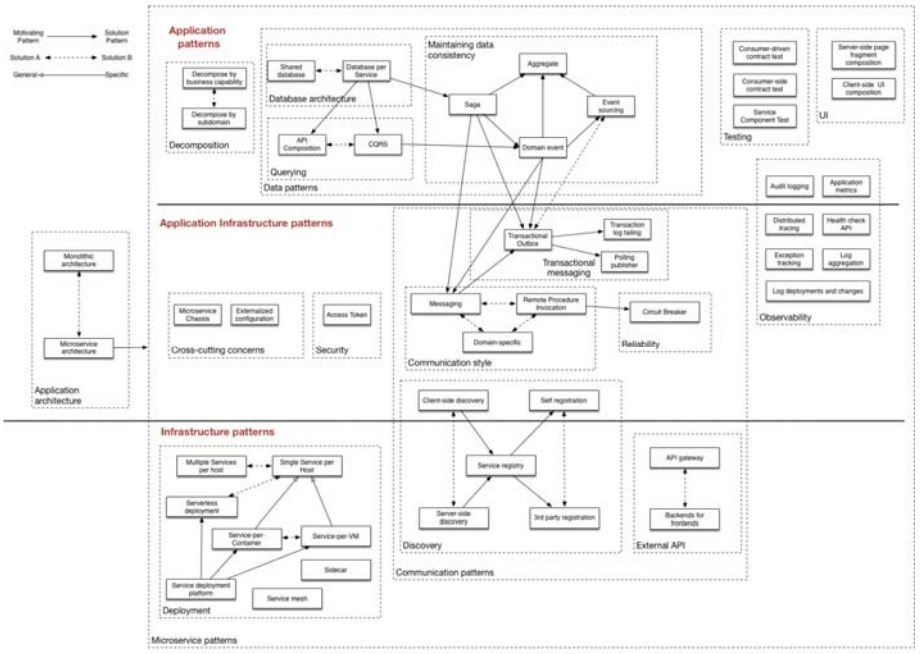
- Alcune tecniche per gestire la coesistenza di più versioni di uno stesso microservizio (per un periodo di tempo più o meno lungo)
  - i client possono fare richieste nei confronti di versioni specifiche del microservizio
  - un microservizio implementa una singola versione (e una singola interfaccia) – nel sistema vengono rilasciate e usate istanze del microservizio relative a versioni differenti
  - in alternativa, un microservizio implementa più versioni (più interfacce, esposte mediante endpoint distinti) – vengono usate istanze identiche dello stesso microservizio
  - i client vengono reindirizzati verso la nuova versione del microservizio – viene effettuato un monitoraggio delle richieste, e la vecchia versione del microservizio può essere spenta quando non riceve più richieste – in ogni caso, anche le versioni dei client sono soggette a scadenze

- Rischi e sfide

- L'architettura a microservizi pone diverse sfide da affrontare
  - un'applicazione a microservizi è un sistema largamente distribuito – e, come tale, può essere molto complesso
  - l'identificazione dei microservizi e dei loro confini è critica
    - il refactoring è difficile – può essere difficile cambiare la scelta dei microservizi e muovere responsabilità tra di essi
    - in genere è suggerito un approccio "monolith first"
  - può essere difficile definire e comprendere le relazioni tra microservizi – poiché possono essere nascoste, ad es., se basate sulla notifica di eventi
  - l'architettura complessiva è importante
    - anche se in teoria molte scelte possono essere effettuate localmente ed autonomamente ai microservizi, in pratica solo una buona architettura di dominio complessiva può sostenere lo sviluppo indipendente dei microservizi

- Pattern per microservizi

- Microservices Patterns [Richardson] presenta un (inizio di) pattern language per microservizi – con oltre 40 pattern



Pattern per microservizi

- Microservices Patterns [Richardson] presenta un (inizio di) pattern language per microservizi – con oltre 40 pattern
  - questo linguaggio di pattern è utile per conoscere le soluzioni che possono essere applicate nell'architettura a microservizi – ma anche per capire quali sono i (tanti) problemi da affrontare
  - ecco i principali ambiti a cui si riferiscono questi pattern
    - architettura applicativa (microservizi vs monolito)
    - applicazione – decomposizione in servizi, gestione dei dati (basi di dati, interrogazioni, consistenza dei dati), interfaccia utente, verificabilità
    - infrastruttura applicativa – stile di comunicazione, affidabilità, scambio transazionale di messaggi, sicurezza, osservabilità (monitoraggio), gestione di interessi trasversali
    - infrastruttura – deployment (deployment dei servizi e piattaforma di deployment), service discovery, API esterna

* Discussione

- Per riassumere
  - i microservizi – conosciuti anche come architettura a microservizi – sono uno stile architetturale che struttura un'applicazione come una collezione di servizi che sono
    - altamente mantenibili e verificabili – da parte di piccoli team
    - debolmente accoppiati
    - rilasciabili indipendentemente – in modo automatizzato
    - organizzati attorno a capacità di business
  - questi servizi comunicano tra loro mediante protocolli leggeri – sincroni o asincroni
  - l'architettura a microservizi
    - sostiene agilità, scalabilità e disponibilità dell'applicazione
    - abilita il rilascio continuo dell'applicazione
    - sostiene la sperimentazione e l'adozione di nuove tecnologie

Discussione

- Alcune organizzazioni stanno usando già da qualche anno l'architettura a microservizi con successo – soprattutto organizzazioni con un modello di business basato su Internet
  - i vantaggi principali dei microservizi sono il supporto per l'evoluzione agile e la scalabilità di una singola applicazione
  - tuttavia, questo pattern architetturale probabilmente non è adatto a tutte le applicazioni e nemmeno a tutti i team di sviluppo – poiché richiede molteplici competenze, in ambito tecnologico, infrastrutturale e metodologico – ed inoltre perché questi ambiti sono in continua evoluzione
  - inoltre, non è passato abbastanza tempo per poter esprimere un giudizio oggettivo su questo stile architetturale – spesso le vere conseguenze di alcune decisioni architetturali si manifestano solo alcuni anni dopo la loro applicazione
  - in ogni caso, il mio suggerimento, oggi, è di cercare di acquisire queste competenze