



Luca Cabibbo
Architettura
dei Sistemi
Software

Architettura a microservizi

dispensa asw550
ottobre 2023

*I am a strong believer that
great software comes from great people.*
Sam Newman



- Riferimenti

- ❑ Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
 - Capitolo 33, **Architettura a microservizi**
- ❑ Lewis, J. and Fowler, M. **Microservices: a definition of this new architectural term**. 2014.
- ❑ Newman, S. **Building Microservices: Designing Fine-grained Systems**. O'Reilly, second edition, 2021.
- ❑ Richardson, C. **Microservices Patterns: With examples in Java**. Manning, 2019.
- ❑ Ford, N. and Richards, M. **Fundamentals of Software Architecture: An Engineering Approach**. O'Reilly, 2020.
- ❑ Ford, N., Richards, M., Saladage, P. and Dehghani, Z. **Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures**. O'Reilly, 2021.



- Obiettivi e argomenti

□ Obiettivi

- presentare e discutere i microservizi e l'architettura a microservizi

□ Argomenti

- introduzione
- architettura a microservizi
- caratteristiche dell'architettura a microservizi
- conseguenze dell'architettura a microservizi
- ulteriori considerazioni
- discussione



* Introduzione

- L'architettura a servizi ha, in generale, l'obiettivo di sostenere la costruzione di sistemi software in grado di soddisfare gli obiettivi di business, correnti e futuri, delle organizzazioni
 - esistono diversi tipi specifici di "architetture a servizi"
 - l'*architettura a microservizi (microservices architecture)* è un altro importante caso specifico di "architettura a servizi"



Dai servizi ai microservizi

- L'architettura a microservizi è emersa (negli anni '10) dalle sperimentazioni relative a variazioni dell'architettura a servizi
 - molti team hanno provato ad applicare in modi nuovi le idee dell'architettura a servizi, combinandole con le pratiche agili e con le possibilità offerte dalla virtualizzazione e dal cloud
 - l'architettura a microservizi nasce proprio da queste sperimentazioni
 - alcuni casi di successo sono Amazon, Ebay, Groupon, Netflix, Spotify, Uber, Zalando e Zoom
 - è uno stile architetturale recente e ancora in “rapido movimento” – la sua definizione non si può ancora considerare completamente consolidata



Influenze sull'architettura a microservizi

- Ecco i metodi, le pratiche e le tecnologie principali che hanno influenzato l'architettura a microservizi – e che sono necessarie o raccomandate per la sua adozione
 - i metodi per lo sviluppo agile e lean
 - 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 come guida per la progettazione del software
 - le pratiche ingegneristiche per la delivery del software
 - la virtualizzazione e l'automazione dell'infrastruttura
 - il cloud computing e i container
 - la Continuous Delivery



- Note terminologiche

- L'“architettura a microservizi” viene talvolta chiamata informalmente solo “microservizi”
 - ad es., “i microservizi sono uno stile architeturale per ...”
 - tuttavia, in questo modo è difficile capire se il termine “microservizi” si riferisce allo stile architeturale oppure a un insieme di microservizi



* Architettura a microservizi

- Una definizione preliminare
 - l'*architettura a microservizi* è uno stile architeturale per lo sviluppo di una singola applicazione distribuita – organizzata come un insieme di microservizi
 - i *microservizi* sono dei servizi “piccoli” e autonomi, eseguiti come processi distinti, che lavorano insieme comunicando mediante meccanismi leggeri



Architettura a microservizi e SOA

- Ecco alcune similitudini e alcune differenze significative tra l'architettura a microservizi e la SOA
 - similitudini
 - sono entrambe delle architetture a servizi, basate sulla nozione di servizio
 - differenze
 - la portata dell'architettura a microservizi è una singola applicazione – mentre la portata della SOA è l'insieme di tutte le applicazioni e i sistemi software di un'organizzazione
 - i microservizi sono “piccoli” – questo termine deve suggerire l'utilizzo di metodi e pratiche agili per lo sviluppo del software
 - i microservizi comunicano mediante “meccanismi leggeri” – e non “pesanti” come quelli usati nella SOA



- 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 – un'interruzione di servizio dell'applicazione è anche un'interruzione del business
 - è richiesta agilità di business – deve essere possibile offrire rapidamente funzionalità e caratteristiche sempre migliori o innovative nell'ambito di questa applicazione
 - l'applicazione potrebbe essere stata inizialmente sviluppata in modo monolitico – deve essere reingegnerizzata affinché possa soddisfare tutti questi requisiti



Architettura a microservizi

- La soluzione dell'architettura a microservizi
 - l'applicazione viene definita come l'integrazione e la composizione di un insieme di microservizi
 - ogni *microservizio* è un servizio coeso e "piccolo", che rappresenta una specifica capacità di business in modo autocontenuto e in esecuzione in un proprio processo
 - i microservizi sono autonomi – possono essere realizzati con tecnologie software differenti
 - i microservizi comunicano sulla base di meccanismi leggeri
 - i microservizi possono essere rilasciati in modo completamente automatizzato e indipendente tra loro
 - i client dell'applicazione interagiscono con i microservizi in modo indiretto, tramite un API gateway
 - è in genere necessario un minimo di controllo centralizzato e di supporto infrastrutturale per i microservizi

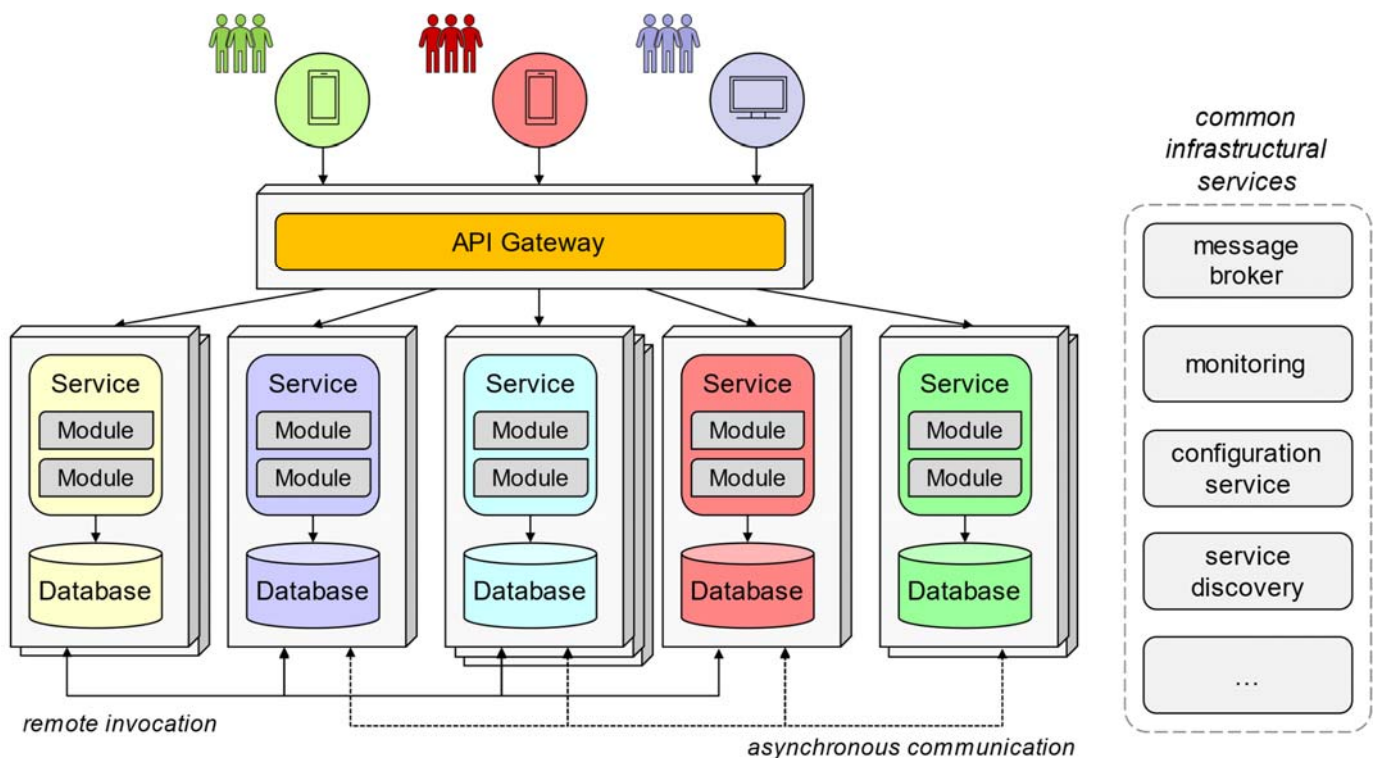
11

Architettura a microservizi

Luca Cabibbo ASW



Architettura a microservizi



12

Architettura a microservizi

Luca Cabibbo ASW



* Caratteristiche dell'architettura a microservizi

- ❑ Discutiamo ora in maggior dettaglio la soluzione proposta dall'architettura a microservizi



- I microservizi sono servizi

- ❑ Innanzitutto, i microservizi sono **servizi** – come tali, devono soddisfare i diversi principi per la progettazione dei servizi
 - contratto e astrazione (incapsulamento) – i microservizi hanno un'interfaccia separata dalla loro implementazione, mediante la quale è possibile interagire con essi
 - accoppiamento debole e autonomia – sia dei microservizi che dei rispettivi team di sviluppo
 - componibilità e riusabilità – i microservizi possono essere composti in modo agile
 - scopribilità – da parte dei diversi team di sviluppo e anche a runtime
 - i microservizi sono (preferibilmente) stateless
 - i microservizi sono accessibili in rete, e la loro locazione è trasparente



I microservizi sono coesi

- Ciascun microservizio è **coeso** – rappresenta una singola capacità di business, che deve svolgere bene
 - i microservizi sono unità di sviluppo fortemente modulari – per favorire la modificabilità dell’applicazione e dei suoi microservizi
 - la coesione e la modularità dei microservizi sono una caratteristica molto più importante della “dimensione” dei microservizi



I microservizi sono “piccoli”

- Ciascun microservizio è **“piccolo”**
 - purtroppo, il termine “microservizio” sembra suggerire che la “dimensione” sia la loro caratteristica più importante
 - in realtà, la dimensione dei microservizi è per lo più irrilevante
 - un microservizio ben progettato è un servizio che può essere sviluppato da un team agile, con un tempo di sviluppo e rilascio breve e con una collaborazione minima con gli altri team
 - a tal fine, è importante soprattutto che i microservizi siano fortemente modulari, e che soddisfino i diversi principi per la progettazione dei servizi
 - dunque, “micro” va considerata per lo più un’etichetta per distinguere i microservizi dai servizi delle altre architetture a servizi



I microservizi sono “piccoli”

- Ciascun microservizio è “piccolo”
 - in ogni caso, i microservizi sono in genere “piccoli”
 - non c’è una misura precisa della dimensione “corretta” per i microservizi
 - un microservizio potrebbe corrispondere al software che un team di sviluppo agile può inizialmente sviluppare e rilasciare in una singola iterazione
 - se ciascun microservizio è “piccolo”, allora l’intera applicazione comprende in genere una molteplicità di microservizi
 - ad es., Spotify è basato su qualche migliaio di microservizi, gestiti da circa 500 team di sviluppo agili (ogni team è composto da al più 8 persone), con rilascio in container (circa 20000 rilasci al giorno) (*dati: 2016/2020*)



I microservizi sono componibili

- L’intera applicazione è definita come la **composizione** di una molteplicità di microservizi
 - la composizione deve essere flessibile – in modo da poter integrare nuovi microservizi quando vengono sviluppati
 - viene di solito realizzata come una coreografia di microservizi – basata sulla notifica asincrona di eventi di dominio
 - questo consente infatti di aggiungere nuovi microservizi alla coreografia, senza modificare i microservizi già esistenti
 - ove necessario, la composizione può essere realizzata anche come un’orchestrazione – basata sull’invocazione remota



- Microservizi e autonomia

- Ciascun microservizio costituisce un'entità software **autonoma** e separata dagli altri microservizi
 - i diversi microservizi possono essere sviluppati con tecnologie software differenti e rilasciati in esecuzione in processi separati
 - ogni microservizio viene realizzato nel modo più opportuno possibile
 - l'eterogeneità viene risolta mediante l'uso di meccanismi di comunicazione interoperabili
 - l'esecuzione di ciascun microservizio in un ambiente di esecuzione autonomo consente di associare a ogni microservizio le proprie configurazioni e dipendenze specifiche



Microservizi e autonomia

- 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 ciascun microservizio separatamente dagli altri
 - sostituire un microservizio con una sua nuova implementazione, se e quando necessario
 - (se questo non è vero, allora i “microservizi” non sono veri microservizi)
 - l'autonomia si riferisce anche ai team di sviluppo
 - i team possono prendere decisioni di progetto autonome riguardo ai microservizi di cui sono responsabili



- Microservizi e capacità di business

- Ciascun microservizio rappresenta una (singola) specifica capacità di business in modo autocontenuto
 - una **capacità di business** (*business capability*) è un'attività o un compito che un'organizzazione fa o può fare – al fine di generare valore di business
 - è una nozione dalla disciplina del business architecture modeling
 - ad es., per un'organizzazione che si occupa di commercio elettronico, esempi di capacità di business sono la gestione degli ordini, la gestione dell'inventario, le spedizioni e la raccomandazione di prodotti
 - le capacità di business si concentrano su “che cosa” fa un'organizzazione – in opposizione a “come” l'organizzazione fa quelle cose per svolgere il proprio business



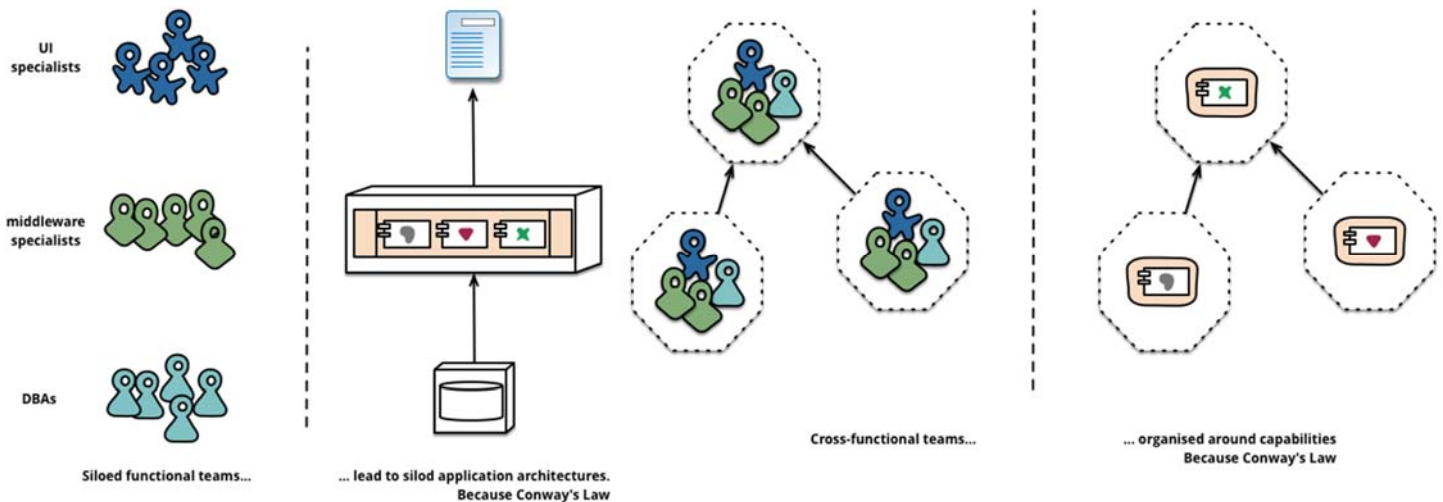
Microservizi e capacità di business

- Ciascun microservizio rappresenta una (singola) specifica capacità di business in modo autocontenuto
 - ciascun microservizio si occupa di tutti gli aspetti relativi alla propria specifica capacità di business – logica di business, UI e gestione dei dati persistenti
 - una decomposizione in microservizi basati su capacità tecniche non è invece opportuna
 - una decomposizione basata su capacità di business
 - favorisce meglio la scalabilità e la modificabilità che non una decomposizione basata su capacità tecniche
 - è compatibile con i team di sviluppo cross-funzionali (o full stack) – anziché con i team di sviluppo mono-funzionali



Microservizi e capacità di business

- ❑ Le precedenti osservazioni sui team di sviluppo 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



23

Architettura a microservizi

Luca Cabibbo ASW



- Meccanismi di comunicazione leggeri

- ❑ La comunicazione tra microservizi deve supportare l'interoperabilità tra i microservizi
 - l'invocazione remota è spesso basata su chiamate REST mediante HTTP
 - la comunicazione asincrona può essere basata sullo scambio di messaggi e la notifica di eventi, mediante un message broker leggero
 - la comunicazione asincrona offre numerosi vantaggi rispetto all'invocazione remota
 - sostiene modificabilità, scalabilità, disponibilità e affidabilità
 - tuttavia, offre un supporto limitato alla consistenza dei dati
 - in pratica, può essere usata sia la comunicazione asincrona che l'invocazione remota

24

Architettura a microservizi

Luca Cabibbo ASW



Meccanismi di comunicazione leggeri

- Nella SOA, invece, vengono usati meccanismi e protocolli di comunicazione “pesanti”
 - che supportano sia la comunicazione tra servizi che la loro composizione e orchestrazione
 - la logica di composizione viene definita in modo centralizzato – questo tende in genere ad inibire i cambiamenti, anziché favorirli

25

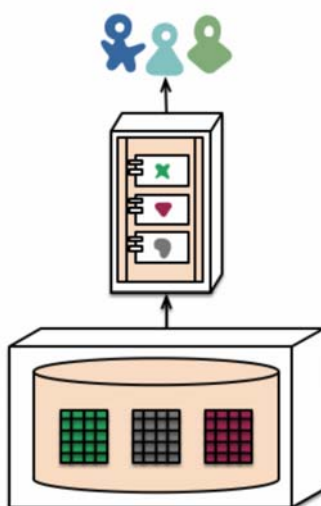
Architettura a microservizi

Luca Cabibbo ASW

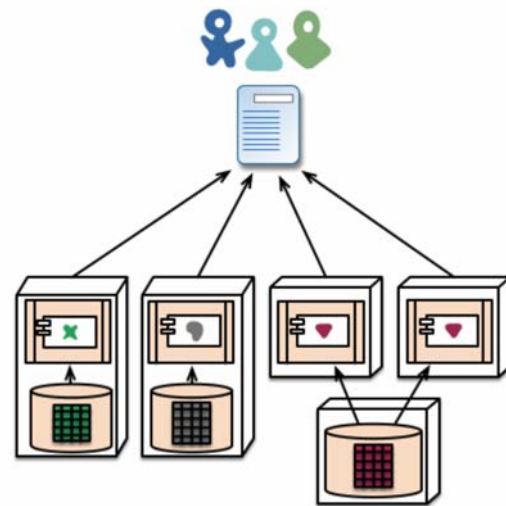


- Gestione decentralizzata dei dati

- Di solito, ciascun microservizio incapsula la memorizzazione e l'accesso ai propri dati – mediante una propria base di dati separata e “privata”
 - dunque, un microservizio si occupa direttamente anche della gestione dei dati per la capacità di business che rappresenta



monolith - single database



microservices - application databases

26

Architettura a microservizi

Luca Cabibbo ASW



Gestione decentralizzata dei dati

- L'uso di basi di dati separate
 - sostiene l'autonomia dei microservizi – ogni microservizio può gestire i propri dati usando la tecnologia più opportuna
 - sostiene una coesione alta e un accoppiamento basso
 - sostiene la scalabilità – una base di dati monolitica potrebbe invece ostacolare la scalabilità dell'intera applicazione



Microservizi e gestione dei dati



- Tuttavia, la gestione decentralizzata dei dati pone anche dei problemi e solleva delle sfide
 - nelle interrogazioni distribuite
 - se un microservizio ha bisogno dei dati gestiti da un altro microservizio, non li può accedere direttamente dalla relativa base di dati
 - per combinare i dati di più microservizi, non è in genere possibile effettuare un join “globale” tra le diverse basi di dati
 - una possibilità è usare dei join distribuiti “applicativi” (*API Composition*) – che però hanno diversi svantaggi
 - un'altra soluzione comune consiste nell'usare viste materializzate per i pattern di accesso ai dati più importanti, usando il pattern *CQRS (Command-Query Responsibility Segregation)*

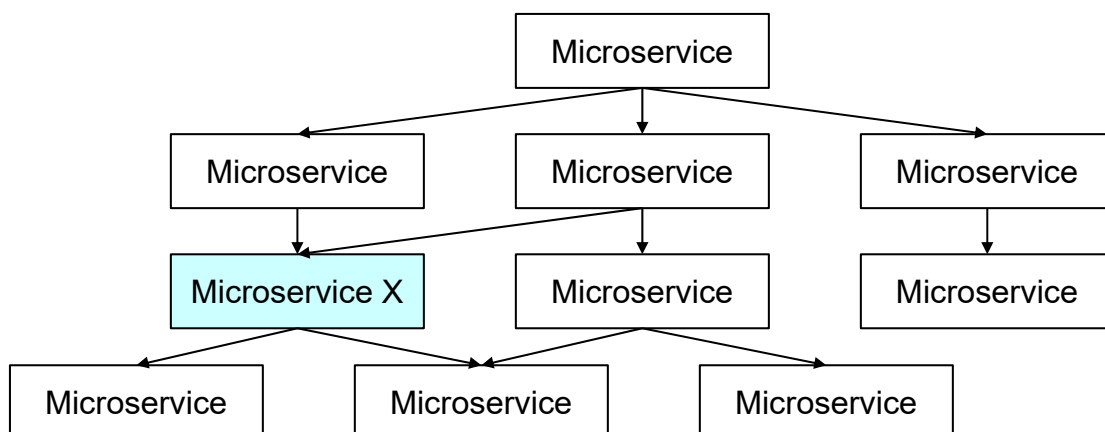


- Tuttavia, la gestione decentralizzata dei dati pone anche dei problemi e solleva delle sfide
 - nelle transazioni distribuite
 - le transazioni distribuite “tradizionali” sincrone (come il 2PC) sostengono una consistenza forte dei dati
 - tuttavia, possono limitare disponibilità e scalabilità
 - con i microservizi, una soluzione comune è basata sulle saga – una **saga** è una sequenza di transazioni atomiche locali, coordinate in modo asincrono
 - le saga sostengono però solo una consistenza “debole” dei dati



- Rilascio indipendente e automatizzato

- Il rilascio di nuove versioni dei microservizi nell’ambiente di produzione è importante per sostenere agilità e disponibilità
 - i microservizi vengono aggiornati e rilasciati individualmente
 - ad es., un rilascio potrebbe riguardare l’aggiornamento di un microservizio X da una versione A ad una nuova versione B





Rilascio indipendente e automatizzato

- Il rilascio di nuove versioni dei microservizi
 - deve poter avvenire frequentemente
 - deve avvenire in modo affidabile
 - per minimizzare i rischi del rilascio, che sono non solo tecnici ma anche di “business”
 - per questo, avviene in modo automatizzato – adottando la *Continuous Delivery (CD)*
 - in particolare, la deployment pipeline e il rilascio senza interruzioni di servizio



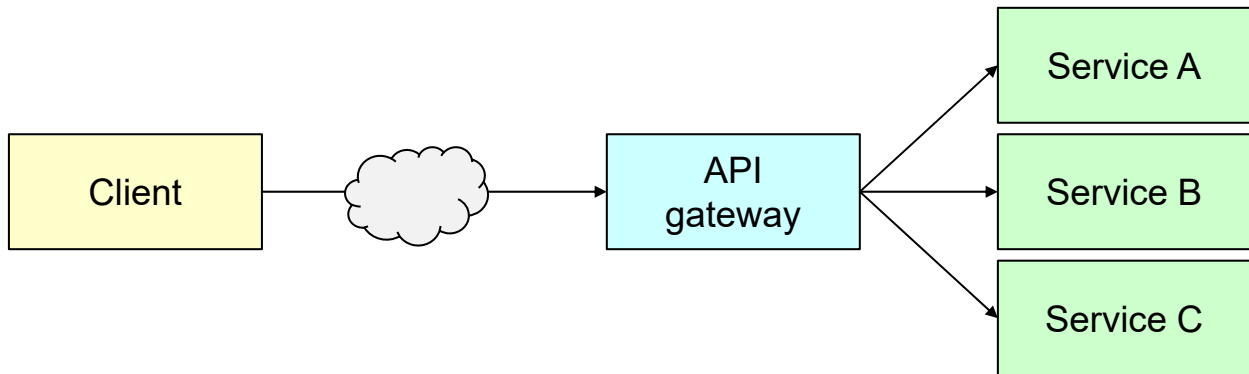
- API gateway

- Un'applicazione a microservizi può esporre le proprie funzionalità al suo esterno sotto forma di applicazione web, app mobile e/o come API REST – per consentire l'accesso mediante una molteplicità di dispositivi e di client
 - i client finali però non devono interagire direttamente con i microservizi dell'applicazione
 - per evitare un accoppiamento diretto con i microservizi
 - per evitare interazioni con i client basate su richieste multiple e a grana fine
 - per fornire a ciascun tipo di client un'API specializzata
 - è preferibile che questi client accedano all'applicazione mediante un punto di accesso integrato e unificato



API gateway

- Un *API gateway* è un servizio (lato server) che fornisce un'API specializzata e un punto di accesso unificato a un'applicazione composta da più (micro-)servizi



API gateway

- Un *API gateway* è un servizio (lato server) che fornisce un'API specializzata e un punto di accesso unificato a un'applicazione composta da più (micro-)servizi
 - agisce da intermediario verso i client esterni – e incapsula l'accesso ai servizi interni
 - effettua il routing delle chiamate dei client ai servizi specifici di interesse
 - può fornire un'API personalizzata per ciascun tipo di client
 - può assumere ulteriori responsabilità aggiuntive, come ad es.
 - la gestione della sicurezza (Single-Sign On)
 - la composizione di API
 - la conversione di protocolli
 - supportare il monitoraggio



- Integrazione e composizione di microservizi

- L'integrazione e la composizione di microservizi può avvenire in diversi modi
 - la modalità più comune è a livello della logica di business
 - invocazione di microservizi – basata su chiamate remote – ad es., REST su HTTP
 - comunicazione asincrona – scambio di messaggi e notifica di eventi



Integrazione e composizione di microservizi

- L'integrazione e la composizione di microservizi può avvenire in diversi modi
 - a livello di interfaccia utente (UI)
 - ciascun microservizio può avere una propria UI, che può contenere link alle UI di altri microservizi
 - ad es., un'applicazione web basata su una singola pagina, con sezioni differenti per i diversi microservizi
 - i diversi microservizi potrebbero non avere la necessità di interagire tra di loro
 - a livello della base di dati
 - più microservizi accedono a una base di dati condivisa
 - questa modalità di integrazione è spesso sconsigliata – non rispetta l'autonomia dei microservizi e può indurre un accoppiamento alto



- Un minimo di gestione centralizzata

- Nell'architettura a microservizi, molte decisioni di progetto possono essere effettuate in modo decentralizzato e autonomo
 - per garantire la coerenza dell'intera applicazione, è però necessario basare l'architettura su alcune scelte comuni a tutti i microservizi
 - queste scelte riguardano soprattutto alcuni interessi tecnici e infrastrutturali



Infrastruttura per i microservizi

- Le principali decisioni tecniche e infrastrutturali da prendere a livello dell'intero sistema – in modo standardizzato e centralizzato
 - protocolli e standard di comunicazione
 - ad es., la scelta del message broker
 - approcci standard per
 - la gestione dei dati di configurazione
 - la scoperta dei microservizi (service discovery)
 - la sicurezza
 - l'automazione dell'infrastruttura e il rilascio automatizzato dei microservizi (continuous delivery)
 - il monitoraggio e il logging



Infrastruttura per i microservizi



- La gestione dei dati di configurazione è particolarmente complessa nell'architettura a microservizi
 - ogni microservizio richiede dei dati di configurazione differenti per i diversi ambienti in cui va rilasciato
 - viene di solito usato un approccio basato su configurazioni separate dal codice dei microservizi (*externalized configuration*) – gestite in un servizio centralizzato (*configuration service*) mediante un configuration server
 - è spesso necessario anche il supporto per il cambiamento dinamico dei dati di configurazione dei microservizi – ad es., per supportare il feature toggling



Infrastruttura per i microservizi



- Un servizio di service discovery consente ai diversi microservizi di scoprirsi l'un l'altro – per abilitare le interazioni tra di essi
 - per registrare i microservizi in esecuzione, ciascuno con le sue istanze
 - per supportare la scoperta a runtime dei microservizi e il bilanciamento del carico tra le diverse istanze dei microservizi
 - per monitorare lo stato di salute dei microservizi



- ❑ Per la sicurezza, è spesso necessario implementare dei meccanismi di
 - autenticazione degli utenti – spesso nell'API gateway
 - autorizzazione degli accessi – di solito se ne occupano i singoli microservizi, in modo decentralizzato

- ❑ Per il monitoraggio, si utilizza un'architettura distribuita
 - la raccolta di metriche e log avviene nei singoli microservizi
 - la memorizzazione e l'elaborazione di questi dati avviene invece in modo centralizzato

- ❑ Per il rilascio automatizzato, è necessario un approccio standardizzato comune a tutti i microservizi
 - basato sulla *continuous delivery*



* Conseguenze dell'architettura a microservizi

- ❑ Discutiamo ora le principali conseguenze dell'architettura a microservizi



- Microservizi e scalabilità

- L'architettura a microservizi sostiene la scalabilità
 - l'applicazione viene definita come un insieme di microservizi – i microservizi sono replicati
 - sono decomposizioni lungo gli assi Y e X del cubo della scalabilità
 - i microservizi comunicano preferibilmente in modo asincrono, e sono preferibilmente stateless
 - l'adozione di ambienti virtuali e di pratiche per l'automazione dell'infrastruttura consentono di creare e distruggere istanze di microservizi in modo dinamico, elastico e a grana fine, separatamente per ciascun microservizio



- Microservizi e disponibilità

- L'architettura a microservizi sostiene la disponibilità
 - tolleranza ai guasti – di istanze di microservizi e di nodi
 - i microservizi sono replicati
 - l'applicazione viene eseguita in un ambiente fisicamente replicato
 - monitoraggio dei microservizi e dei nodi e riavvio automatico dei microservizi falliti
 - isolamento dei guasti – il guasto di un microservizio non deve provocare guasti a cascata in altri microservizi
 - ad es., se un microservizio fa una chiamata a un'istanza che nel frattempo è fallita
 - i microservizi devono essere progettati per sostenere un opportuno isolamento dei guasti



- Microservizi e agilità di business

- L'architettura a microservizi sostiene la capacità di implementare e rilasciare rapidamente funzionalità e caratteristiche sempre migliori o innovative
 - l'agilità di business è una forma forte di modificabilità – sostenuta dalla coesione e modularità dei microservizi
 - i diversi team lavorano (in modo agile) sui vari microservizi dell'applicazione – ciascuno alla propria velocità e con un impatto minimale sugli altri microservizi e sugli altri team
 - i team operano in modo autonomo, per implementare e far evolvere i propri microservizi in modo ottimale
 - i team possono effettuare e rilasciare rapidamente cambiamenti ai propri microservizi – e possono anche introdurre e integrare nuovi microservizi



- Dal monolito ai microservizi

- L'architettura a microservizi viene spesso applicata per ristrutturare un'applicazione che è stata inizialmente realizzata in modo monolitico
 - l'applicazione monolitica può fornire un supporto scarso e insoddisfacente per una o più qualità importanti – come modificabilità, scalabilità e rilasciabilità
 - l'architettura a microservizi consente di sostenere meglio queste qualità



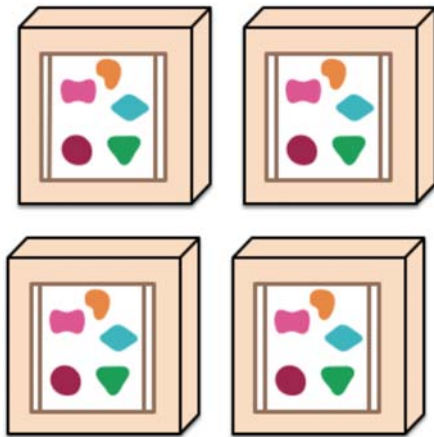
Dal monolito ai microservizi

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

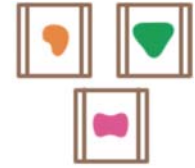
A monolithic application puts all its functionality into a single process...



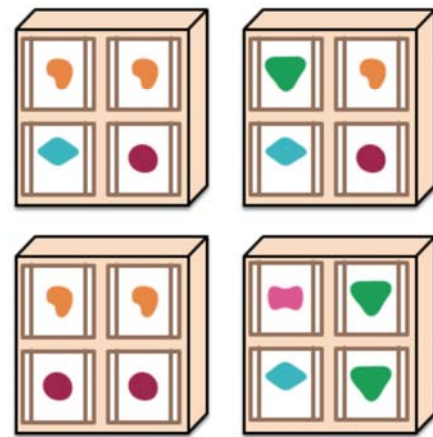
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



47

Architettura a microservizi

Luca Cabibbo ASW



Dal monolito ai microservizi

- Ma come decomporre in microservizi un'applicazione monolitica?
 - ciascun microservizio deve rappresentare una singola capacità di business in modo autocontenuto
 - la decomposizione avviene in modo iterativo e incrementale – applicando il pattern *Strangler*, che suggerisce di
 - aggiungere un API gateway di fronte all'applicazione
 - l'API gateway inoltra le richieste degli utenti finali verso i nuovi microservizi oppure verso la vecchia applicazione
 - sostituire (in modo iterativo e incrementale) pezzi specifici di funzionalità dell'applicazione con nuovi microservizi
 - contestualmente, queste funzionalità vengono “spente” dall'applicazione originale
 - la migrazione delle funzionalità avviene dunque in modo graduale

48

Architettura a microservizi

Luca Cabibbo ASW



- Decomposizione di funzionalità e dati



- Nella progettazione dell'architettura a microservizi
 - da un punto di vista strutturale, è necessario decidere in quanti e quali servizi decomporre il sistema
 - e quali sono le responsabilità di ciascun servizio, in termini di funzionalità e dati
 - da un punto di vista dinamico, è necessario anche decidere le modalità di interazione tra i servizi (anche in relazione alla gestione dei dati), per formare un sistema coerente
 - in effetti, questi due aspetti sono correlati e non indipendenti tra loro
 - questa decomposizione può essere guidata da opportuni criteri



Decomposizione di funzionalità e dati



- Nella progettazione dell'architettura a microservizi
 - possibili criteri di decomposizione sono di tipo “integratore” e “disintegratore” (quando è meglio tenere due responsabilità insieme oppure separarle?), relativi a funzionalità e dati
 - ad es., la separazione delle funzionalità sostiene la loro modificabilità, scalabilità e rilasciabilità – ma tenere unite delle funzionalità in un unico servizio ne facilita l'interazione e la condivisione di codice – inoltre questa scelta ha impatto sulla decomposizione dei dati
 - ad es., la separazione dei dati sostiene l'ottimizzazione della gestione dei dati – ma tenere uniti i dati in un'unica base di dati semplifica la gestione di transazioni e interrogazioni
 - in genere, questi criteri possono fornire suggerimenti contrastanti – al fine di supportare requisiti di qualità differenti – e quindi è di solito necessario identificare dei compromessi



* Ulteriori considerazioni

- ❑ Facciamo ora delle ulteriori considerazioni sull'architettura a microservizi



- 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
 - si basa su due categorie principali di pattern – tattici e strategici
 - la progettazione strategica riguarda l'architettura del software
 - la progettazione tattica riguarda la progettazione interna dei singoli elementi architettonici
 - nel contesto dei microservizi sono rilevanti soprattutto i pattern strategici *Bounded Context* e *Context Map*



Bounded Context [DDD]



- **Bounded Context** rappresenta il principio di progettazione strategica fondamentale di DDD
 - un possibile modo per identificare i contesti limitati di un'applicazione è in relazione alla divisione del lavoro nel dominio di business di interesse, per capacità di business o in relazione ai gruppi di lavoro dell'organizzazione



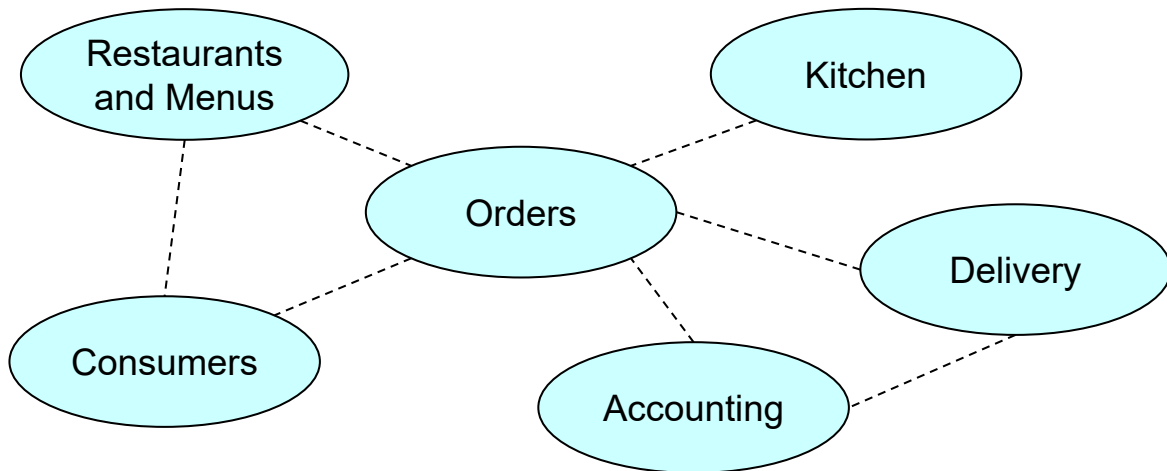
Context Map [DDD]



- **Context Map** è un altro pattern strategico fondamentale di DDD – per guidare la decomposizione complessiva di un'applicazione
 - ragionare in termini di singoli contesti limitati è spesso insufficiente – è necessaria anche una visione globale del sistema
 - una mappa dei contesti descrive le relazioni tra i contesti limitati – che rappresentano le necessità di comunicazione e di integrazione richieste tra i diversi contesti limitati
 - questa mappa rappresenta anche le relazioni tra i team responsabili dei diversi contesti limitati



- I contesti limitati identificati e una mappa dei contesti che li correlano possono essere utilizzati per guidare la decomposizione in microservizi di un'applicazione



- ciascun microservizio può implementare un intero bounded context – ma è anche possibile decomporre i bounded context in microservizi a grana più fine



- Microservizi e architettura esagonale

- Una scelta comune per l'architettura interna di ciascun microservizio è l'architettura esagonale
 - ogni microservizio può essere implementato come un "esagono" (servizio)
 - l'interno (la logica di business) rappresenta una capacità di business in modo autocontenuto
 - le porte rappresentano le interfacce (fornite e richieste) del microservizio
 - gli adattatori consentono di implementare queste porte in modo interoperabile



- 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
 - per effettuare una buona decomposizione basata su capacità di business, viene in genere suggerito un approccio “monolith first”
 - l'architettura complessiva è importante
 - molte scelte possono essere effettuate localmente e autonomamente ai microservizi
 - tuttavia, solo una buona architettura di dominio complessiva può sostenere lo sviluppo indipendente dei microservizi

57

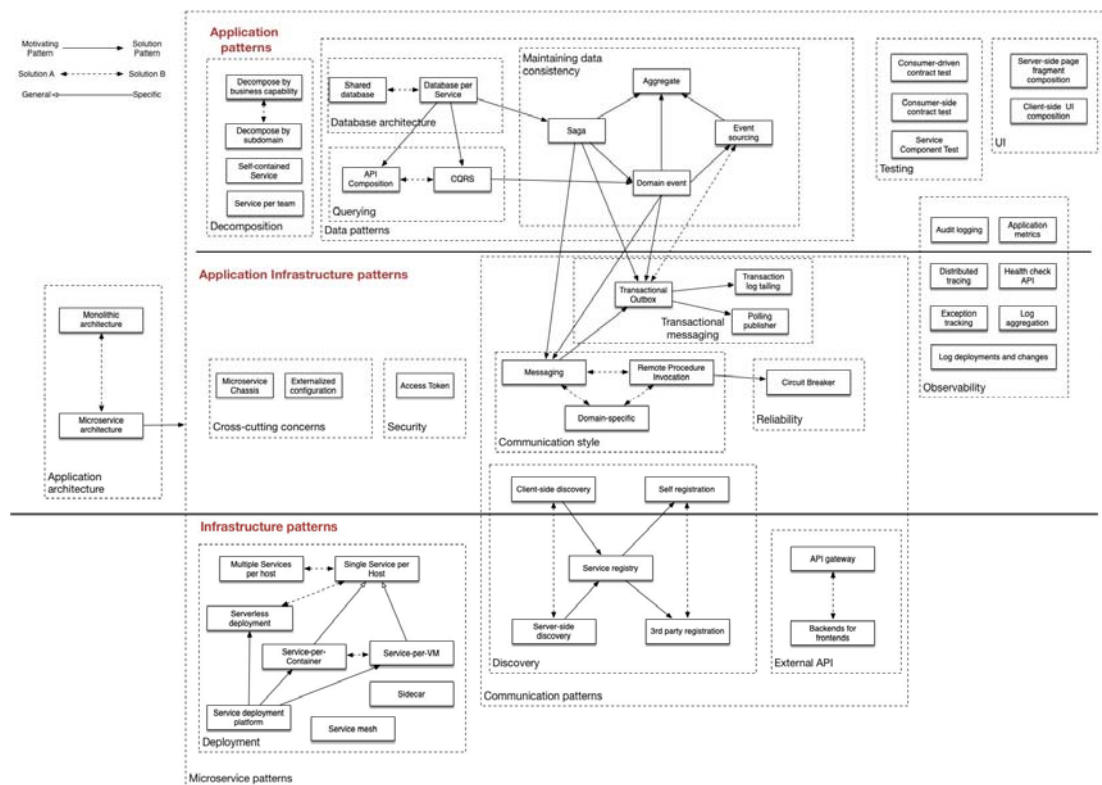
Architettura a microservizi

Luca Cabibbo ASW



- Pattern per microservizi

- **Microservices Patterns** [Richardson] (microservices.io) presenta un pattern language per microservizi – con oltre 40 pattern



58

Architettura a microservizi

Luca Cabibbo ASW



Pattern per microservizi

- **Microservices Patterns** [Richardson] (microservices.io) presenta un pattern language per microservizi – con oltre 40 pattern
 - consente di comprendere i tanti problemi da affrontare nell'architettura a microservizi – e le soluzioni che possono essere applicate
 - ecco i principali ambiti a cui si riferiscono questi pattern
 - architettura applicativa – monolito vs microservizi
 - applicazione – decomposizione in servizi, gestione dei dati, interfaccia utente, verificabilità
 - infrastruttura applicativa – comunicazione, sicurezza, osservabilità, gestione dei dati di configurazione e di altri interessi trasversali
 - infrastruttura e deployment – pattern relativi all'ambiente di deployment ed esecuzione e al rilascio dei microservizi



* Discussione

- L'architettura a microservizi è uno stile architetturale che organizza un'applicazione come un insieme di servizi che sono
 - altamente coesi e debolmente accoppiati
 - organizzati attorno a capacità di business
 - che comunicano mediante protocolli leggeri
 - che vengono fatti evolvere in modo agile da team di sviluppo autonomi
 - che vengono rilasciati indipendentemente e in modo automatizzato
 - 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
 - è uno stile architetturale complesso, che consente di ottenere benefici in diverse aree



Discussione

- ▣ Alcune organizzazioni stanno usando già da diversi anni l'architettura a microservizi con successo – si tratta soprattutto di organizzazioni con un modello di business basato su Internet
 - tuttavia, questo stile architetturale non è adatto per tutte le applicazioni – e nemmeno per tutti i team di sviluppo
 - richiede molteplici competenze, in ambito tecnologico, infrastrutturale e metodologico
 - inoltre, non è passato abbastanza tempo per poter esprimere un giudizio oggettivo su questo stile architetturale
 - non è nemmeno sempre chiaro quali siano le scelte obbligatorie e quelle opzionali, quelle raccomandate e quelle semplicemente possibili
 - in ogni caso, il mio suggerimento, oggi, è di cercare di acquisire le competenze necessarie per i microservizi



Discussione



- ▣ Attorno all'architettura a microservizi circolano tanti (falsi) miti, tante verità (incomplete) e soprattutto tante incomprensioni





- Attorno all'architettura a microservizi circolano tanti (falsi) miti, tante verità (incomplete) e soprattutto tante incomprensioni
 - i microservizi sono adatti a ogni applicazione
 - i microservizi sono basati sulla vecchia idea della SOA
 - i microservizi sono semplicemente servizi piccoli
 - i microservizi riguardano (solo) la delivery del software
 - i microservizi riguardano (solo) l'uso dei container e dell'orchestrazione di container
 - i microservizi riguardano (solo) l'uso di framework e strumenti specifici
 - i microservizi si ottengono decomponendo la logica di business del monolito – ma la base di dati può rimanere monolitica
 - i microservizi richiedono (solo) un cambiamento tecnologico
 - ...