

→ Una prospettiva di servizi

E' necessario un cambiamento nella progettazione, implementazione

- I sistemi tradizionali (distribuiti) sono sistemi singoli sviluppati dall'alto verso il basso

Ogni componente dovrebbe essere in grado di servire diversi sistemi in diversi modi

↳ un'organizzazione mantiene il sistema

→ No Interface

- Gli eService tradizionali si basano su TCP/IP e su protocolli proprietari (SOAP).
- L'attenzione si è spostata dalle interfacce (IDL, WSDL) alla comunicazione e conversazione (protocolli)
- REST è la tendenza attuale per la costruzione di applicazioni

HTTP è il protocollo di incollaggio
(metodi + controllo + media type)

↳ Il Web si basa su client / server sulle risorse

→ Microservizi vs servizi

- I servizi tendono ad essere grandi e monolitici, i microservizi sono componibili, sostituibili, più facili da manutenzione

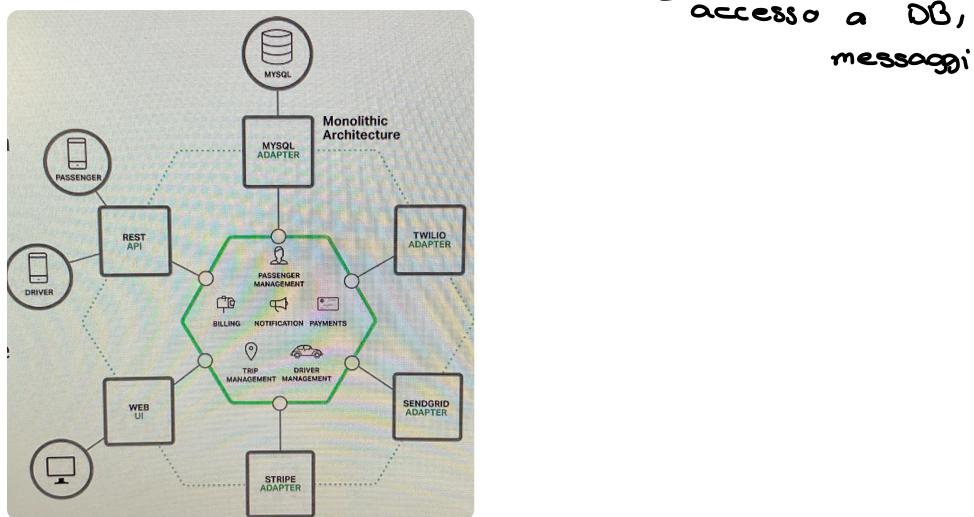
↳ autonami
piccoli e concentrati sul fare bene una cosa

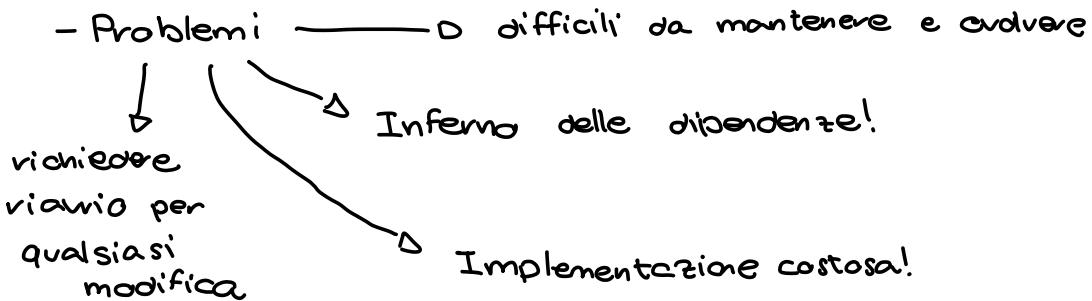
-D Microservizi

- Un'architettura microservices
 - mette ogni elemento di funzionalità in un servizio separato
 - ↓
 - rende ogni servizio un'unità di distribuzione indipendente
- I microservizi accelerano la consegna dei dati, riducendo la comunicazione e il coordinamento.
- Le applicazioni basate su microservizi favoriscono protocolli più semplici e leggeri (REST)
- Organizzato attorno alle capacità aziendali
 - ↑
 - miglior coordinamento
 - ↓
 - L'elevata coesione facilita la gestione e la manutenzione

-D Applicazioni Monolitiche

- Al centro dell'applicazione c'è la logica di business, che è implementato da moduli che definiscono servizi
- Intorno al nucleo ci sono adattatori che si interfacciano con il mondo esterno.





-D Architettura dei microservizi

- Invece di costruire un unico mostruoso, monolitico, l'idea è quella di dividere la tua applicazione in set di servizi più piccoli e interconnessi
- Ogni microservizio è una mini-applicazione che espone, un API, un'interfaccia
- Il gateway API è responsabile di attività come il bilanciamento, cache, controllo accessi.
- Il modello dell'architettura dei microservizi ha un impatto significativo sulla relazione con il DB
 - ogni servizio ha il proprio schema di DB

-D VANTAGGI

- ① Affronta il problema della complessità
- ② Consente a ciascun servizio di essere sviluppato in autonomia da un team
- ③ Distribuita in modo indipendente
- ④ Consente di scalare ogni servizio in modo indipendente

-D SVANTAGGI

- ① Uno svantaggio è il nome stesso
- ② Un altro grave inconveniente è la complessità che deriva dal fatto che a l'applicazione di microservizi è un sistema distribuito
- ③ DB partizionato
- ④ Testare
- ⑤ Difficile modificare microservizi collegati

-D Distribuzione

→ D è molto complessa

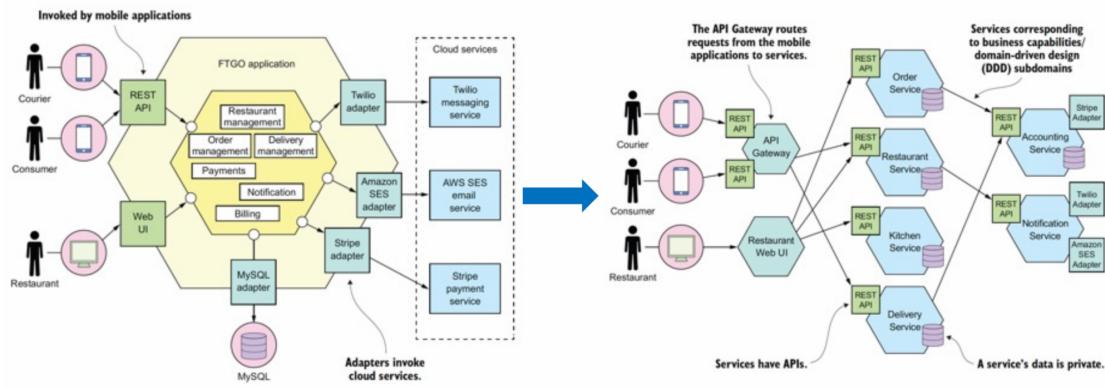
Un app di microservizi
è tipicamente costituita
da un grande numero di
servizi. Più parti mobili
che devono essere configurate,
distribuite, ridimensionate e
monitorate.

Una app. monolitica
viene semplicemente
distribuita su un insieme
di server (=) con
un bilanciamento del
carico.

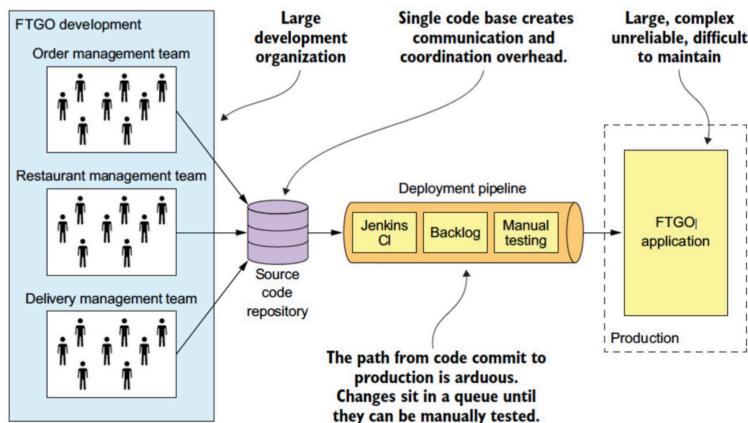
PaaS
come
Cloud Foundry

La distribuzione corretta di
un applicazione richiede un
maggiore controllo dei metodi
di distribuzione da parte degli
sviluppatori e un alto livello di
automazione.

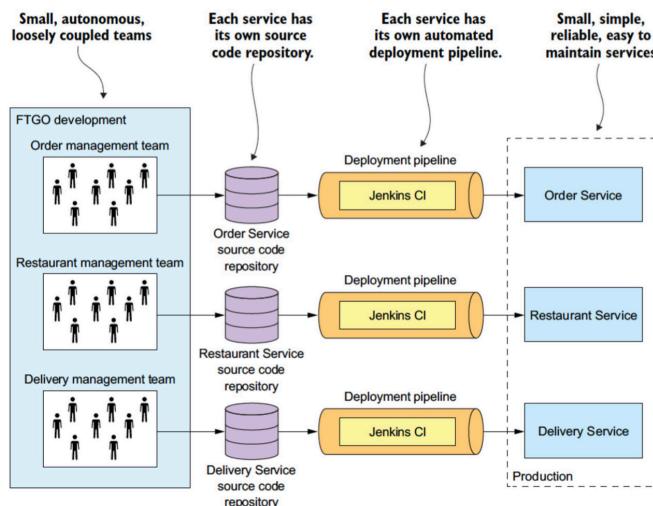
→ Monolitico vs Architettura Microservizi



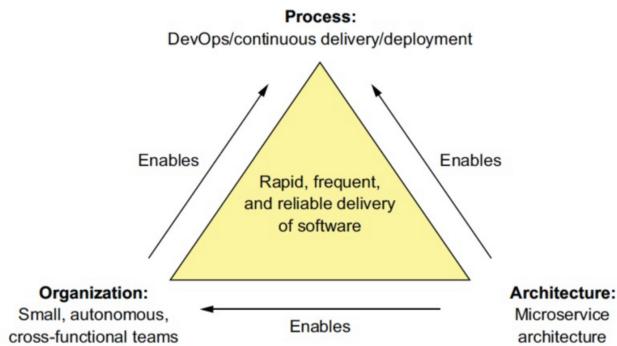
→ Monolitico



→ Microservice architecture

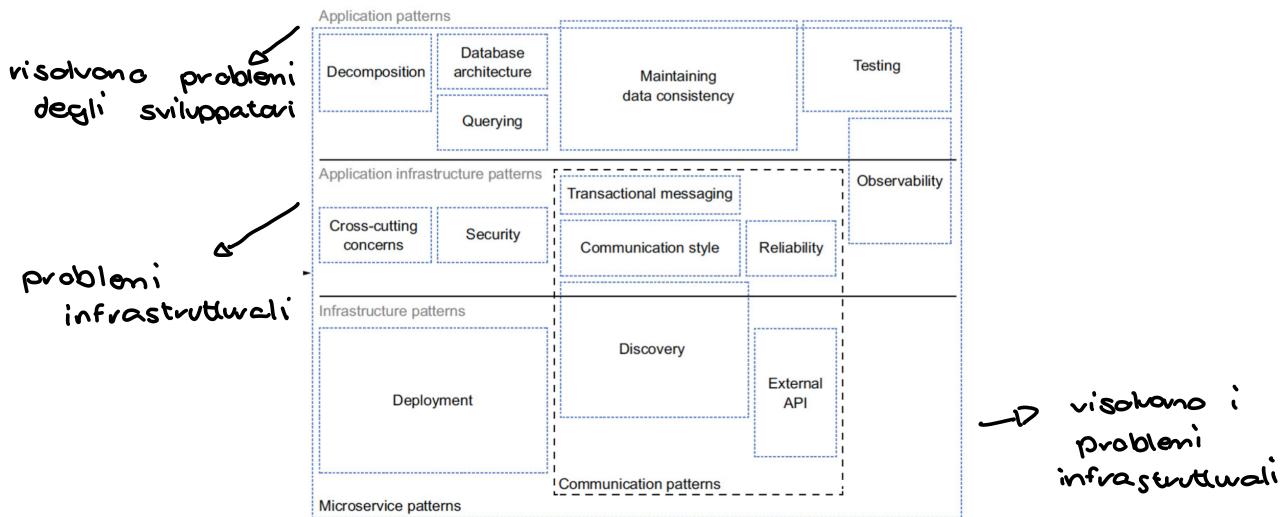


-D Microservices



- D Un **pattern** è una soluzione riutilizzabile a un problema che si verifica in un contesto.
- D Un **modello software** risolve un'architettura software o un problema di progettazione mediante definire un insieme di elementi software collaboranti
- D Un **pattern** descrive il contesto in cui si applica
- D **Pattern Structure**
 - Uno schema ti costringe a descrivere altri aspetti critici ma spesso ignorati di una soluzione
- Forza:** i problemi che devi affrontare quando risoli un problema
- **Contesto risultante:** le conseguenze dell'applicazione di un pattern
- **Modelli correlati:** i cinque diversi tipi di relazione

→ Patterns dei Microservizi



→ Hexagonal architecture style

→ mettono la business logic al centro

↓

ha uno o più adattatori di I/O

→ dipendono dalla logica aziendale

→ Monolithic vs Microservice patterns

↓

struttura l'applicazione come un singolo componente eseguibile / distribuibile

→ Struttura l'applicazione come una raccolta di servizi in dipendenti

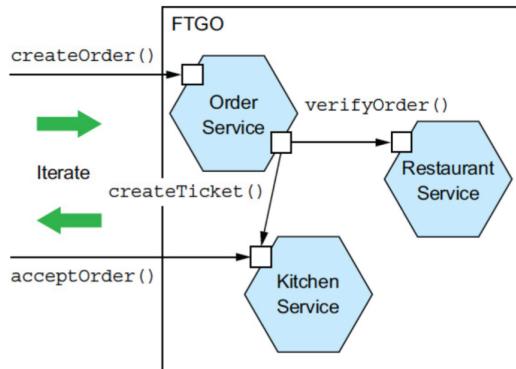
→ Un **servizio** è autonomo, indipendente, componente software distribuibile che implementa alcune funzionalità. Tutte le interazioni avvengono tramite le sue API

-D Definire i microservizi

① Trovare i requisiti dell'applicazione

② Decomporre i servizi

③ Determinare ogni servizio API

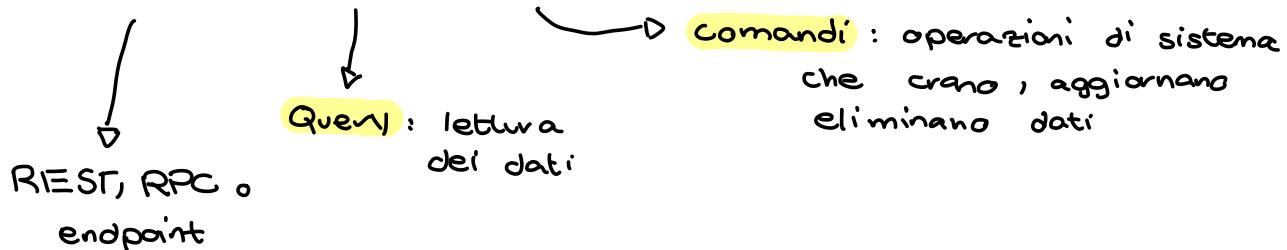


-D Identificazione delle operazioni di sistema

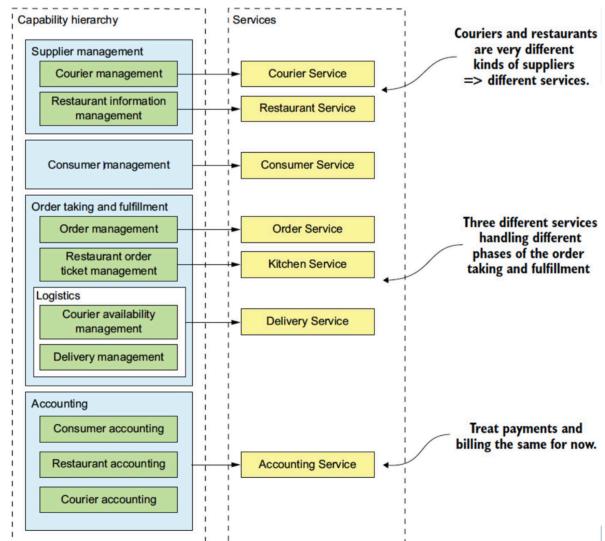
① Definisco il modello di dominio di alto livello costituito dalle classi chiave (vocabolario)

② Identifica le operazioni del sistema e descrive ogni comportamento

-D Operazioni di sistema



-D Un esempio



→ Scamponi per pattern di sottodomini

acquisiamo la conoscenza di un dominio

→ Un design guidato dal dominio (DDD) è un approccio per la costruzione di complessi

→ DDD e Microservizi

Il concetto di sotto-dominio con il proprio modello di dominio è un ottimo modo per eliminare le classi di divinità

→ Il concetto DDD di sottodomini e contesti limitati si adatta bene a servizi all'interno di un'architettura

→ Le classi di Dio sono le classi grandi che vengono utilizzate in un'applicazione

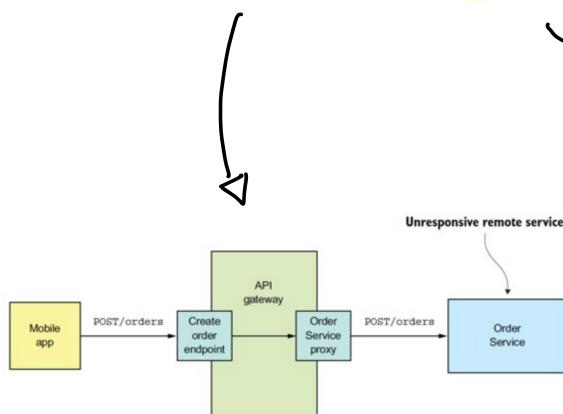
→ Comunicazione tra processi (IPC)

↓
comunicazione
a sincrona
AMQP e STOMP

→ I servizi possono utilizzare meccanismi di comunicazione sincroni basati su richiesta/risposta

→ REST basato su HTTP

→ Schema dell'interruttore di circuito

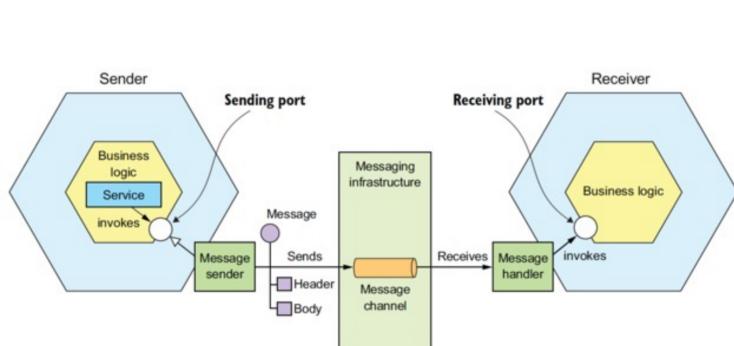


→ In un sistema distribuito, ogni volta che un servizio effettua una richiesta sincrona a un altro servizio

-> Sviluppo di proxy API

- ↓
Schema dell'interruttore di circuito
- ↓
Limitare il numero di richieste in sospeso da un cliente a un servizio
- Timeout di rete

-> Modello di messaggistica asincrona



- point-to-point , conversazione tra due ↗
- publish-subscribe , consegna ogni messaggio a tutti gli allegati consumatori

-> API di servizio basata sulla messaggistica

- Le operazioni di un servizio possono essere invocate utilizzando uno dei due diversi stili di interazione

↓
API unidirezionale

→ API di tipo richiesta/risposta asincrona

- risposte
- i tipi e i formati
- bipi di messaggi di comando e accetta
- canale del messaggio di comando