Documentazione Architetturale

---

1. Introduzione

La presente documentazione descrive in modo completo l'architettura **Microservices Architecture**, progettata per operare in un contesto ad alta variabilità di carico, con requisiti stringenti in termini di scalabilità, sicurezza, affidabilità e manutenibilità.

L'obiettivo è fornire una descrizione architetturale strutturata conforme ai principi IEEE 1016, includendo requisiti, driver architetturali, alternative considerate, trade-off e rappresentazioni UML.

---

2. Contesto e Problema

Il sistema deve sostenere utenti concorrenti, transazioni sicure e integrazione con sistemi esterni garantendo scalabilità, resilienza e disponibilità continua.

---

3. Requisiti del Sistema

3.1 Requisiti Funzionali

Il sistema deve supportare le seguenti funzionalità principali:

- **Gestione del catalogo prodotti**: inserimento, aggiornamento e consultazione prodotti.
- **Ricerca e navigazione prodotti**: filtri avanzati e ricerca per parole chiave.
- **Gestione del carrello**: aggiunta, modifica e rimozione prodotti.
- **Creazione e gestione ordini**: conferma acquisto e tracciamento stato.
- **Integrazione con gateway di pagamento**: elaborazione sicura delle transazioni.
- **Integrazione con servizi di spedizione**: gestione consegne e aggiornamenti.
- **Gestione account utente**: autenticazione, autorizzazione e gestione profilo.

3.2 Requisiti Non Funzionali

Performance: tempo di risposta ridotto per operazioni critiche.

Scalabilità: supporto a elevati volumi di utenti concorrenti.

Disponibilità: elevato uptime tramite ridondanza.

Sicurezza: cifratura dati e controlli di accesso.

Manutenibilità: evoluzione con impatto minimo.

---

4. Trade-Off Architetturali

Performance vs Complessità

Un'architettura distribuita migliora la scalabilità ma aumenta la complessità operativa.

Disponibilità vs Costi

La ridondanza migliora l'affidabilità ma incrementa i costi infrastrutturali.

Sicurezza vs Performance

I controlli di sicurezza introducono overhead ma garantiscono protezione dei dati.

SEZIONE DIAGRAMMI

---

5. Diagramma del Contesto

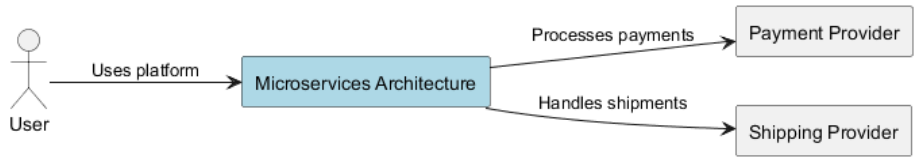

Figure 1: Diagramma del Contesto

Il diagramma del contesto rappresenta il sistema come **black-box** e mostra gli attori/sistemi esterni con cui interagisce. È utile per chiarire **confini**, **responsabilità** e integrazioni con provider esterni (es. pagamenti e spedizioni).

6. Diagramma dei Componenti

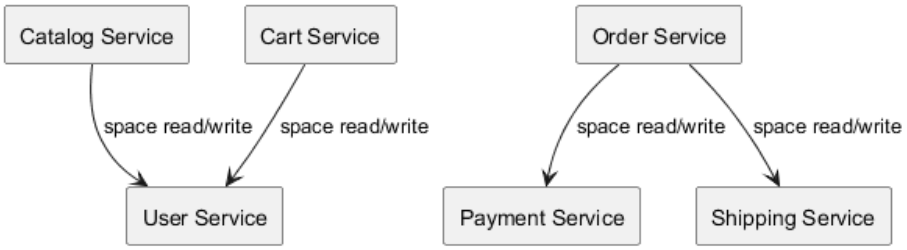

Figure 2: Diagramma dei Componenti

Il diagramma dei componenti mostra la scomposizione logica del sistema in servizi/moduli. Nel caso corrente sono presenti i seguenti elementi principali:

- Catalog Service
- Cart Service
- Order Service
- User Service
- Payment Service
- Shipping Service

Le dipendenze principali (parziali) sono: - Catalog Service → User Service (space read/write) - Cart Service → User Service (space read/write) - Order Service → Payment Service (space read/write) - Order Service → Shipping Service (space read/write)

7. Diagramma del Deployment

Figure 3: Diagramma del Deployment

Il diagramma di deployment rappresenta la distribuzione fisica dei componenti su nodi infrastrutturali (es. web/app/database). Serve a evidenziare **separazione dei livelli**, scalabilità e isolamento dei failure. È utile anche per discutere aspetti di rete, bilanciamento e fault tolerance.

8. Diagramma di Sequenza

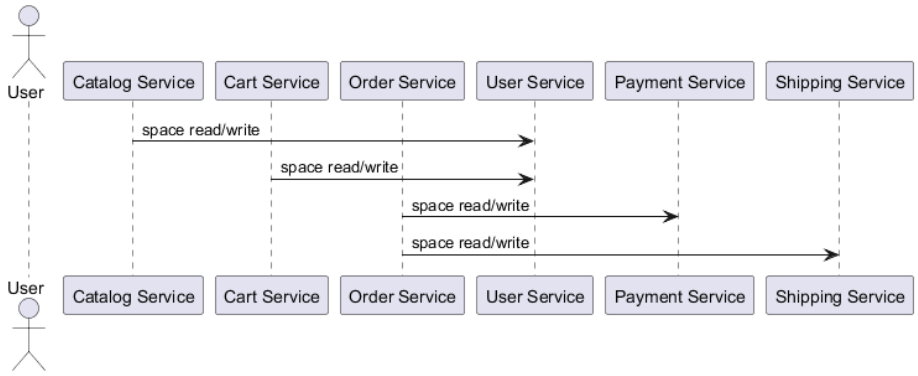

Figure 4: Diagramma di Sequenza

Il diagramma di sequenza descrive il comportamento dinamico del sistema, evidenziando l'ordine temporale delle interazioni tra attore e servizi. In un contesto e-commerce tipicamente include: consultazione catalogo, gestione carrello, creazione ordine, pagamento e avvio spedizione.

---

9. Diagramma di Sicurezza

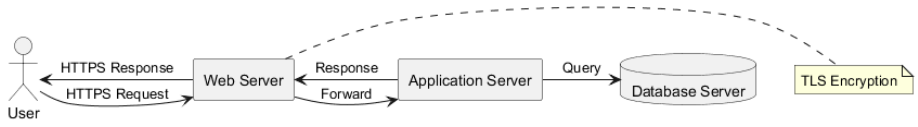

Figure 5: Diagramma di Sicurezza

Il diagramma di sicurezza evidenzia i principali confini di fiducia (trust boundaries) e le misure di protezione: TLS in transito, autenticazione/autorizzazione, validazione input, logging e auditing. È utile per ragionare su threat model e punti critici (pagamenti, dati utenti).

---

10. Conclusioni

L'architettura **Microservices Architecture** rappresenta una soluzione equilibrata tra scalabilità, sicurezza e manutenibilità, fornendo una base solida per evoluzione futura e deployment distribuito.