Applicazioni Web & Modello Client–Server

Tecnologie client-side/server-side, linguaggi del Web e comunicazione

Francesco Gobbi

29 ottobre 2025

Applicazioni web: generalità

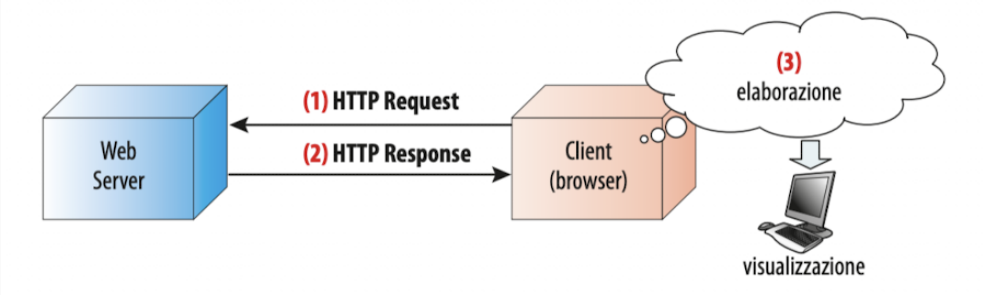
- **Tecnologie del Web**: client-side e server-side.
- **Linguaggi del Web**: linguaggi di mark-up e linguaggi di programmazione.

Idea chiave

Un'applicazione web (web-app) è un software fruito via tecnologie e protocolli del Web.

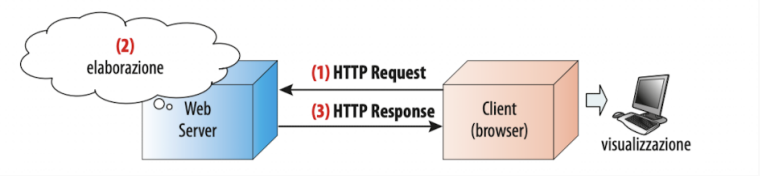
Tecnologia Client-side: elaborazione nel browser

- Le elaborazioni avvengono sul **client** (tipicamente il *browser*).
- Flusso tipico: (1) HTTP Request → (2) HTTP Response → (3) **elaborazione** nel client e visualizzazione.
- Esempi: HTML, CSS, JavaScript. Il **codice è visibile** nel sorgente della pagina.
- Anche da file locale: non serve per forza un web server per visualizzare risorse puramente client-side.



Tecnologia Server-side: elaborazione sul server

- Le elaborazioni avvengono sul **server** (web server che esegue codice dinamico).
- Flusso tipico: (1) HTTP Request → (2) **elaborazione** sul server → (3) HTTP Response.
- Esempi: PHP, Java Servlet, ... Necessario **connettersi a un web server**.
- Il codice **non è visibile** al client: il browser vede solo il risultato (HTML generato).



Nota

Nelle applicazioni web reali **client-side e server-side si integrano**: non si scelgono in modo arbitrario, dipende dall'elaborazione richiesta.

Linguaggi del Web: Mark-up vs Programmazione

- **Linguaggi di mark-up**: descrivono *documenti strutturati* con *tag* (es. **HTML**, **XML**).
- **Linguaggi di programmazione**: scrivono *sequenze di istruzioni*.
- Evoluzione: approcci **ibridi** (AJAX (Asynchronous JavaScript and XML) ecc.) fondono client e server in un unico flusso.
- In **HTML5** alcuni tag consentono logica; la distinzione è meno netta ma *utile didatticamente*.

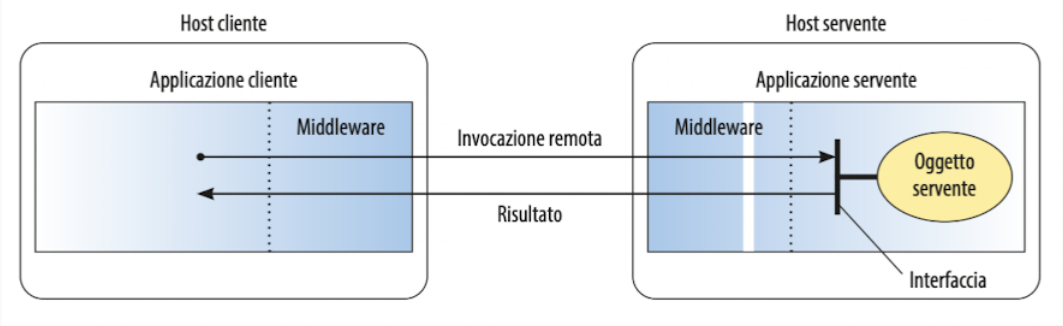
Il modello client-server

- Insieme di **host** con processi **server** (gestiscono risorse/servizi) e **client** (richiedono accesso).
- **Sono i processi** ad essere client o server, non gli host; uno stesso host può eseguire più processi.
- Attenzione: da non confondere **servizio** (entità astratta offerta) con **server** (processo/macchina che lo espone).

Caratteristiche del modello

Molti utenti concorrenti; logica applicativa complessa; grandi archivi distribuiti; forti requisiti di sicurezza; sistemi transazionali.

Il modello client-server

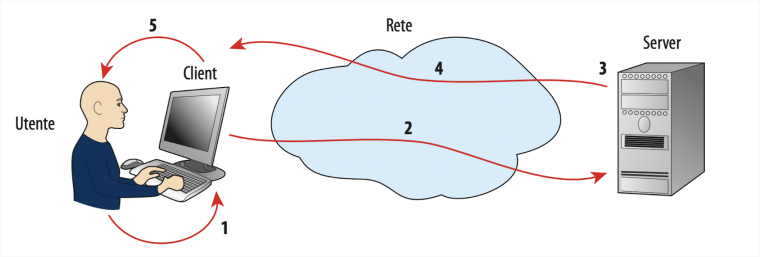


Schema di funzionamento

1. Il **client** invia una *richiesta* al server.
2. Il **server** la riceve (in attesa sulla porta del servizio).
3. Il server **esegue** il servizio (spesso generando un *thread* dedicato).
4. Il server invia la **risposta** e i dati.
5. Il client riceve e visualizza.

Servizi tipici

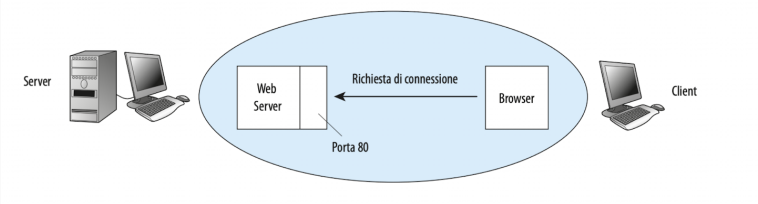
Telnet; **HTTP** (browser $\leftrightarrow$ web server); **FTP**; altri: *SMTP*, *IMAP*, *NFS*, *NIS*, ...



Esempi: Socket –> host(IP) e porta

Differenza Client-Server

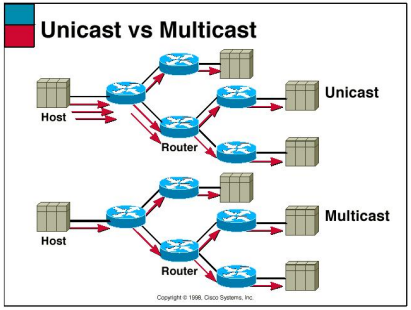
Un client, quindi, per comunicare con un server usando il protocollo TCP/IP deve, per prima cosa, "connettersi" al socket dell'host dove il server è in esecuzione specificando l'indirizzo IP della macchina e il numero di porta sulla quale il server è in ascolto.



- Il **server** è in *ascolto* su una **porta** dell'host.
- Il **client** si connette specificando **indirizzo IP** dell'host e **numero di porta**.
- Il **socket** è la coppia ⟨indirizzo IP, porta⟩ che identifica un servizio.
- Esempio HTTP: web server in ascolto sulla **porta 80**.

Tipi di comunicazione possibili

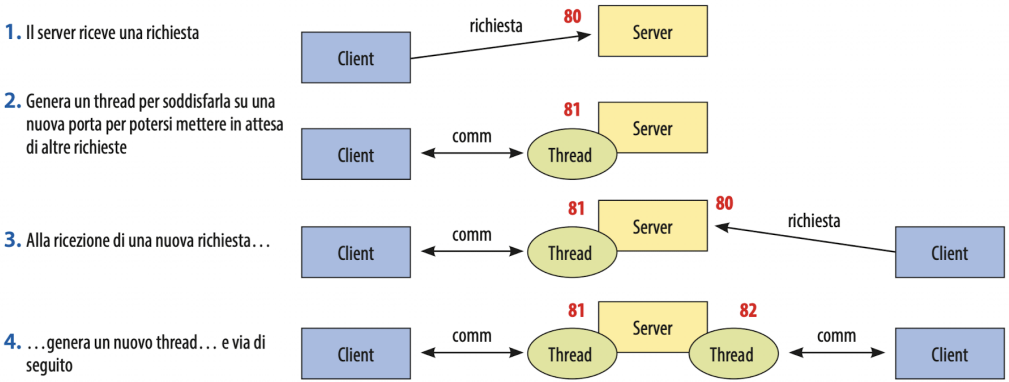
- **Unicast**: il server gestisce *una* connessione per volta.
- **Multicast / concorrente**: il server può servire *più client* contemporaneamente.
- Strategia tipica: la richiesta arriva sulla porta del servizio (es. 80), il server **sposta** la comunicazione su una **nuova porta** e avvia un **thread** dedicato, mantenendo libera la porta principale.



Tipi di comunicazione possibili

Esempio

Un **web server** deve garantire gli stessi servizi a molti utenti: resta in ascolto sulla porta 80, crea thread e usa porte 81, 82, ... per le singole sessioni.



Livelli e strati: definizioni chiave

- Architetture client–server spesso organizzate **a livelli (tier)**: ogni livello può essere un nodo o un gruppo di nodi.
- Ogni livello può fungere da **server** per il precedente e da **client** per il successivo.
- Vista **a strati**: livelli di astrazione funzionale (presentazione, logica, dati).
- In pratica convivono: *deployment a livelli* e *organizzazione a strati*.

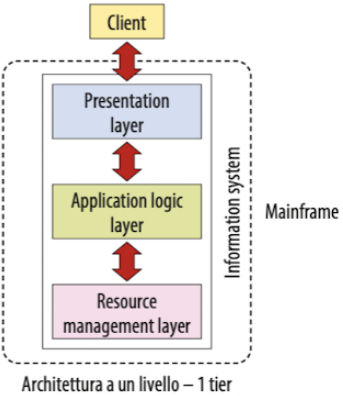
Modello 3-tier: i tre grandi strati

- **Presentation / front-end**: interfaccia verso l'utente.
- **Application logic / middle tier**: regole di business e coordinamento.
- **Data / back-end**: persistenza e accesso ai dati (DAL).

Nomenclatura nel Web: moduli web (HTML/Servlet/PHP/ASP), BLL/RML per la logica, DBMS per i dati.

Architettura a un livello (1 tier)

- Scenario storico: **mainframe** con terminali "stupidi".
- Tutta l'elaborazione su un unico calcolatore; terminali per sola I/O.
- Non è propriamente client–server; utile come riferimento.



Architettura a un livello – 1 tier

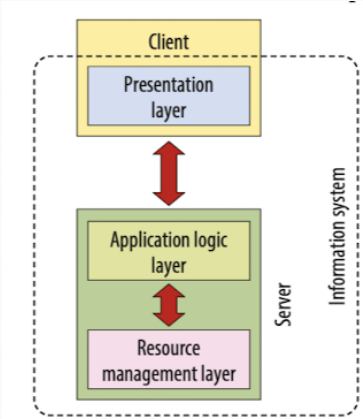
Architettura a due livelli (2 tier)

- Nata con le LAN (fine anni '80). Due ruoli distinti: **server** e **client**.
- **Thin-client**: logica e dati sul server; presentazione al client.
- **Thick-client**: dati sul server; presentazione e parte di logica al client.

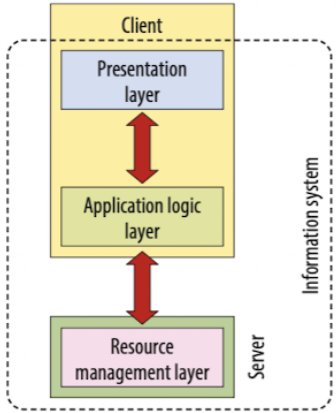
Limiti del modello del 2 tier:

- **Poca scalabilità**: il server gestisce connessioni e stato di molte sessioni.
- Carico centralizzato $\Rightarrow$ limite al numero di client concorrenti.
- Il **thick-client** ha traghettato verso le **architetture distribuite moderne**.

Architettura a due livelli (2 tier)



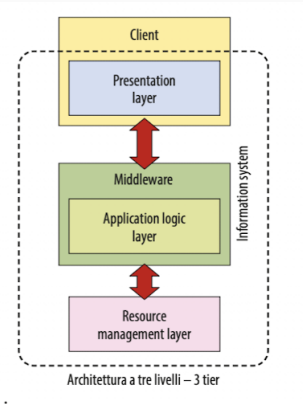
Architettura a due livelli – 2 tier – modello thin



Architettura a due livelli – 2 tier – modello thick

Architettura a tre livelli (3 tier)

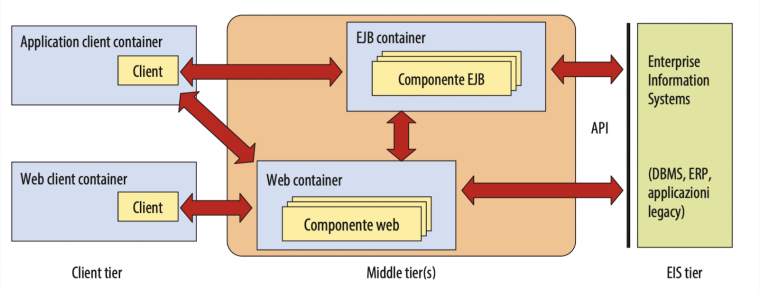
- Strati: **Presentation**, **Business/Middle**, **Data/Back-end**.
- Introduce **middleware** per distribuire carico e favorire **scalabilità** e **manutenibilità**.
- Vantaggi: scaling orizzontale, deployment indipendente, sicurezza per strato.



Architettura a tre livelli – 3 tier

Esempio 3-tier: piattaforma Java EE

- **Client tier**: web/app client.
- **Middle tier**: web container ed EJB container con componenti applicativi.
- **EIS/Data tier**: DBMS, ERP, sistemi legacy accessibili via API.
- Nota: corrispondenza *strati–livelli* non sempre netta (più livelli fisici per lo stesso strato logico).



Architetture a N livelli (multi-tier)

- Generalizzazione del 3 tier con **livelli intermedi** (cache, API gateway, code, microservizi, servizi di dominio).
- Obiettivi: **suddivisione fine delle responsabilità**, **resilienza**, **scalabilità** e **deployment** indipendente.
- Il termine **multi-tier** indica la scomposizione tra strati logici e più livelli fisici.