SIMULATORE ESAME COMPLETO

API REST & Cloud Computing - 90 Domande

SEZIONE 1: FONDAMENTI DI API (Domande 1-10)

1. Che cos'è un'API? Definizione e scopo principale

Domanda chiusa:
Un'API (Application Programming Interface) è principalmente:
A) Un linguaggio di programmazione
B) Un'interfaccia che permette a due applicazioni di comunicare in modo standardizzato
C) Un database
D) Un sistema operativo

Risposta: B) Un'interfaccia che permette a due applicazioni di comunicare in modo standardizzato.
Un'API definisce regole, protocolli e strumenti per permettere la comunicazione controllata tra software diversi, nascondendo i dettagli implementativi.

---

2. Differenza tra API Locale e API Web

Domanda aperta:
Spiega la differenza tra API locali (Desktop API) e API web (Web API).

Risposta suggerita:

- **API Locale:** Comunicazione tra componenti dello stesso sistema operativo (es. Windows API per file system). Non richiedono rete.
- **API Web:** Basate su protocolli di rete come HTTP/HTTPS, accessibili tramite Internet. Permettono comunicazione client-server tra sistemi diversi (es. API REST).
  Differenza chiave: Le API locali sono intra-sistema; le API web sono inter-sistema.

---

3. Cosa significa API Cloud?

Domanda chiusa:
Le API Cloud si caratterizzano per:
A) Accessibilità solo locale
B) Essere fornite da provider cloud, accessibili via Internet e basate su protocolli standard
C) Essere sempre gratuite
D) Non richiedere autenticazione

Risposta: B) Essere fornite da provider cloud, accessibili via Internet e basate su protocolli standard.
Esempio: API di AWS, Azure, Google Cloud permettono di gestire risorse cloud via API REST.

---

4. API Pubblica vs API Privata - Differenze

Domanda aperta:
Quali sono le differenze principali tra API pubbliche e API private? Fai almeno 3 esempi.

Risposta suggerita:

- **API Pubblica:** Disponibile al pubblico, documentata pubblicamente, può essere gratuita o a pagamento. Esempi: Twitter API, OpenWeather API, GitHub API.
- **API Privata:** Utilizzata solo all'interno dell'organizzazione, non esposta pubblicamente, sotto controllo diretto dell'azienda.
  Differenze: Le pubbliche hanno SLA e supporto; le private hanno contratto di servizio interno.

---

5. Information Hiding in API

Domanda chiusa:
Il principio di "information hiding" in un'API significa:
A) Nascondere l'indirizzo IP del server
B) Esporre solo le funzionalità necessarie, nascondendo i dettagli implementativi
C) Usare solo HTTPS
D) Criptare tutti i dati

Risposta: B) Esporre solo le funzionalità necessarie, nascondendo i dettagli implementativi.
L'API è un contratto: il client usa solo ciò che serve, senza conoscere come è implementato internamente.

---

6. Quale fra questi è un corretto scopo di un'API?

Domanda chiusa:
A) Rendere il sistema più lento
B) Eliminare la necessità di database
C) Standardizzare e semplificare la comunicazione tra sistemi
D) Aumentare i costi di sviluppo

Risposta: C) Standardizzare e semplificare la comunicazione tra sistemi.
Un'API ben progettata standardizza come i sistemi interagiscono, facilitando l'integrazione.

---

7. Analogia del ristorante per capire le API

Domanda aperta:
Spiega l'analogia del ristorante per comprendere il concetto di API.

Risposta suggerita:
Un'API è come il menu di un ristorante:

- Il **cliente** (client) non ha bisogno di sapere come il cuoco prepara il piatto (implementazione interna)
- Il **cliente** legge il menu (API) e ordina ciò che desidera
- Il **cameriere** riceve l'ordine e lo passa al cuoco (server)

- Il **cameriere** restituisce il piatto (risposta API)
  Analogamente, un'API definisce come richiedere servizi senza conoscere i dettagli tecnici.

---

8. REST vs SOAP - Differenze fondamentali

Domanda chiusa:
La principale differenza tra REST e SOAP è:
A) REST è più nuovo di SOAP
B) REST usa HTTP/HTTPS in modo nativo, SOAP è più complesso e usa XML esclusivamente
C) SOAP è più veloce di REST
D) Non c'è differenza

Risposta: B) REST usa HTTP/HTTPS in modo nativo e sfrutta i metodi HTTP; SOAP è un protocollo complesso indipendente.
REST è uno stile architetturale semplice; SOAP è uno standard formale e più pesante.

---

9. Quando dovresti usare un'API REST?

Domanda aperta:
Descrivi tre scenari in cui usare un'API REST è la scelta migliore.

Risposta suggerita:

1. **Microservizi web:** Quando hai servizi che devono comunicare via HTTP (es. backend Node.js + database)
2. **Applicazioni mobile:** Le app mobile consumano API REST per accedere ai dati del server
3. **Integrazioni third-party:** Quando servizi terzi devono integrarsi con la tua piattaforma (es. pagamenti, analytics)
   REST è leggera, scalabile e sfrutta HTTP natively.

---

10. Livelli di maturità API - Definizione generale

Domanda chiusa:
Il modello di maturità delle API (Richardson Maturity Model) classifica le API in quanti livelli?
A) 2 livelli
B) 3 livelli
C) 4 livelli
D) 5 livelli

Risposta: C) 4 livelli (0, 1, 2, 3).
Dal livello 0 (RPC) al livello 3 (HATEOAS), ognuno aggiunge più caratteristiche REST.

SEZIONE 2: VERBI HTTP (Domande 11-25)

11. GET - Quale delle seguenti è corretta?

Domanda chiusa:
GET /api/users/123 è usato per:
A) Creare un nuovo utente
B) Leggere i dati dell'utente 123
C) Eliminare l'utente 123
D) Aggiornare l'utente 123

Risposta: B) Leggere i dati dell'utente 123.
GET è il verbo per lettura ed è idempotente (non modifica lo stato del server).

---

12. POST - Idempotente o non-idempotente?

Domanda chiusa:
POST è:
A) Idempotente
B) Non-idempotente
C) A volte sì, a volte no
D) Deprecato

Risposta: B) Non-idempotente.
Eseguire POST due volte crea due risorse diverse. POST è per azioni non-idempotente come creazione.

---

13. PUT vs PATCH - Cosa cambia?

Domanda aperta:
Spiega la differenza tra PUT e PATCH con un esempio pratico di aggiornamento di un utente.

Risposta suggerita:

- **PUT:** Sostituisce **interamente** la risorsa. Richiede tutti i campi. Se non includi `name`, il campo viene cancellato.
  PUT /api/users/123
  { "name": "Marco", "email": "marco@example.com", "role": "admin" }
- **PATCH:** Aggiorna **parzialmente** solo i campi specificati.
  PATCH /api/users/123
  { "email": "marco.nuovo@example.com" }
  Dopo PATCH, il `name` e `role` rimangono invariati; solo l'email cambia.

---

14. DELETE - Status code corretto

Domanda chiusa:
Quale status code è più appropriato per DELETE /api/users/123 dopo aver eliminato l'utente?
A) 200 OK
B) 201 Created

C) 204 No Content
D) 404 Not Found

Risposta: C) 204 No Content.
DELETE di successo restituisce tipicamente 204 (niente da restituire, solo successo).

---

15. HEAD - A cosa serve?

Domanda chiusa:
HEAD è simile a GET, ma:
A) Restituisce solo gli header, non il body
B) Crea una risorsa
C) Elimina una risorsa
D) Non esiste

Risposta: A) Restituisce solo gli header, non il body.
HEAD è usato per verificare se una risorsa esiste senza scaricare il body.

---

16. OPTIONS - Preflight CORS

Domanda chiusa:
OPTIONS è usato principalmente per:
A) Leggere dati
B) Scoprire quali metodi HTTP sono supportati (usato per CORS preflight)
C) Creare risorse
D) Eliminare risorse

Risposta: B) Scoprire quali metodi HTTP sono supportati e per CORS preflight.
OPTIONS /api/users → Restituisce Allow: GET, POST, PUT, DELETE, PATCH, OPTIONS

---

17. Idempotenza - Quali verbi sono idempotenti?

Domanda aperta:
Elenca tutti i verbi HTTP idempotenti e spiega perché lo sono.

Risposta suggerita:
Verbi idempotenti:

- **GET:** Leggere la stessa risorsa 100 volte = stesso risultato
- **HEAD:** Identico a GET, quindi idempotente
- **PUT:** Sostituire una risorsa 100 volte = stato finale identico
- **DELETE:** Eliminare la stessa risorsa 2 volte = stato finale identico (già eliminata)

Verbi non-idempotenti:

- **POST:** Ogni POST crea una nuova risorsa
- **PATCH:** Potrebbe non essere idempotente (dipende dall'implementazione)

Perché importa? Idempotenza permette al client di ritentare in sicurezza (safe retries).

---

18. Query Parameters vs Body

Domanda chiusa:
Per GET /api/users?role=admin&limit=10, i parametri sono:
A) Nel body della richiesta
B) Negli header
C) Negli query parameters (URL)
D) Nel path dell'URL

Risposta: C) Negli query parameters (URL).
`?role=admin&limit=10` sono query parameters. GET non dovrebbe avere body.

---

19. POST con parametri

Domanda aperta:
Quando fai `POST /api/users` per creare un utente, dove vanno i dati dell'utente (nome, email, password)?

Risposta suggerita:
Nel **body** della richiesta, solitamente in formato JSON:
POST /api/users
Content-Type: application/json

{
"name": "Marco Rossi",
"email": "marco@example.com",
"password": "securePassword123",
"role": "user"
}
I dati non vanno nei query parameters (quegli sono per filtri); vanno nel body per POST.

---

20. Status 201 vs 200 per creazione

Domanda chiusa:
Quando POST crea una nuova risorsa, il server restituisce:
A) 200 OK
B) 201 Created
C) 202 Accepted
D) 204 No Content

Risposta: B) 201 Created.
201 indica che una nuova risorsa è stata creata. Il response include solitamente l'URI della risorsa creata nell'header `Location`.

---

21. Metodi safe e idempotenti

Domanda chiusa:
Quale coppia di affermazioni è corretta?
A) POST è safe e idempotente
B) GET è safe e idempotente

C) PUT è safe e idempotente
D) DELETE è safe e non-idempotente

Risposta: B) GET è safe e idempotente.

- **Safe:** Non modifica lo stato del server (GET, HEAD, OPTIONS)
- **Idempotente:** Eseguire N volte = eseguire 1 volta (GET, PUT, DELETE, HEAD, OPTIONS)

---

22. Quando usare PATCH vs PUT?

Domanda aperta:
Descrivi uno scenario dove PATCH è preferibile a PUT e uno dove PUT è necessario.

Risposta suggerita:
PATCH è preferibile quando:
Utente aggiorna solo il suo profilo email via mobile con connessione lenta:
PATCH /api/users/123
{ "email": "newemail@example.com" } ← solo email, risparmia bandwidth

PUT è necessario quando:
Un form web richiede che tutti i campi obbligatori siano aggiornati insieme (es. cambio indirizzo completo):
PUT /api/users/123
{ "street": "Via Roma", "city": "Milano", "zip": "20100", "country": "IT" }

---

23. Trailing slash in URI

Domanda chiusa:
Secondo REST best practices, `/api/users` e `/api/users/` sono:
A) Identici
B) Diversi (pointing a risorse diverse)
C) Uno è errato
D) Dipende dal server

Risposta: B) Diversi (pointing a risorse diverse), anche se molti server li tratta come equivalenti.
Best practice: scegli una convenzione (con o senza trailing slash) e mantienila coerente.

---

24. API versioning nel path

Domanda chiusa:
Qual è il migliore approccio per versionare un'API?
A) `/users` (senza versione)
B) `/api/v1/users`, `/api/v2/users`
C) `/users?version=1`
D) Cambiare il domain `/api-v2.example.com/users`

Risposta: B) `/api/v1/users`, `/api/v2/users`.
Versioning nel path è standard di fatto. Permette coesistenza di versioni diverse.

---

25. Content-Type header

Domanda aperta:
Quando POST un JSON a un'API, quale header devi settare? Perché è importante?

Risposta suggerita:
Devi settare:
POST /api/users
Content-Type: application/json

Perché è importante?

- Il server sa che il body è JSON (non XML o form-data)
- Il server può parsearlo correttamente
- CORS preflight controlla questo header
- Senza Content-Type, il server potrebbe rifiutare o malinterpretare i dati

---

SEZIONE 3: STATUS CODES HTTP (Domande 26-35)

26. 2xx Success - 200 vs 201 vs 204

Domanda aperta:
Spiega le differenze tra 200 OK, 201 Created e 204 No Content.

Risposta suggerita:

- **200 OK:** Richiesta riuscita, response body contiene i dati. Usare per GET, PUT, POST (non creazione).
- **201 Created:** Nuova risorsa creata con successo. Usare per POST di creazione. Include Location: /resource/id header.
- **204 No Content:** Richiesta riuscita, NO body nella risposta. Usare per DELETE, PUT vuoto, PATCH.

Differenza chiave: 201 ha il corpo della risorsa creata; 204 no.

---

27. 3xx Redirect - 301 vs 307

Domanda chiusa:
Quale status code significa "la risorsa si è spostata PERMANENTEMENTE a un nuovo URI"?
A) 302 Found
B) 304 Not Modified
C) 301 Moved Permanently
D) 307 Temporary Redirect

Risposta: C) 301 Moved Permanently.
301 = permanente; 307 = temporaneo; 302 = redirect generico (ambiguo).

---

28. 304 Not Modified - Caching

Domanda chiusa:
304 Not Modified è usato quando:
A) La risorsa non esiste
B) Il client non ha permessi
C) La risorsa non è cambiata da ultimo check (caching)
D) Errore del server

Risposta: C) La risorsa non è cambiata da ultimo check (caching).
Se il client invia `If-Modified-Since` header e la risorsa non è stata modificata, server rispondi 304 (no body, no bandwidth).

---

29. 4xx Client Errors - 400, 401, 403, 404

Domanda aperta:
Distingui tra 400 Bad Request, 401 Unauthorized, 403 Forbidden, e 404 Not Found.

Risposta suggerita:

- **400 Bad Request:** La richiesta è malformata (JSON non valido, parametri mancanti, syntax error). Responsabilità: client.
- **401 Unauthorized:** Autenticazione richiesta o fallita (credenziali sbagliate, token scaduto). Responsabilità: client non autenticato.
- **403 Forbidden:** Client autenticato ma NON ha permessi per accedere (authorization fallita). Responsabilità: client non autorizzato.
- **404 Not Found:** La risorsa richiesta non esiste. Responsabilità: risorsa non esiste.

Regola mnemonica: 401 = "chi sei?"; 403 = "non puoi"; 404 = "non esiste"

---

30. 422 Unprocessable Entity

Domanda chiusa:
422 Unprocessable Entity significa:
A) Errore di grammatica nella richiesta
B) La richiesta è ben formata MA contiene errori semantici (es. email già esiste, validazione fallita)
C) Il server è down
D) Timeout

Risposta: B) La richiesta è ben formata MA contiene errori semantici (es. email già esiste).
Differenza da 400: la sintassi è corretta, ma il contenuto è invalido logicamente.

---

31. 429 Too Many Requests

Domanda chiusa:
429 Too Many Requests è inviato quando:
A) Il client fa troppe richieste in poco tempo (rate limiting)
B) Il database ha troppi record
C) Il server ha troppe risorse in uso
D) La memoria è piena

Risposta: A) Il client fa troppe richieste in poco tempo (rate limiting).
Protezione da abuse. Response include `Retry-After` header.

---

32. 5xx Server Errors - 500 vs 503

Domanda chiusa:
La differenza tra 500 Internal Server Error e 503 Service Unavailable è:
A) Non c'è differenza
B) 500 = errore generico del server; 503 = server temporaneamente non disponibile
C) 500 = lato client; 503 = lato server
D) 500 è deprecato

Risposta: B) 500 = errore generico del server; 503 = server temporaneamente non disponibile (manutenzione, overload).

---

33. Scegliere il status code corretto

Domanda aperta:
Un utente invia `POST /api/users` con email già registrata nel sistema. Quale status code dovresti inviare e perché?

Risposta suggerita:
Dovresti inviare **409 Conflict** (o 422 Unprocessable Entity).

- **409 Conflict:** La richiesta è in conflitto con lo stato attuale (email duplicata viola unicità).
- Alternativa **422:** La richiesta è valida sintatticamente ma semanticamente errata.
  Response example:
  409 Conflict
  {
  "error": "Email already registered",
  "message": "An account with this email already exists"
  }

---

34. Location header con 201

Domanda chiusa:
Quando POST crea una nuova risorsa con 201 Created, il server dovrebbe includere:
A) Content-Length header
B) Location header che indica l'URI della nuova risorsa
C) Authorization header
D) Nessuno di questi

Risposta: B) Location header.
Example:
201 Created
Location: /api/users/124
Content-Type: application/json

{ "id": 124, "name": "Marco", "email": "marco@example.com" }

---

35. HEAD vs GET status codes

Domanda chiusa:
HEAD /api/users/123 restituisce gli stessi status code di GET /api/users/123?
A) Sì, esattamente gli stessi
B) No, HEAD ha status code diversi
C) Dipende dal server
D) HEAD non usa status code

Risposta: A) Sì, esattamente gli stessi.
HEAD è identico a GET negli status code e header, solo il body è assente. Se GET torna 200, HEAD torna 200; se GET torna 404, HEAD torna 404.

SEZIONE 4: RICHARDSON MATURITY MODEL (Domande 36-45)

36. Livello 0 - RPC

Domanda aperta:
Descrivi come funziona un'API di Livello 0 del Richardson Model. Fai un esempio di tre operazioni (get, create, delete) per una risorsa "prodotto".

Risposta suggerita:
Livello 0 è essenzialmente RPC (Remote Procedure Call) wrapped in HTTP:

- Un solo endpoint `/api`
- Tutto usa POST
- Un parametro specifica l'azione da eseguire

Esempi:
POST /api
{ "action": "getProduct", "productId": 123 }
→ Restituisce i dati del prodotto

POST /api
{ "action": "createProduct", "name": "Laptop", "price": 999 }
→ Crea nuovo prodotto

POST /api
{ "action": "deleteProduct", "productId": 123 }
→ Elimina prodotto

Problema: Non sfrutta HTTP verbs, status code, caching, scaling. È inefficiente.

---

37. Livello 0 - Vantaggi e Svantaggi

Domanda chiusa:
Un vantaggio del Livello 0 è:
A) Usa HTTP in modo ottimale
B) Facile per chi conosce RPC

C) Semantica chiara dei metodi HTTP
D) Support per HATEOAS

Risposta: B) Facile per chi conosce RPC.
Ma non è un vero vantaggio REST, anzi: Livello 0 è il peggior design per un'API REST.

---

38. Livello 1 - Resources (Risorse)

Domanda aperta:
Spiega il Livello 1 del Richardson Model. Cosa aggiunge rispetto al Livello 0? Fai un esempio.

Risposta suggerita:
Livello 1 introduce le **risorse** con URI dedicate:

- Ogni risorsa ha un endpoint univoco
- Continua a usare POST per tutto (no distinzione tra GET, PUT, DELETE)

Esempio (vs Livello 0):
Livello 0:
POST /api { "action": "getProduct", "productId": 123 }

Livello 1:
POST /api/products/123 { "action": "get" }

POST /api/products { "action": "create", "name": "Laptop" }

POST /api/products/123 { "action": "delete" }

Miglioramento: Almeno organizza il codice per risorse, ma non sfrutta ancora i verbi HTTP.

---

39. Livello 2 - HTTP Verbs

Domanda chiusa:
Il Livello 2 del Richardson Model è caratterizzato da:
A) Uso appropriato dei verbi HTTP (GET, POST, PUT, DELETE)
B) HATEOAS in tutte le risposte
C) Un solo endpoint centralizzato
D) Nessuna autenticazione

Risposta: A) Uso appropriato dei verbi HTTP.
Livello 2 = **HTTP Verbs** + **Status Codes corretti** + **Resources** + **Caching**.

---

40. Livello 2 - Esempio pratico

Domanda aperta:
Convertisci questo endpoint di Livello 1 a Livello 2:
POST /api/products/123 { "action": "update", "name": "Nuovo nome", "price": 500 }

Risposta suggerita:
Livello 2 usa PUT o PATCH con i verbi HTTP corretti:
PUT /api/products/123
{ "name": "Nuovo nome", "price": 500 }

Oppure PATCH (aggiornamento parziale):
PATCH /api/products/123
{ "price": 500 }

Livello 2 inoltre usa status code appropriati:

- 200 OK con body se il server restituisce il prodotto aggiornato
- 204 No Content se niente viene restituito
- 400 Bad Request se dati invalidi
- 404 Not Found se il prodotto non esiste

---

41. Livello 3 - HATEOAS

Domanda chiusa:
HATEOAS (Hypermedia As The Engine Of Application State) significa:
A) Le risposte contengono link per navigare le risorse correlate
B) Usare XML al posto di JSON
C) Aumentare la complessità dell'API
D) Criptare tutti i dati

Risposta: A) Le risposte contengono link per navigare le risorse correlate.
HATEOAS = "Hypermedia As The Engine" - la risposta guida il client su cosa fare dopo.

---

42. HATEOAS - Esempio completo

Domanda aperta:
Mostra un'API di Livello 3 (HATEOAS) per GET /api/products/123. Includi link per update, delete, e torna alla lista.

Risposta suggerita:
GET /api/products/123

200 OK
{
"id": 123,
"name": "Laptop",
"price": 999,
"_links": {
"self": {
"href": "/api/products/123",
"method": "GET"
},
"update": {
"href": "/api/products/123",
"method": "PUT"
},
"patch": {
"href": "/api/products/123",
"method": "PATCH"
},
"delete": {
"href": "/api/products/123",

"method": "DELETE"
},
"all_products": {
"href": "/api/products",
"method": "GET"
}
}
}

Vantaggio: Il client non deve conoscere gli URI; li scopre dalla risposta (self-describing API).

---

43. Livello 3 - Vantaggi e Svantaggi

Domanda aperta:
Elenca 3 vantaggi e 2 svantaggi di HATEOAS (Livello 3).

Risposta suggerita:
Vantaggi:

1. **Decoupling:** Il client non deve conoscere gli URI; li scopre dalla risposta.
2. **Evoluzione API:** Se cambi gli URI, il client continua a funzionare (usa i link).
3. **Documentazione implicita:** I link disponibili mostrano cosa si può fare.

Svantaggi:

1. **Aumento dimensione response:** Ogni risposta contiene i link (overhead).
2. **Complessità:** Più difficile da implementare e debuggare.
3. **Non standardizzato:** Niente format standard per i link (HAL vs JSON-LD vs altro).

---

44. Quale livello è lo standard di fatto?

Domanda chiusa:
La maggior parte delle API REST moderne si ferma a quale livello?
A) Livello 1
B) Livello 2
C) Livello 3
D) Dipende

Risposta: B) Livello 2.
Livello 2 (HTTP Verbs + Status Codes) è il sweet spot tra semplicità e funzionalità. Livello 3 HATEOAS è raro nella pratica.

---

45. Migliorare da Livello 1 a Livello 2

Domanda aperta:
Un'API attualmente al Livello 1 invia sempre 200 OK per ogni richiesta. Come migliorare a Livello 2?

Risposta suggerita:
Per passare a Livello 2:

1. **Usare verbi HTTP appropriati:** GET per lettura, POST per creazione, PUT/PATCH per update, DELETE per eliminazione.
2. **Usare status code appropriati:**
    - 201 Created per POST che crea
    - 204 No Content per DELETE di successo
    - 400 Bad Request per input invalido
    - 404 Not Found per risorsa inesistente
    - 409 Conflict per duplicati
3. **Supportare caching:** Aggiungere header Cache-Control, ETag
4. **Documentazione:** Indicare quali verbi sono supportati per ogni endpoint

Example:
Prima (L1): POST /api/users/123 { "action": "delete" } → 200 OK
Dopo (L2): DELETE /api/users/123 → 204 No Content

SEZIONE 5: CORS - Same Origin Policy (Domande 46-60)

46. Same-Origin Policy - Definizione

Domanda aperta:
Definisci cos'è la Same-Origin Policy e spiega quali componenti di un URL definiscono un "origin".

Risposta suggerita:
Same-Origin Policy è una politica di sicurezza dei browser che impedisce a JavaScript di una pagina di accedere ai dati di un'altra origin senza autorizzazione esplicita.

Un origin è una combinazione di:

1. **Protocollo** (http, https)
2. **Dominio** (example.com, api.example.com)
3. **Porta** (80, 443, 3000, 8080)

Esempi di same origin:

- `https://example.com` e `https://example.com/page2` ✓ (same)
- `https://example.com` e `https://example.com:8080` ✗ (diverso: porta diversa)
- `https://example.com` e `http://example.com` ✗ (diverso: protocollo diverso)
- `https://example.com` e `https://api.example.com` ✗ (diverso: dominio diverso)

---

47. Perché esiste la Same-Origin Policy?

Domanda chiusa:
La Same-Origin Policy protegge principalmente da:
A) Virus
B) Attacchi cross-site (XSS, CSRF)
C) Perdita di password
D) Spamming

Risposta: B) Attacchi cross-site (XSS, CSRF).
Se un sito malevolo potesse leggere dati di Facebook, potrebbe rubare token di sessione, cookie, dati personali.

---

48. Quando la SOP blocca?

Domanda aperta:
Un'applicazione web in `https://app.example.com` chiama un'API su `https://api.other-domain.com/users`. La SOP blocca questa richiesta. Perché?

Risposta suggerita:
Perché il dominio è **diverso**:

- Origin richiesta: `https://app.example.com` (protocollo + dominio + porta)
- Origin API: `https://api.other-domain.com` (protocollo + dominio + porta)

Dominio diverso (`example.com` vs `other-domain.com`) = origin diverso = SOP blocca.

Come risolvere? Usare CORS sul server.

---

49. CORS - Definizione e scopo

Domanda chiusa:
CORS (Cross-Origin Resource Sharing) permette:
A) Di disabilitare completamente la sicurezza
B) Al server di indicare quali origin esterni possono accedere alle sue risorse
C) Di usare qualsiasi dominio senza restrizioni
D) Di evitare il login

Risposta: B) Al server di indicare quali origin esterni possono accedere alle sue risorse.
CORS = meccanismo di opt-in per permettere richieste cross-origin controllate.

---

50. Simple Request vs Preflight

Domanda aperta:
Spiega la differenza tra "simple request" e "preflight request" nel contesto CORS. Quando ciascuna è usata?

Risposta suggerita:
Simple Request: Il browser invia direttamente la richiesta (niente preflight):

- Verbi: GET, HEAD, POST
- Content-Type semplice: application/x-www-form-urlencoded, multipart/form-data, text/plain
- No custom header
- No Authorization header

Esempio:
GET https://api.other-domain.com/users
Accept: application/json
→ Browser invia direttamente, nessun preflight

Preflight Request: Il browser invia prima OPTIONS (richiesta preflight):

- Verbi: PUT, DELETE, PATCH
- Content-Type: application/json
- Custom header: Authorization, X-Custom-Header

Flusso:

1. Browser invia OPTIONS /api/users
   (chiede: posso fare POST con Authorization?)
2. Server risponde con CORS headers
   (dice: sì, puoi)
3. Browser invia richiesta vera: POST /api/users
   (con Authorization header)

---

51. Preflight REQUEST headers

Domanda chiusa:
In una richiesta OPTIONS preflight, quale header specifica il metodo HTTP che il client intende usare?
A) Access-Control-Allow-Methods
B) Access-Control-Request-Method
C) Access-Control-Request-Headers
D) Allow

Risposta: B) Access-Control-Request-Method.
Esempio:
OPTIONS /api/users
Origin: https://example.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Authorization, Content-Type

---

52. Preflight RESPONSE headers

Domanda aperta:
Quale header CORS deve il server inviare nella risposta al preflight per autorizzare una richiesta POST con Authorization header?

Risposta suggerita:
200 OK
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, PATCH, OPTIONS
Access-Control-Allow-Headers: Authorization, Content-Type
Access-Control-Max-Age: 86400
Access-Control-Allow-Credentials: true

Spiegazione:

- Access-Control-Allow-Origin: Quale origin è autorizzato
- Access-Control-Allow-Methods: Quali metodi HTTP sono permessi
- Access-Control-Allow-Headers: Quali header personalizzati sono permessi
- Access-Control-Max-Age: Cache il preflight per X secondi
- Access-Control-Allow-Credentials: Se i cookie sono permessi

---

53. Access-Control-Allow-Origin

Domanda chiusa:
Quale valore in Access-Control-Allow-Origin è il più permissivo?
A) Access-Control-Allow-Origin: https://example.com
B) Access-Control-Allow-Origin: *
C) Access-Control-Allow-Origin: localhost
D) Non settare l'header

Risposta: B) Access-Control-Allow-Origin: *
`*` significa: qualsiasi origin può accedere. **Ma è meno sicuro**: non usare per API sensibili.

---

54. Wildcard vs Specific Origin

Domanda aperta:
Confronta `Access-Control-Allow-Origin: *` vs `Access-Control-Allow-Origin: https://example.com`. Quando usare ciascuno?

Risposta suggerita:
`Access-Control-Allow-Origin: *`:

- Permette a QUALSIASI origin di accedere
- **Quando usare:** API pubbliche, dati non sensibili (weather API, news API)
- **Quando NON usare:** API private, dati sensibili, backend
- **Sicurezza:** Bassa (ma ok per public data)

`Access-Control-Allow-Origin: https://example.com`:

- Permette SOLO a `https://example.com` di accedere
- **Quando usare:** API private, siti specifici autorizzati
- **Quando NON usare:** API pubblica aperta a tutti
- **Sicurezza:** Alta (whitelist) - **RECOMMENDED**

---

55. Access-Control-Allow-Credentials

Domanda chiusa:
Se il server setta `Access-Control-Allow-Credentials: true`, cosa significa?
A) Accesso senza password
B) I cookie e credenziali HTTP sono permessi nella richiesta cross-origin
C) Niente login richiesto
D) Accesso gratuito

Risposta: B) I cookie e credenziali HTTP sono permessi nella richiesta cross-origin.
Necessario se il client invia `credentials: 'include'` nel fetch.

---

56. CORS Error - Debugging

Domanda aperta:
Uno sviluppatore riceve questo errore nel browser:
Access to XMLHttpRequest at 'https://api.example.com/users' from origin 'https://app.example.com' has been blocked by CORS policy: No 'Access-Control-Allow-Origin'

header is present on the requested resource.
Cosa significa? Quali sono 3 possibili soluzioni?

Risposta suggerita:
Significa: Il browser blocca la richiesta perché il server non ha inviato l'header CORS appropriato.

3 Soluzioni:

1. **Il server aggiunge CORS:** Implementare CORS sul server e inviare `Access-Control-Allow-Origin: https://app.example.com`
2. **Usare un proxy:** Passare la richiesta attraverso un proxy che aggiunge i CORS header
3. **Backend-to-backend:** Fare la richiesta dal backend (Node.js/Python) invece che dal browser (backend non ha restrizione SOP)

---

57. CORS in Node.js Express

Domanda aperta:
Scrivi il codice per abilitare CORS su un server Express per permettere a `https://app.example.com` di accedere.

Risposta suggerita:

```
const express = require('express');
const cors = require('cors');
const app = express();

// CORS per tutte le rotte
const corsOptions = {
origin: 'https://app.example.com', // solo questo origin
methods: ['GET', 'POST', 'PUT', 'DELETE', 'PATCH'],
allowedHeaders: ['Content-Type', 'Authorization'],
credentials: true, // permetti cookie
maxAge: 3600 // cache preflight per 1 ora
};

app.use(cors(corsOptions));

app.get('/api/users', (req, res) => {
res.json([...]);
});

Oppure manualmente:
app.use((req, res, next) => {
res.setHeader('Access-Control-Allow-Origin', 'https://app.example.com');
res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');

if (req.method === 'OPTIONS') {
return res.sendStatus(200); // preflight
}
next();
});
```

---

58. CORS in Client JavaScript

Domanda aperta:
Scrivi il fetch che trigga un preflight CORS (non simple request).

Risposta suggerita:
```
// Questo trigga preflight (non simple request)
fetch('https://api.other-domain.com/users', {
method: 'POST', // Non simple
headers: {
'Content-Type': 'application/json', // Non simple
'Authorization': 'Bearer token123' // Custom header
},
credentials: 'include', // Includi cookie
body: JSON.stringify({
name: 'Marco',
email: 'marco@example.com'
})
})
.then(res => res.json())
.then(data => console.log(data))
.catch(err => console.error('CORS error:', err));
```

Questo trigga preflight perché:

- Metodo POST (non simple)
- Content-Type: application/json (non simple)
- Authorization header (custom)
- credentials: 'include' (cookie)

---

59. CORS Max-Age

Domanda chiusa:
`Access-Control-Max-Age: 3600` significa:
A) Il preflight scade dopo 3600 secondi
B) La richiesta timeout dopo 3600 secondi
C) Il token scade dopo 3600 secondi
D) Il cookie scade dopo 3600 secondi

Risposta: A) Il preflight scade dopo 3600 secondi.
Il browser cache il preflight per 3600 secondi; le richieste successive non lo reinviano.

---

60. CORS Expose Headers

Domanda chiusa:
Se il server aggiunge `Access-Control-Expose-Headers: X-Total-Count`, cosa significa?
A) Questo header viene nascosto
B) Il client JavaScript può leggere questo header nella risposta
C) Il server nasconde dati
D) Solo GET può usare questo header

Risposta: B) Il client JavaScript può leggere questo header nella risposta.
Per impostazione predefinita, JavaScript non può leggere custom response header. `Expose-Headers` li rende leggibili.

SEZIONE 6: CLOUD COMPUTING FOUNDATION (Domande 61-75)

61. Cloud Computing - Definizione NIST

Domanda aperta:
Fornisci la definizione ufficiale di Cloud Computing secondo NIST. Includi i componenti chiave.

Risposta suggerita:
NIST Definition:
Cloud Computing è un modello che consente accesso conveniente e **on-demand** tramite rete a un pool **condiviso** di risorse informatiche configurabili (reti, server, storage, applicazioni, servizi) che possono essere **rapidamente fornite e rilasciate** con **minimo sforzo di gestione** o interazione con il provider.

Componenti chiave:

- On-demand self-service
- Network access
- Resource pooling (multi-tenant)
- Rapid elasticity
- Measured service (pay-as-you-go)

---

62. Caratteristica: On-Demand Self-Service

Domanda chiusa:
On-Demand Self-Service significa:
A) Solo il provider può aggiungere risorse
B) L'utente acquisisce risorse autonomamente senza intervento umano del provider
C) Servizio gratuito al primo mese
D) Accesso solo durante orari di ufficio

Risposta: B) L'utente acquisisce risorse autonomamente senza intervento umano del provider.
Clicchi un bottone → VM viene provisionata in minuti senza ticket support.

---

63. Caratteristica: Broad Network Access

Domanda chiusa:
Broad Network Access in cloud significa:
A) Accesso solo da rete locale
B) Servizi accessibili via Internet da diversi dispositivi
C) Rete illimitata senza restrizioni
D) Accesso solo via VPN

Risposta: B) Servizi accessibili via Internet da diversi dispositivi.
Puoi usare l'app cloud da smartphone, tablet, laptop, browser - ovunque ci sia Internet.

---

64. Caratteristica: Resource Pooling

Domanda chiusa:
Resource Pooling (multi-tenancy) nel cloud significa:
A) Le risorse sono dedicate a una sola azienda
B) Le risorse sono condivise tra molteplici clienti, ma isolate logicamente
C) Tutti i clienti condividono i dati
D) Niente isolamento

Risposta: B) Le risorse sono condivise tra molteplici clienti, ma isolate logicamente.
Un solo server fisico serve 100 VM virtuali di 100 clienti diversi, isolate logicamente ma condivisi fisicamente (economia).

---

65. Caratteristica: Rapid Elasticity

Domanda aperta:
Spiega cosa significa "Rapid Elasticity" nel cloud con un esempio pratico di un e-commerce nel Black Friday.

Risposta suggerita:
Rapid Elasticity: Le risorse possono essere aumentate o diminuite rapidamente in base alla domanda.

Esempio Black Friday (e-commerce):

- **Novembre (normale):** 100 server in uso
- **24 novembre (Black Friday):** Auto-scaling attiva, 1000 server in uso (alta richieste)
- **25 novembre (dopo):** Auto-scaling riduce a 100 server (no più necessari)
- **Paghi:** Solo i server usati (elasticità di prezzo)

Senza cloud: Dovresti comprare 1000 server per Black Friday, usarli solo 1 giorno all'anno (waste).

---

66. Caratteristica: Measured Service

Domanda chiusa:
Measured Service significa:
A) Pagamento fisso annuale
B) L'uso di risorse è misurato e fatturato (pay-as-you-go)
C) Servizio gratuito
D) Niente tracciamento

Risposta: B) L'uso di risorse è misurato e fatturato (pay-as-you-go).
AWS CloudWatch monitora: CPU, memoria, disco, bandwidth usati → fattura mensile basata su consumo reale.

---

67. Cloud Deployment - Public Cloud

Domanda aperta:
Definisci Public Cloud e elenca almeno 3 caratteristiche e 3 use case.

Risposta suggerita:
Definizione: Infrastruttura cloud posseduta/gestita da provider, disponibile al pubblico via Internet.

Caratteristiche:

1. Infrastruttura condivisa tra molte organizzazioni
2. Massima scalabilità e flessibilità
3. Costi inferiori (condivisione risorse)
4. Minore controllo sulla sicurezza vs Private

Use case:

1. **Startup:** Costi iniziali bassi, scalabilità rapida (es. SaaS)
2. **Applicazioni web pubbliche:** Blog, e-commerce, piattaforme social
3. **Dev/Test:** Ambienti isolati per testing, prototipazione rapida

Esempi provider: AWS, Azure, Google Cloud Platform

---

68. Cloud Deployment - Private Cloud

Domanda aperta:
Definisci Private Cloud. Quando dovresti scegliere Private Cloud instead di Public Cloud?

Risposta suggerita:
Definizione: Infrastruttura cloud dedicata SOLO a un'organizzazione, non condivisa con altri. Può essere on-premise o gestita da provider terzo.

Quando scegliere Private Cloud:

1. **Sicurezza elevata:** Dati sensibili (finanza, sanità, governi)
2. **Compliance normativa:** GDPR, HIPAA, PCI-DSS richiedono controllo totale
3. **Performance critica:** Bassa latenza, niente multi-tenant overhead
4. **Dati on-premise:** Vincoli di residenza dati

Svantaggi: Capex iniziale elevato, complessità gestione.

---

69. Cloud Deployment - Hybrid Cloud

Domanda chiusa:
Hybrid Cloud è:
A) Combinazione di Public e Private Cloud con orchestrazione tra loro
B) Due provider public diversi
C) Niente internet
D) Accesso localizzato

Risposta: A) Combinazione di Public e Private Cloud con orchestrazione tra loro.
Esempio: Dati sensibili su Private Cloud, frontend scalabile su Public Cloud.

---

70. Cloud Deployment - Multi-Cloud

Domanda aperta:
Qual è la differenza tra Hybrid Cloud e Multi-Cloud? Quando dovresti usare Multi-Cloud?

Risposta suggerita:
Differenza:

- **Hybrid:** Public + Private cloud del STESSO provider (es. AWS public + AWS private)
- **Multi-Cloud:** Più provider PUBLIC diversi (es. AWS + Azure + GCP)

Quando usare Multi-Cloud:

1. **Evitare vendor lock-in:** Se AWS diventa costoso, puoi spostare su Azure
2. **Best-of-breed services:** AWS per compute, Google per BigQuery (analytics), Azure per enterprise
3. **Redundancy geografica:** Ridondanza cross-provider
4. **Competizione prezzo:** Gioca i provider l'uno contro l'altro

Sfida: Complessità significativa di gestione.

---

71. Modello IaaS - Responsabilità

Domanda aperta:
In IaaS, cosa gestisce il provider e cosa gestisce l'utente? Fai una lista completa.

Risposta suggerita:
Provider gestisce:

- Infrastruttura fisica (server, storage)
- Virtualizzazione
- Rete (router, switch)
- Datacenter (cooling, potenza)

Utente gestisce:

- Sistema Operativo
- Middleware/Runtime
- Applicazioni
- Dati
- Security (firewall, patching OS)
- Backup
- Disaster recovery

Esempio: AWS EC2

- AWS: Virtual machine hardware
- Tu: OS (Windows/Linux), installed software, patching

---

72. Modello PaaS - Quando usare

Domanda chiusa:
PaaS è ideale quando:
A) Hai bisogno di controllo totale dell'OS
B) Vuoi sviluppare rapidamente senza gestire infrastruttura
C) Il costo è prioritario assoluto
D) Usi solo containersnon

Risposta: B) Vuoi sviluppare rapidamente senza gestire infrastruttura.
PaaS = sviluppatore scrive codice, platform gestisce il resto.

---

73. Modello SaaS - Caratteristiche

Domanda aperta:
Spiega SaaS e fornisci 5 esempi. Quali sono i vantaggi e 2 svantaggi?

Risposta suggerita:
SaaS (Software as a Service): Applicazione completa fornita dal provider, accessibile via browser.

Esempi:

1. Google Workspace (email, docs, sheets)
2. Salesforce (CRM)
3. Slack (comunicazione)
4. Office 365 (productivity)
5. Zoom (videoconferenza)

Vantaggi:

- Zero gestione infrastruttura
- Aggiornamenti automatici
- Accesso ovunque (browser-based)
- Collaborazione built-in
- Costi prevedibili (subscription)

Svantaggi:

- Vendor lock-in significativo (hard to migrate)
- Personalizzazione limitata (one-size-fits-all)
- Dipendenza internet

---

74. IaaS vs PaaS vs SaaS - Tabella Comparativa

Domanda aperta:
Crea una tabella comparativa tra IaaS, PaaS e SaaS che mostri: cosa gestisce il provider, cosa gestisce l'utente, complessità, controllo, e un esempio.

Risposta suggerita:
ASPETTO IaaS PaaS SaaS
_____

Gestisce Provider:

- Infrastruttura ✓ ✓ ✓
- OS X ✓ ✓
- Middleware X ✓ ✓
- Applicazione X X ✓
- Dati X X ✓

Gestisce Utente:

- Tutto dal SO in su

Complessità ALTA MEDIA BASSA
Controllo ALTO MEDIO BASSO
Costi VARIABILI BASSI BASSI
Flessibilità MASSIMA MEDIA MINIMA
Vendor Lock-in BASSO MEDIO ALTO

Esempio AWS EC2 Heroku Salesforce

---

75. TCO - Cloud vs On-Premise

Domanda aperta:
Spiega la differenza tra CapEx (Capital Expenditure) e OpEx (Operational Expenditure) nel contesto cloud vs on-premise.

Risposta suggerita:
On-Premise (CapEx - Capital Expenditure):

- Upfront cost: Compri server fisici ($100,000)
- Ongoing cost: Manutenzione, electricity, cooling
- ROI lento (investimento ammortizzato in anni)
- Sottoutilizzato nella maggior parte dei giorni

Cloud (OpEx - Operational Expenditure):

- Upfront cost: Vicino a 0
- Ongoing cost: Pay-as-you-go ($1000/mese per risorse usate)
- ROI veloce (puoi scalare subito)
- Paga SOLO quello che usi

Vantaggi cloud: Flessibilità finanziaria, niente investimento iniziale.

SEZIONE 7: CLOUD STORAGE & DATABASE (Domande 76-85)

76. Block Storage - Definizione e use case

Domanda aperta:
Cosa è Block Storage? Quando lo usi? Quali sono provider e limiti?

Risposta suggerita:
Block Storage: Storage organizzato in blocchi di dimensione fissa, come dischi rigidi tradizionali. Accesso diretto a indirizzi di blocco.

Quando usare:

- Database (MySQL, PostgreSQL) - transazioni high-speed
- Sistema operativo (OS root volume)
- Applicazioni con latenza critica

Provider:

- AWS: EBS (Elastic Block Store)
- Azure: Managed Disks
- GCP: Persistent Disks

Caratteristiche:

- Performance altissima
- Bassa latenza
- Costoso (~$0.10-$0.15/GB/mese)
- Max storage dipende dal tipo (GiB-TiB)

---

77. File Storage - NFS, SMB

Domanda chiusa:
File Storage nel cloud è ottimale per:
A) Singola VM che legge/scrive dati
B) Accesso condiviso da multiple VM tramite NFS/SMB
C) Oggetti immutabili non strutturati
D) Backup annuali

Risposta: B) Accesso condiviso da multiple VM tramite NFS/SMB.
File storage è come un'unità network share (NAS) accessibile da multiple macchine.

---

78. Object Storage - S3, Blob Storage

Domanda aperta:
Descrivi Object Storage. Quali sono i vantaggi rispetto a Block Storage? Fai 3 use case.

Risposta suggerita:
Object Storage: Storage per oggetti (file) con metadata. Accesso via HTTP/REST API. No struttura gerarchica tradizionale.

Vantaggi vs Block:

1. **Costo:** 100x più economico (~$0.023/GB/mese vs $0.10)
2. **Scalabilità:** Infinita (exabyte scale)
3. **Durabilità:** 11 nines (99.999999999%)
4. **Accesso globale:** HTTP/REST da qualsiasi luogo

Use case:

1. **Backup/Archivio:** Dati storici rarely accessed (S3 Glacier)
2. **Hosting media:** Immagini, video per CDN (S3 + CloudFront)
3. **Data Lakes:** Big data storage (S3 → Spark, Athena)

Provider: AWS S3, Azure Blob Storage, GCP Cloud Storage

---

79. S3 Pricing - Standard vs Glacier

Domanda chiusa:
S3 Glacier è ottimale per:
A) Accesso frequente ai dati
B) Archivio long-term con accesso raro e costo basso
C) Real-time analytics
D) Database transazionale

Risposta: B) Archivio long-term con accesso raro e costo basso.
Glacier: storage ultra-cheap, ma retrieval lento (ore).

---

80. RDS - Managed Relational Database

Domanda aperta:
Cosa è RDS (Relational Database Service) in AWS? Quali DB supporta? Quali sono i vantaggi rispetto a database on EC2?

Risposta suggerita:
RDS: Servizio managed che fornisce database relazionali hostati (no gestione infrastruttura).

DB Supportati:

- MySQL / MariaDB
- PostgreSQL
- Oracle
- SQL Server
- Aurora (cloud-native)

Vantaggi vs EC2:

1. **No infrastruttura:** AWS gestisce OS, patching, backup
2. **High availability:** Multi-AZ failover automatico
3. **Backup automatici:** Point-in-time recovery
4. **Replica per lettura:** Scalabilità read
5. **Monitoring:** CloudWatch metriche built-in

---

81. DynamoDB - NoSQL Managed

Domanda chiusa:
DynamoDB è ideale per:
A) SQL complex joins
B) Document storage con schema flessibile e bassa latenza
C) ACID transazioni tradizionali
D) Relazioni complesse

Risposta: B) Document storage con schema flessibile e bassa latenza.
DynamoDB: NoSQL key-value/document database, massicciamente scalabile.

---

82. BigQuery - Data Warehousing

Domanda aperta:
Cos'è BigQuery? Per quale tipo di workload è ottimale? Quali sono i vantaggi?

Risposta suggerita:
BigQuery: Data warehouse managed in cloud per analytics su enormi dataset.

Workload: Analytics, BI (Business Intelligence), reporting su dati storici (non real-time transazioni).

Vantaggi:

1. **Scalabilità massiva:** Query su terabyte/petabyte in secondi
2. **Serverless:** No cluster da gestire
3. **Prezzo:** Pay-per-query (no server fees)
4. **Columnar storage:** Optimized per analytics queries
5. **Built-in ML:** BigQuery ML per modelli semplici

Esempio: SELECT COUNT(*) FROM table WHERE region='Italy' -- query su 1TB in 2 sec

---

83. Database Relazionale vs NoSQL

Domanda aperta:
Quando scegliere un database relazionale (SQL) vs NoSQL? Fai 3 esempi per ciascuno.

Risposta suggerita:
Relazionale (SQL):

- Transazioni ACID critiche
- Dati strutturati con relazioni complesse
- Compliance/Audit

Esempi: E-commerce (ordini, inventory con transazioni), Banca (transfer di denaro), ERP (purchase orders)

NoSQL:

- Denormalizzazione accettabile
- Schema flessibile
- Scalabilità massiva

Esempi: Social network (post/like), User profiles (flessibili), Sensor data (milioni eventi/sec)

---

84. Database Backup Strategy

Domanda aperta:
Descrivi una completa backup strategy per un database mission-critical in cloud. Includi: frequency, retention, testing.

Risposta suggerita:
Backup Strategy:

1. **Frequency:** Daily automated backups

- RDS: Automated snapshots (daily, 35-day retention)
- Incremental dopo snapshot completo

2. **Retention:**
   - 7 days: Fast retrieve
   - 30 days: Standard storage
   - 365 days: Archive (Glacier)
3. **Geo-Redundancy:**
   - Replicate backups to different region
   - Protect da regional outages
4. **Testing:**
   - Monthly restore test su dev environment
   - Verifica RTO (Recovery Time Objective): <1 hour
   - Verifica RPO (Recovery Point Objective): <15 min data loss acceptable
5. **Monitoring:**
   - Alert se backup fails
   - CloudWatch metrics

---

85. Database Migration to Cloud

Domanda aperta:
Descrivi il processo per migrare un database on-premise a un cloud database (es. on-premise MySQL → AWS RDS MySQL).

Risposta suggerita:
Processo Migration:

1. **Planning:**
   - Analyzeurrent database (size, traffic, dependencies)
   - Scegli RDS instance type (compute, storage)
   - Define RTO/RPO
2. **Pre-migration:**
   - Create RDS instance di destinazione
   - Setup security groups, VPC
   - Network connectivity (VPN, Direct Connect)
3. **Migration:**
   - **Option A - Full dump:** mysqldump → restore (downtime)
   - **Option B - Replication:** AWS Database Migration Service (DMS) con minimal downtime
   - **Option C - Hybrid:** Live replication, cutover quando syncronizzato
4. **Validation:**
   - Verify data integrity
   - Performance testing
   - Failover testing
5. **Cutover:**
   - Point app connessione string to RDS
   - Monitor per errori
6. **Post-migration:**
   - Decommission on-premise database
   - Optimize RDS (indexes, parameters)

SEZIONE 8: CLOUD SECURITY & MONITORING (Domande 86-90)

86. IAM - Identity and Access Management

Domanda aperta:
Spiega IAM (Identity and Access Management). Quali sono i tre componenti chiave? Fai un esempio AWS.

Risposta suggerita:
IAM: Sistema per controllare chi ha accesso a quale risorsa e quali azioni può eseguire.

3 Componenti:

1. **Authentication:** Chi sei? (Username/password, 2FA, OAuth)
2. **Authorization:** Cosa puoi fare? (Role-based access control - RBAC)
3. **Audit:** Chi ha fatto cosa? (CloudTrail logging)

Esempio AWS:
User: marco@company.com
Role: Developer
Permissions:

- s3:GetObject on s3://my-bucket/*
- rds:DescribeDBInstances
- (NOT) ec2:TerminateInstances (no permission)
  Marco può leggere S3, visualizzare RDS, ma NON terminare EC2.

---

87. Encryption at Rest vs in Transit

Domanda chiusa:
Encryption in transit significa:
A) Dati criptati quando immagazzinati
B) Dati criptati quando trasmessi via rete
C) Criptare il backup
D) Niente criptazione

Risposta: B) Dati criptati quando trasmessi via rete.

- **At rest:** S3 SSE, RDS encryption
- **In transit:** HTTPS/TLS per API, VPN per private network

---

88. DDoS Protection

Domanda aperta:
Cos'è un attacco DDoS? Come il cloud provider protegge? Quali sono le mitigazioni?

Risposta suggerita:
DDoS (Distributed Denial of Service): Attacco che bombarda il server con milioni di richieste, rendendolo inaccessibile.

Protezioni del provider:

- AWS Shield: Protezione base automatica
- AWS WAF (Web Application Firewall): Regole di filtraggio
- CloudFront CDN: DDoS mitigation via edge

Mitigazioni:

1. **Rate limiting:** Max richieste per IP
2. **IP whitelist/blacklist:** Blocca indirizzi sospetti
3. **WAF rules:** Filtra pattern malevoli
4. **Auto-scaling:** Scale up per assorbire attacco

---

89. CloudWatch Monitoring

Domanda aperta:
Spiega il monitoraggio in AWS con CloudWatch. Quali metriche è importante monitore? Come settare alerting?

Risposta suggerita:
CloudWatch: Servizio di monitoring, logging, e alerting in AWS.

Metriche critiche:

- **CPU Utilization:** If > 80% per 5 min → problem
- **Memory Usage:** If > 85% → risk di crash
- **Disk Space:** If < 10% libero → riempimento
- **Request Count:** If 0 → servizio down
- **Error Rate:** If > 5% → buggy deployment
- **API Latency:** If > 500ms → performance issue

Alerting:
IF cpu_utilization > 80% FOR 5 minutes
THEN send_email(ops-team@company.com) + auto_scale_up()

---

90. Disaster Recovery Strategy

Domanda aperta:
Descrivi una completa Disaster Recovery (DR) strategy per un'applicazione mission-critical in cloud. Includi: RTO, RPO, failover, testing.

Risposta suggerita:
DR Strategy:

1. **Objectives:**
   - **RTO (Recovery Time Objective):** Max 4 hours downtime
   - **RPO (Recovery Point Objective):** Max 1 hour data loss
2. **Architecture:**
   - **Primary Region:** us-east-1 (production)
   - **DR Region:** us-west-1 (standby)
   - **Replication:** RDS cross-region replica, S3 cross-region replication
   - **DNS Failover:** Route 53 health checks → automatic failover
3. **Backup:**
   - Database daily snapshots → DR region

    - S3 cross-region replication (real-time)
    - EBS snapshots to DR region
4. **Failover Process:**
    - Route 53 detects primary region unhealthy
    - Automatic redirect traffic to DR region
    - DR database promoted (read replica → primary)
    - Point applications to DR endpoints
5. **Testing:**
    - Monthly failover drills
    - Simulate primary region failure
    - Verify RTO/RPO met
    - Update runbook
6. **Monitoring:**
    - CloudWatch: Cross-region latency
    - CloudTrail: Audit log
    - SNS: Alerts for failures
7. **Failback:**
    - Repair primary region
    - Sync data back da DR
    - Gradual traffic shift back

CONCLUSIONE

Hai completato i 90 domande! Ecco un riepilogo per la preparazione finale:

Domande per argomento:

- API Fondamenti: 10 domande
- Verbi HTTP: 15 domande
- Status Codes: 10 domande
- Richardson Model: 10 domande
- CORS: 15 domande
- Cloud Foundation: 15 domande
- Cloud Storage/Database: 10 domande
- Cloud Security: 5 domande

Consigli di studio:

1. Leggi ogni domanda PRIMA di guardare la risposta
2. Misura il tuo tempo (70 minuti per 90 domande)
3. Rivedi le domande che hai sbagliato
4. Pratica il vocabolario: "idempotente", "multi-tenant", "elasticità"
5. Crea flashcard per status codes e verbi HTTP

Simulazione d'esame finale: Prendi 10 domande random e rispondi senza guardare le soluzioni. Se rispondi correttamente a 8/10, sei pronto per l'esame!

Buona fortuna!