Reliability

সিস্টেম ডিজাইন বা সফটওয়্যার ইঞ্জিনিয়ারিংয়ে **Reliability (নির্ভরযোগ্যতা)** মানে হলো—একটি সিস্টেম যেন প্রতিকূল পরিস্থিতিতেও (যেমন নেটওয়ার্ক ফেইলুর বা সার্ভার স্লো হওয়া) সঠিকভাবে কাজ চালিয়ে যেতে পারে।

সিস্টেমকে নির্ভরযোগ্য করার জন্য তিনটি অত্যন্ত গুরুত্বপূর্ণ মেকানিজম বা পদ্ধতি ব্যবহার করা হয়: **Retry, Timeout,** এবং **Circuit Breaker**। নিচে এগুলো সহজভাবে বুঝিয়ে বলা হলো:

১. Retry (রিট্রাই)

সহজ কথায়, কোনো কাজ একবার করতে গিয়ে ব্যর্থ হলে সেটি আবার চেষ্টা করা। অনেক সময় নেটওয়ার্কের ছোটখাটো সমস্যার কারণে কোনো রিকোয়েস্ট ফেইল হতে পারে, যা কয়েক মিলিসেকেন্ড পরেই আবার ঠিক হয়ে যায়।

- **কিভাবে কাজ করে:** যদি আপনার অ্যাপ কোনো API থেকে ডাটা আনতে গিয়ে এরর পায়, তবে সে হাল না ছেড়ে দিয়ে আরও ২-৩ বার চেষ্টা করে।
- **সতর্কতা (Exponential Backoff):** বারবার সাথে সাথে চেষ্টা না করে একটু বিরতি দিয়ে দিয়ে চেষ্টা করা উচিত (যেমন: প্রথমবার ১ সেকেন্ড পর, দ্বিতীয়বার ২ সেকেন্ড পর)। নাহলে ফেইল করা সার্ভারের ওপর চাপের পাহাড় জমে যাবে।

২. Timeout (টাইমআউট)

একটি রিকোয়েস্টের উত্তরের জন্য আপনি সর্বোচ্চ কতক্ষণ অপেক্ষা করবেন, তার সময়সীমা হলো টাইমআউট।

- **কেন প্রয়োজন:** মনে করুন আপনি একটি সার্ভারকে রিকোয়েস্ট পাঠালেন কিন্তু সেটি খুবই স্লো বা হ্যাং হয়ে আছে। আপনি যদি অনির্দিষ্টকাল অপেক্ষা করেন, তবে আপনার নিজের সিস্টেমের মেমোরি এবং রিসোর্স আটকে থাকবে।

- **কাজ:** আপনি যদি ৫ সেকেন্ড টাইমআউট সেট করেন, তবে সার্ভার উত্তর না দিলেও ৫ সেকেন্ড পর কানেকশনটি নিজে থেকেই কেটে যাবে। এতে আপনার সিস্টেম হ্যাং হওয়া থেকে বেঁচে যাবে।

৩. Circuit Breaker (সার্কিট ব্রেকার)

এটি সবচেয়ে বুদ্ধিমান পদ্ধতি। আমাদের বাসার ইলেকট্রিক সার্কিট ব্রেকার যেমন ভোল্টেজ বেশি হলে বিদ্যুৎ সংযোগ বিচ্ছিন্ন করে দেয়, এটিও ঠিক তেমন।

- **সমস্যা:** যখন কোনো সার্ভার বারবার ফেইল করতে থাকে, তখন বারবার **Retry** করলে বা **Timeout** পর্যন্ত অপেক্ষা করলে ওই সার্ভারটি আরও বেশি ক্ষতিগ্রস্ত হয় এবং পুরো সিস্টেম স্লো হয়ে যায়।
- **সমাধান:** যদি একটি নির্দিষ্ট সময়ের মধ্যে অনেকগুলো রিকোয়েস্ট ফেইল করে (যেমন: টানা ১০ বার), তবে সার্কিট ব্রেকার **"Open"** হয়ে যায়। এর মানে হলো, সিস্টেম আর ওই সার্ভারকে রিকোয়েস্ট পাঠাবেই না, বরং সাথে সাথে একটি এরর মেসেজ দিয়ে দিবে।
- **স্টেটসমূহ (States):**
    1. **Closed:** সবকিছু ঠিক আছে, রিকোয়েস্ট যাচ্ছে।
    2. **Open:** ফেইলুর রেট বেশি, তাই রিকোয়েস্ট পাঠানো বন্ধ।
    3. **Half-Open:** কিছুক্ষণ পর সিস্টেম চেক করে দেখে সার্ভারটি ঠিক হয়েছে কি না। ঠিক হলে আবার **Closed** স্টেটে ফিরে যায়।

৪. একটি বাস্তব উদাহরণ (ই-কমার্স সাইট)

কল্পনা করুন আপনি একটি শপিং অ্যাপ চালাচ্ছেন যেখানে পেমেন্ট করার জন্য থার্ড-পার্টি একটি গেটওয়ে (যেমন: bKash বা Stripe) ব্যবহার করছেন।

1. **Retry:** পেমেন্ট রিকোয়েস্ট পাঠালেন, নেটওয়ার্কের কারণে ফেইল করল। অ্যাপ নিজে থেকেই আরও একবার চেষ্টা করল এবং পেমেন্ট সফল হলো।

2. **Timeout:** পেমেন্ট গেটওয়ে খুব স্লো। আপনার অ্যাপ ১০ সেকেন্ড অপেক্ষা করে কানেকশন কেটে দিল যাতে কাস্টমারের স্ক্রিন চিরকাল লোডিং না দেখায়।
3. **Circuit Breaker:** পেমেন্ট গেটওয়ে পুরোপুরি ডাউন। অ্যাপ বুঝতে পারল যে বারবার চেষ্টা করে লাভ নেই। সে সার্কিট ওপেন করে দিল এবং কাস্টমারকে সাথে সাথে মেসেজ দিল— "দুঃখিত, পেমেন্ট সার্ভিসটি এখন ডাউন আছে, পরে চেষ্টা করুন।" এতে পেমেন্ট সার্ভারের ওপর বাড়তি চাপ পড়ল না এবং আপনার অ্যাপটিও হ্যাং হলো না।

সারাংশ:

- **Retry:** আবার চেষ্টা করো (যদি সাময়িক সমস্যা হয়)।
- **Timeout:** বেশিক্ষণ অপেক্ষা করো না (রিসোর্স বাঁচাতে)।
- **Circuit Breaker:** সমস্যা বড় হলে রাস্তা বন্ধ করে দাও (সিস্টেমকে বড় বিপর্যয় থেকে বাঁচাতে)।

🛡️ Reliability কী?

Reliability মানে—

System failure হলেও
gracefully handle করবে,
user experience নষ্ট করবে না।

1. Retry — আবার চেষ্টা করো (but smartly)

Retry কী?

কোনো request **temporary failure** হলে
👉 আবার চেষ্টা করা

Service A → Service B ❌ (timeout)

Service A → Retry → Service B ✅

কখন Retry করা উচিত?

✓ Network glitch
✓ Temporary timeout
✓ 5xx server error

❌ Client error (400, 401) → Retry না

❌ Naive Retry Problem

1000 request fail

→ সবাই retry

→ service overload 💥

এটাকে বলে **Retry Storm**

✅ Smart Retry Techniques

🔹 Exponential Backoff

1st retry → 100ms

2nd → 200ms

3rd → 400ms

🔹 Jitter (Random delay)

সব request একসাথে retry না করে
👉 random wait

Retry Golden Line ⭐

“Retry should be bounded, delayed, and combined with backoff to avoid cascading failures.”

2. Timeout — বেশি অপেক্ষা করো না

Timeout কী?

নির্দিষ্ট সময়ের মধ্যে response না এলে
👉 request cancel

Timeout = 2 seconds

2 sec পার → ❌ Stop waiting

কেন Timeout দরকার?

❌ Infinite wait
❌ Thread block
❌ Resource leak

❌ Timeout না থাকলে

Thread stuck

Connection stuck

System slow 😵

✅ Best Practices

- **Short timeout**
- **Client-side timeout**
- Different timeout for:
    - DB
    - External API
    - Internal service

Example:

DB timeout = 50ms

Internal API = 200ms

External API = 1s

Timeout Golden Line ⭐

“Timeouts prevent resource exhaustion and keep systems responsive.”

3. Circuit Breaker — ভাঙা হলে বন্ধ রাখো 🔌

Circuit Breaker কী?

কোনো service বারবার fail করলে
👉 temporarily call বন্ধ করে দেয়

Too many failures

→ Circuit OPEN ❌

→ No request sent

Circuit Breaker States

🟢 Closed

- Normal operation
- Request allowed

🔴 Open

- Failure threshold crossed
- Request blocked

🟡 Half-Open

- Test request
- Success → Close
- Fail → Open again

Example Flow

Service B down

→ Service A retry retry retry ❌

→ Circuit opens

→ Fast failure response

কেন দরকার?

✓ Cascading failure prevent
✓ Faster recovery
✓ Protect downstream service

Circuit Breaker Golden Line ⭐

“Circuit breakers prevent cascading failures by failing fast.”

🔥 Retry + Timeout + Circuit Breaker Together

Client

↓

Timeout (don’t wait forever)

↓

Retry (limited + backoff)

↓

Circuit Breaker (stop if broken)

👉 এগুলো একা একা নয়, একসাথে ব্যবহার হয়

⚖️ Comparison Table

Pattern	**Purpose**	**Protects**
Retry	Temporary failure	Request success
Timeout	Long wait	Resources
Circuit Breaker	Repeated failure	Whole system

🌍 Real-world Example: Payment Service

Pay API call

→ Timeout 1s

→ Retry 2 times

→ If 50% failure → Circuit open

→ Show "Try later"

🧠 Interview Killer Lines ⭐

“Timeouts protect resources, retries handle transient failures, and circuit breakers prevent cascading failures.”

“A resilient system combines all three patterns.”

📝 Memory Trick

Retry = Try again 🔁

Timeout = Don’t wait ⏱️

Circuit Breaker = Stop calling 🔌