Foundational Concepts for Architecture Design

Foundational Concepts for Architecture Design

- Functional vs. Non-Functional requirement
- Back of the envelope calculation
- Some tips

Functional vs. Non-Functional requirement

Analogy – Building a Dream Home

- What your house must have?
  - Think of this as the essential rooms and features you asked for when designing your dream home.
    - A kitchen to cook in
    - A bathroom with a shower
    - Bedrooms to sleep in
    - Doors and windows for access
    - A back garden
  - These are the basic things your house needs to do.

- How well your house is build?
  - Think about the quality, safety, and comfort of that home.
    - Is the house safe and secure? (Security)
    - Can it handle more guests if needed? (Scalability)
    - Does the heater work even on extreme cold nights? (Reliability)
    - Can you enter the house anytime? (Availability)
    - Is it cost-efficient to run? (Cost Optimization)
    - Is it energy efficient? (Performance)
  - These don’t change what the house is — they define how good the experience is.

Analogy – Building a System

- What your system must have?
  - Functionalities or features that the system offers to the end users.
    - A login system so users can access their accounts
    - A way to store and retrieve files
    - A messaging service to communicate between nodes
    - Data backup functionality to recover lost information
    - Load balancers to distribute incoming user requests
  - These are like the rooms and features of your distributed system — they define what it must do.

- How well your system is build?
  - Think about the scalability, security, and performance of the system.
    - Is user data encrypted and secure? (Security)
    - Can the system add more servers when traffic spikes? (Scalability)
    - If one server crashes, does the system still work? (Reliability)
    - Is the system available to users 24/7 without downtime? (Availability)
    - Does it minimize cloud hosting costs? (Cost Optimization)
    - Does it handle requests quickly even under heavy load? (Performance)
  - These describe the quality and experience of using your distributed system, just like the comfort and safety of a house.

Requirements

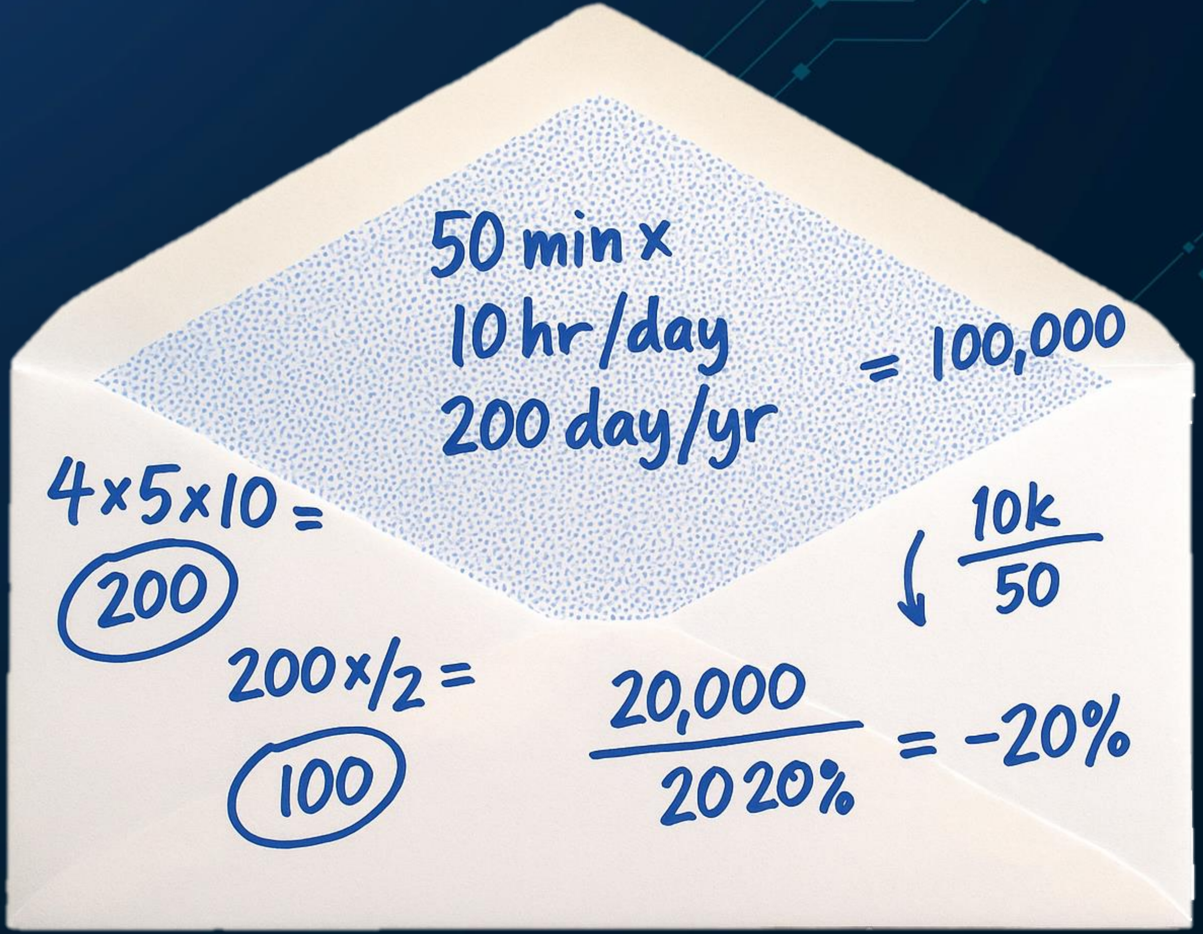
- Functional Requirements (What a system should offer to user?)
  - User can …
  - User will …

- Non-Functional Requirements (How well it does it?)
  - The system should …
  - The system will …

Back of the envelope calculation

Back of the envelope calculation

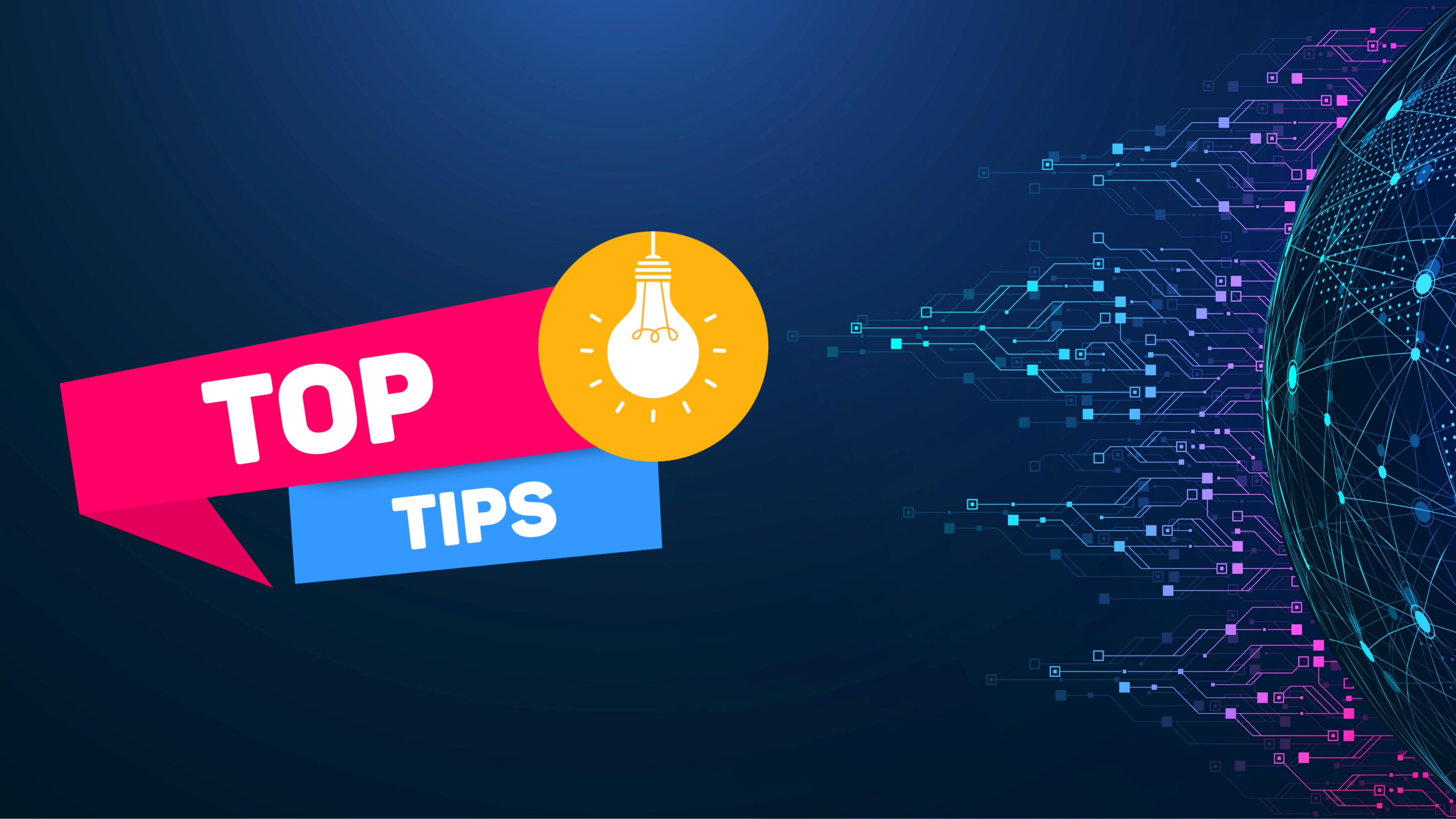
- Rough estimate scribbled quickly—often literally on an envelope
- Focuses on reasonable approximation, not precise accuracy
- Gauges feasibility or provides a ballpark figure
- Ideal for early-stage planning and rapid decision-making



Cost	Capacity	Latency

Example – Calculate storage cost

- You’re planning a cloud app expected to have 1 million users.
- Assumption:
  - Each user generates 100MB of data per month (logs, uploads, etc.).
- Storage Need:
  - 1,000,000 users × 100MB = 100,000,000MB = 100,000GB = = 100TB
- Cost Estimation:
  - Assuming AWS S3 costs $0.023 per GB/month,
    - 100,000GB × $0.023 = $2,300/month
- Takeaway:
  - Enough to assess budget or justify further planning.


TOP
TIPS

My top tips (1)

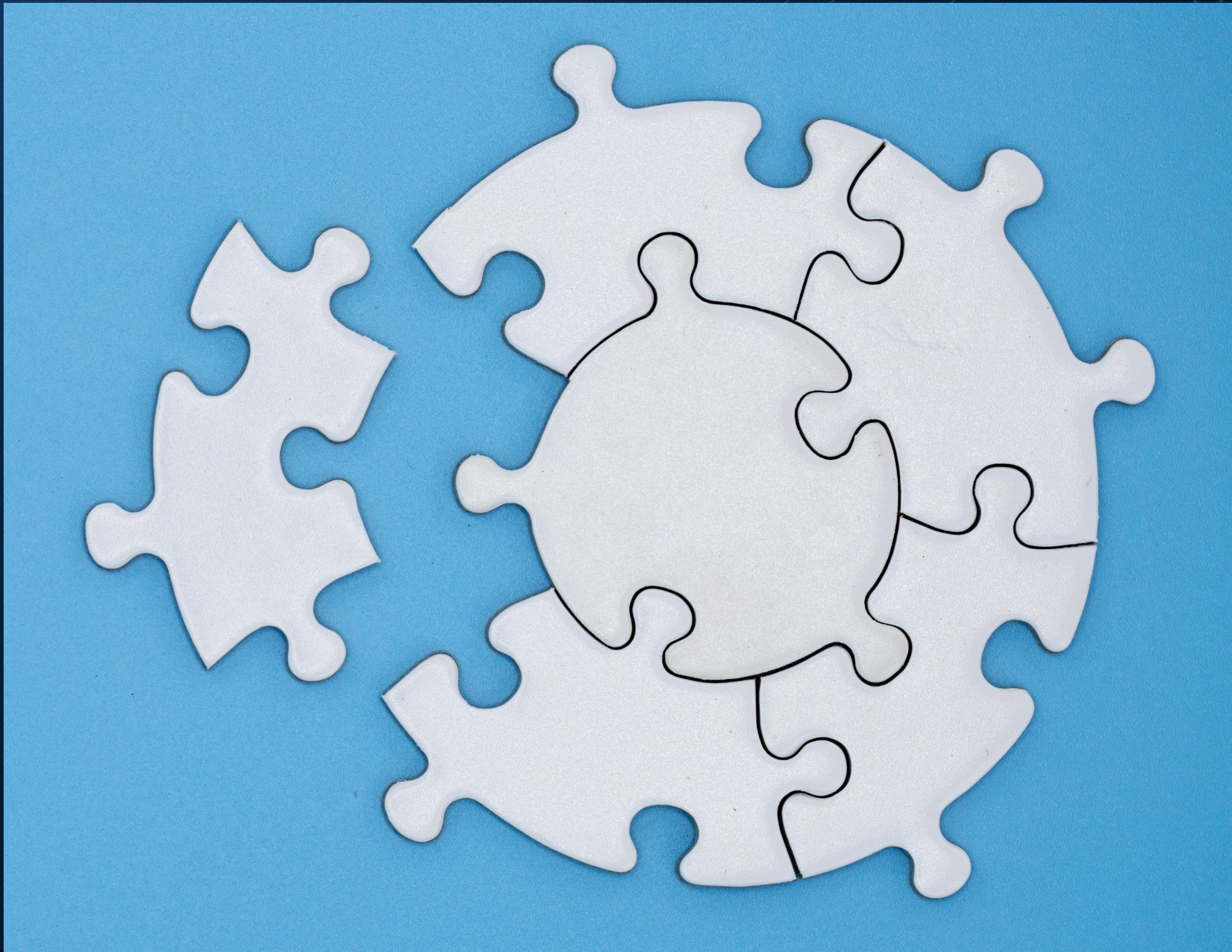
- Start small and enhance
  - Example: Launch food delivery in one city first

My top tips (2)

- Don’t go too deep early
  - Example: Build simple chat before scaling it

My top tips (3)

- Keep systems open-ended
  - Example: API design that allows future extensions



My top tips (4)

- Think stateless
  - Example: Store sessions outside, not in memory

My top tips (5)

- Prefer simplicity over complexity
  - Example: Instead of complex AI, start with "Top 10 popular" products.

Always keep in mind...

- Start small and enhance
  - Example: Launch food delivery in one city first
- Don’t go too deep early
  - Example: Build simple chat before scaling it
- Keep systems open-ended
  - Example: API design that allows future extensions
- Think stateless
  - Example: Store sessions outside, not app memory
- Prefer simplicity over complexity
  - Example: Instead of complex AI, start with "Top 10 popular" products.