System Design Interview Cheat Sheet

Interview Framework

Step 1 10 min.

Understand the Problem

Gather more information about the system requirements and constraints.

Step 2 10 min.

High-Level Design

Explain how each part of the system works together. Start by defining APIs. They are the foundation of the architecture.

Step 3 10 min.

Deep-Dive

Examine system components in detail. Your interviewer may pick a specific area or ask you what you'd like to explore.

Step 4 10 min.

Improve the Design

Take a step back. What are the bottlenecks? How does it scale?

Step 5 5 min.

Wrap Up

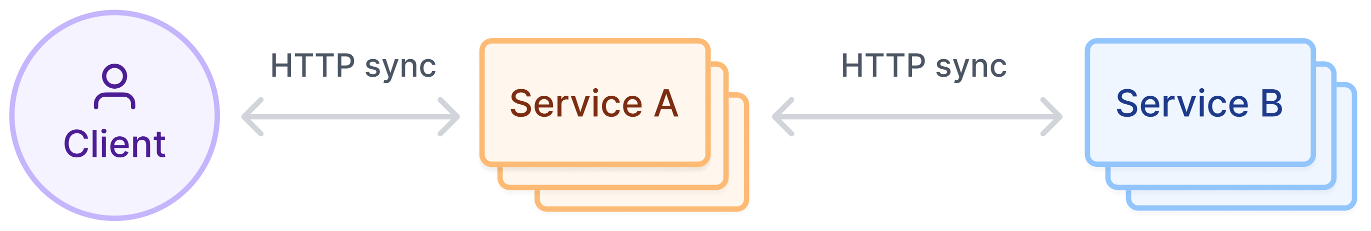
Summarize the requirements, justify your decisions, suggest alternatives, and answer any questions.

API Design Choices

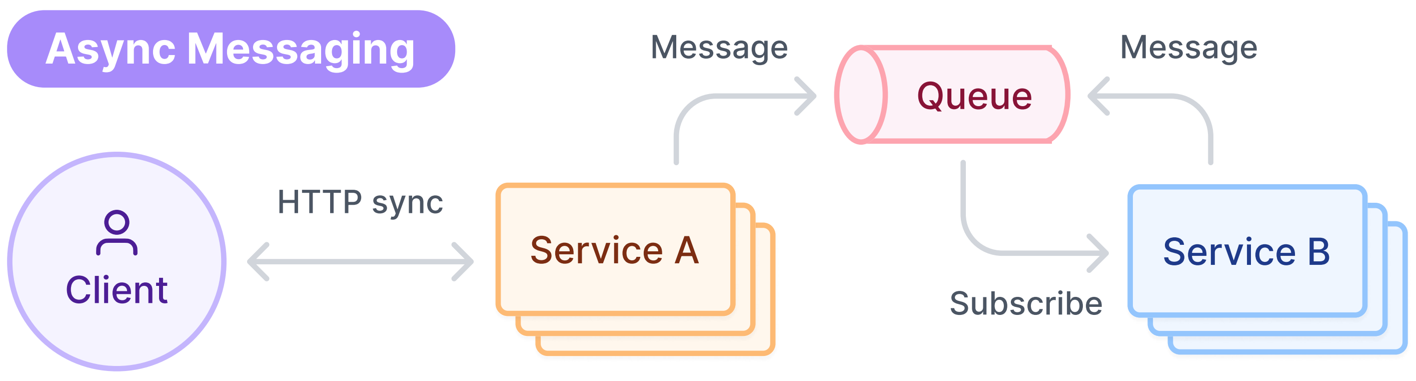
Explain how each part of the system works together. Start by defining APIs and the overall design patterns that your application will use.

	REST	RPC	GraphQL
	Resource	abc 0101 abc	
Properties	✓ resource-oriented ✓ data-driven ✓ flexible	✓ action-oriented ✓ high performance	✓ single endpoint ✓ strongly-typed requests ✓ no data overfetching ✓ self-documenting
Data	JSON, XML, YAML, HTML, plain text	JSON, XML, Thrift, Protobuf, FlatButters	JSON
Use cases	✓ web-based apps ✓ cloud apps ✓ client-server apps ✓ cloud computing services ✓ developer APIs	✓ complex microservices system ✓ IoT applications	✓ high-performance mobile apps ✓ complex systems and microservice-based architectures

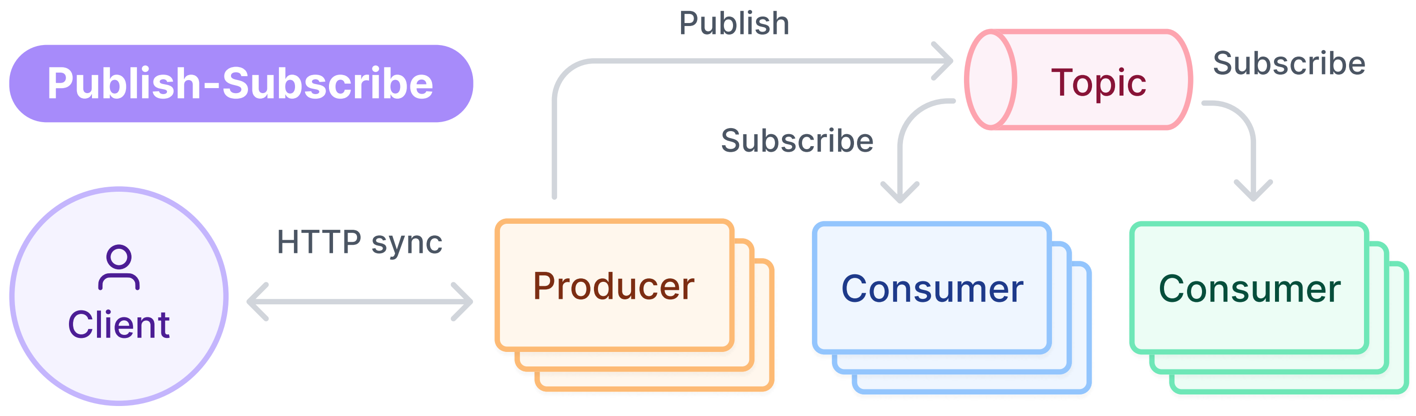
Synchronous



Async Messaging



Publish-Subscribe

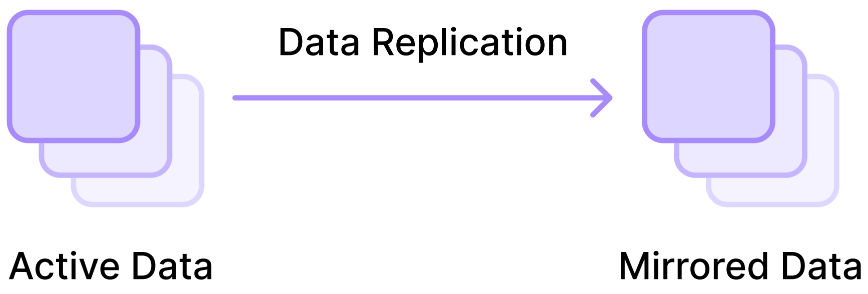


Scalability

Consider the scale of your system. How many users and requests will the server support? What happens with increased demand?

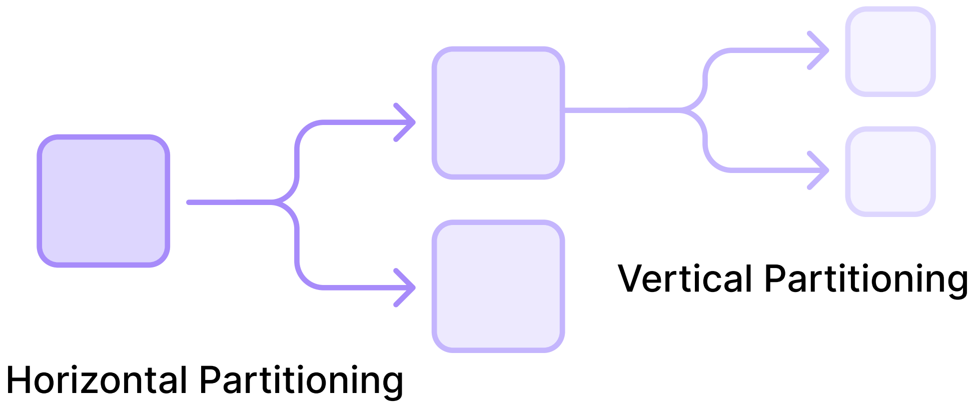
Replication

Is the data important enough to make copies? How important is it to keep all copies the same?



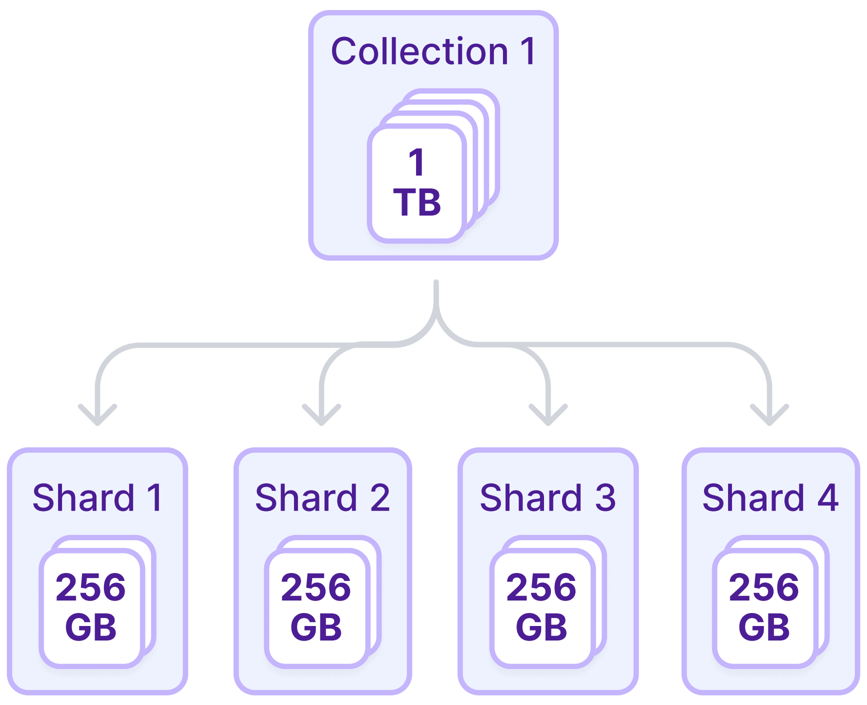
Partitioning

Partitions contain a subset of the whole table. Each partition is stored on a separate server.



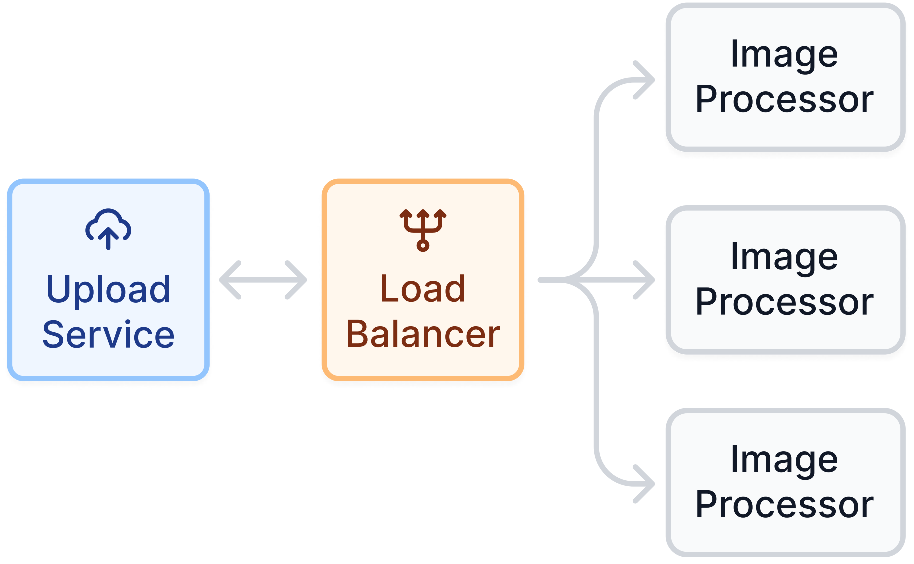
Sharding

Sharding allows a system to scale as data increases, but not all data is suitable for sharding.

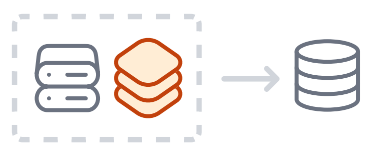


Load Balancing

Load balancing distributes incoming traffic across multiple servers or resources.



Caching

In-memory Cache	Distributed Cache
	
✓ **Latency** - in-memory cache is faster because it doesn't require a network request like distributed.	✓ **Sharing data / Consistency** - data can be shared across machines with a distributed cache. ✓ **Availability** - distributed cache is not affected by individual server failures.

- No. items
- Cache Miss & Hit
- Disk & Memory Usage

- Write-Through
- Read-Through
- Write-Around
- Write-Back

Popular caches:

- In-memory
- Redis
- Memcached
- AWS Elasticache
- GCP Memorystore

Eviction:

- LRU (Least Recently used)
- LFU (Least Freq. used)
- FIFO
- MRU
- Random Eviction
- Least Used
- On-Demand Expiration
- Garbage Collection

- Storing user sessions
- Communication between microservices
- Caching frequent database lookups

tryexponent.com