# 탄력성, 고가용성, 고성능, 데이터 일관성을 고려한 Distributed Processing Solution을 개발하기 위한 구조 설계를 도식도 사용해서 알려줘

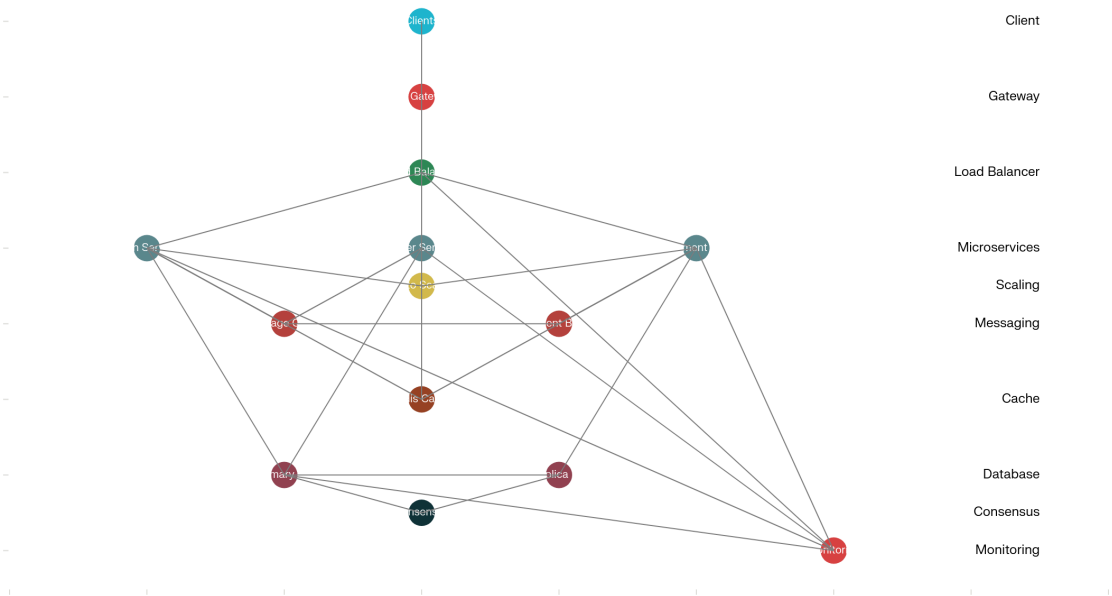
분산 처리 솔루션을 설계할 때 탄력성, 고가용성, 고성능, 데이터 일관성을 모두 달성하려면 각 계층의 역할을 명확히 분리하고, 독립적으로 실패 복구가 가능하며, 병목 없이 확장할 수 있는 구조가 필요합니다.[1] [2] [3]

### 아키텍처 구조 요약

- 클라이언트 접속은 API Gateway에서 라우팅, 인증, 속도 제한을 담당.[2]
- Load Balancer는 요청을 자동으로 여러 서비스 인스턴스에 분배해 과부하 및 장애에 대응.[4]
- Microservices 계층은 Stateless(상태 비저장)로 설계해 필요 시 자동 확장 가능하고, Bulkhead 등 격리 설계 적용.[3] [5]
- 비동기 처리는 Message Queue/Event Bus(예: Kafka, RabbitMQ)로 분리해 장애 전파를 분산.[3]
- 처리 속도를 높이기 위해 분산 Cache(Redis 등) 계층을 배치.[4]
- 데이터 저장은 Replication, Sharding, Consensus(예: Raft, Paxos, Quorum, 2PC 등)이 적용된 분산 DB로 구성해 일관성과 가용성 균형 유지.[6] [7]
- 시스템 전반에 Health Check, 모니터링, 자동 복구, Chaos Engineering을 적용해 다운타임 및 장애 영향 최소화.[5] [3]

### 참고 도식도

아래 도식은 각 계층별 구성 요소 및 주요 흐름을 시각화한 분산 시스템 구조입니다.




Distributed Processing Solution Reference Architecture Diagram

## 주요 설계 포인트

- **Redundancy(중복성):** Load Balancer/Service/DB 모두 다중 인스턴스로 장애에 강함.[2] [6]
- **Consistency:** 분산 DB는 Replication, Consensus 알고리즘(예: Quorum, Raft 등) 활용. 일관성 수준(CP or AP)은 CAP이론에 따라 서비스 특성에 맞게 결정.[8] [7]
- **High Performance:** Stateless 방식, 분산 캐시 활용, 비동기 메시징, 자동 스케일링 등.[9]
- **Monitoring/Recovery:** 모니터링 시스템과 복구/자동 Failover, Chaos Engineering 등 운영 자동화 적용.[3] [5]

이 구조를 기본으로, 서비스 특성에 맞게 일관성/가용성/확장성의 균형을 맞추는 것이 이상적입니다.

⁂


1. https://www.geeksforgeeks.org/system-design/resilient-distributed-systems/
2. https://vfunction.com/blog/distributed-architecture/
3. https://www.geeksforgeeks.org/system-design/architecture-patterns-for-resilient-systems/
4. https://redis.io/blog/high-availability-architecture/
5. https://www.ijsat.org/papers/2025/1/2440.pdf
6. https://www.meegle.com/en_us/topics/distributed-system/distributed-system-reliability
7. https://systemdesign.one/consistency-patterns/
8. https://www.designgurus.io/blog/distributed-system-design-guide-for-beginners
9. https://www.imaginarycloud.com/blog/scalability-patterns-for-distributed-systems-guide

10. https://aws.amazon.com/blogs/database/building-resilient-applications-design-patterns-for-handling-database-outages/
11. https://estuary.dev/blog/distributed-architecture/
12. https://www.multiplayer.app/blog/6-best-practices-for-backend-design-in-distributed-system/
13. https://www.arxiv.org/pdf/2408.10281.pdf
14. https://gprjournals.org/journals/index.php/ijea/article/download/355/374/1035
15. https://www.sciencedirect.com/science/article/pii/S0278612525001876
16. https://docs.aws.amazon.com/pdfs/whitepapers/latest/availability-and-beyond-improving-resilience/availability-and-beyond-improving-resilience.pdf
17. https://docs.oracle.com/en/database/oracle/oracle-database/12.2/haovw/ha-architectures.html
18. https://hackernoon.com/a-senior-engineers-guide-to-scalable-and-reliable-system-design
19. https://community.f5.com/kb/technicalarticles/customer-edge-site-high-availability-for-application-delivery---reference-archit/330332
20. https://www.geeksforgeeks.org/system-design/scaling-distributed-systems/