perplexity

주요 컴포넌트별 역할과 상호작용

분산 처리 솔루션에서 **탄력성, 고가용성, 고성능, 데이터 일관성**을 모두 만족하려면 각 컴포넌트가 어떤 역할을 하는지, 그리고 어떻게 서로 연계되는지 명확히 해야 해요. 아래에 대표적인 계층과 컴포넌트별 역할, 그리고 **상호작용 흐름**을 설명할게요.

1. API Gateway / Entry Point

- **역할:** 클라이언트의 모든 요청이 가장 먼저 거치는 관문입니다. 인증, 권한 부여, 라우팅, 요청 유효성 검사, 속도 제한(rate limit) 등을 처리합니다.
- **상호작용:** 클라이언트 ↔ API Gateway

2. Load Balancer

- **역할:** 들어오는 트래픽을 여러 백엔드 서비스 인스턴스로 분배하여 과부하/장애를 막고, 트래픽 증가 시 자동 확장(스케일 아웃)합니다.
- **상호작용:** API Gateway ↔ Load Balancer ↔ Service Instances

3. Microservices (Service Instances)

- **역할:** 실제 비즈니스 로직(예: 유저 관리, 주문 처리 등)을 수행합니다. 각 인스턴스는 상태를 저장하지 않고, 필요 시 언제든 쉽게 늘리고 줄일 수 있습니다.
- **상호작용:**
  - 로드밸런서에서 트래픽 수신
  - 내부적으로 메시지 큐 또는 이벤트 버스, DB로 데이터 전달 또는 조회/저장

4. Distributed Cache (예: Redis, Memcached)

- **역할:** 데이터베이스 부하를 줄이고, 읽기 성능을 높이기 위해 자주 사용하는 데이터를 메모리에 저장합니다.
- **상호작용:**
  - 마이크로서비스 ↔ 분산 캐시 (읽기/쓰기)

5. Message Queue / Event Bus (예: Kafka, RabbitMQ)

- **역할:** 비동기 처리가 필요한 작업(예: 이메일 발송, 로그 저장, 대용량 데이터 분석 등)의 요청 메시지를 잠시 저장하고, 순차/동시로 처리할 워커들에게 전달합니다.
- **상호작용:**
  - 마이크로서비스 ↔ 메시지 큐
  - 워커(비즈니스 프로세스) ↔ 메시지 큐

6. Distributed Database (예: Cassandra, MongoDB, CockroachDB, RDBMS+Replication)

- **역할:** 서비스 핵심 데이터를 저장하고, 여러 서버에 분산하여 장애/부하에 강하며, 복제와 일관성 정책(예: Strong, Eventual Consistency)을 선택할 수 있습니다.
- **상호작용:**
  - 마이크로서비스 ↔ 데이터베이스
  - 분산 DB 사이에 데이터 복제, 컨센서스 알고리즘(Quorum, Paxos, Raft 등)을 통한 일관성 확보

7. Monitoring, Health Check, Auto Recovery System

- **역할:** 전체 시스템을 모니터링(지표 수집, 장애 감지)하고, 이상 발생 시 자동 알림 및 복구(예: 자동 재기동, 스케일 아웃/인) 역할을 담당합니다.
- **상호작용:** 모든 컴포넌트 ↔ 모니터링 시스템

전체 상호작용 예시 흐름 (요청 → 처리 → 결과)

1. **클라이언트**가 요청을 API Gateway에 전달.
2. API Gateway가 **인증/권한** 점검 후 Load Balancer로 전달.
3. Load Balancer가 상태 비저장 **마이크로서비스 인스턴스** 중 하나로 트래픽 배분.
4. 마이크로서비스는 데이터를 조회할 경우, **Cache**에서 먼저 시도(Miss일 경우 DB 조회 및 Cache 갱신).
5. 무거운 작업일 경우, 메시지를 **Queue/Event Bus**에 넣고, 워커가 비동기로 처리.
6. 상태 데이터 저장은 **분산 DB**에 쓰기/읽기, 각 노드간 컨센서스 메커니즘(Quorum, Raft 등)으로 **일관성 보장**.
7. 모든 컴포넌트는 **모니터링/헬스체크** 시스템이 주기적으로 상태 점검 및 장애 감지, 필요 시 자동 복구.

기억해야 할 점

- 각 계층은 **장애 및 트래픽 급증에 대비해 중복/확장성** 구조로 설계합니다.
- 데이터 일관성과 가용성의 트레이드오프(CAP 이론)를 서비스 성격에 맞게 조정해야 합니다.
- 분산 캐시와 메시지 큐는 장애 전파 차단, 부하 분산, 성능 향상에 매우 중요합니다.

이제 이 설명을 바탕으로 각 계층별 더 자세히, 혹은 특정 시나리오(예: 장애 상황, 데이터 동시성 처리 등) 중심으로 더 깊은 예시가 필요하다면 원하는 부분을 말해 주세요.