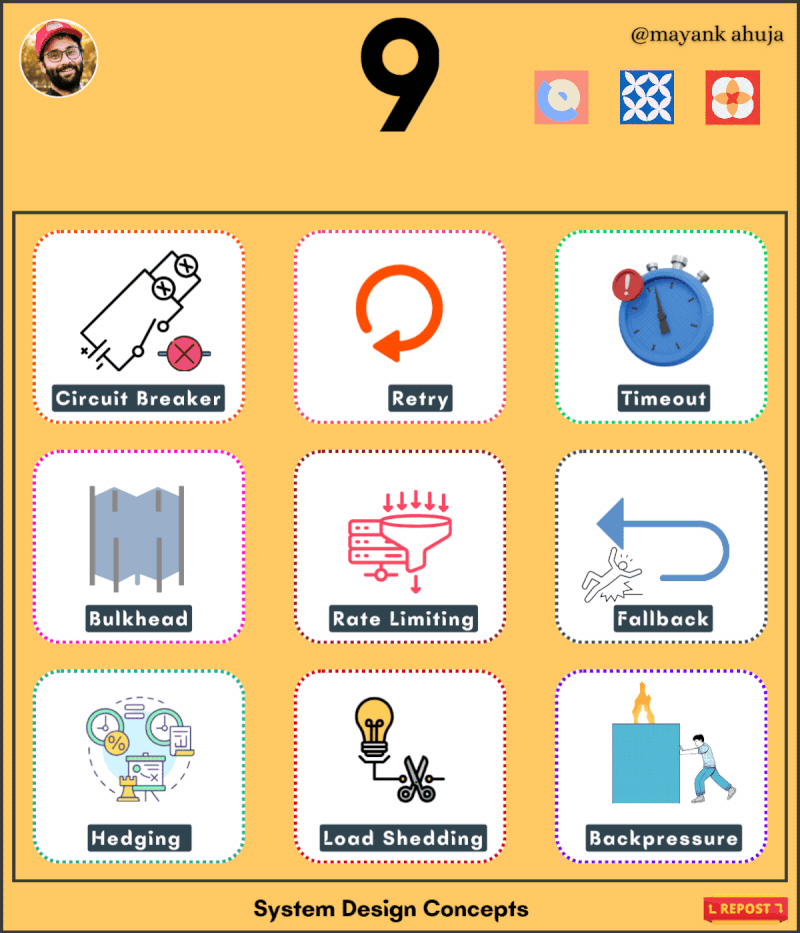
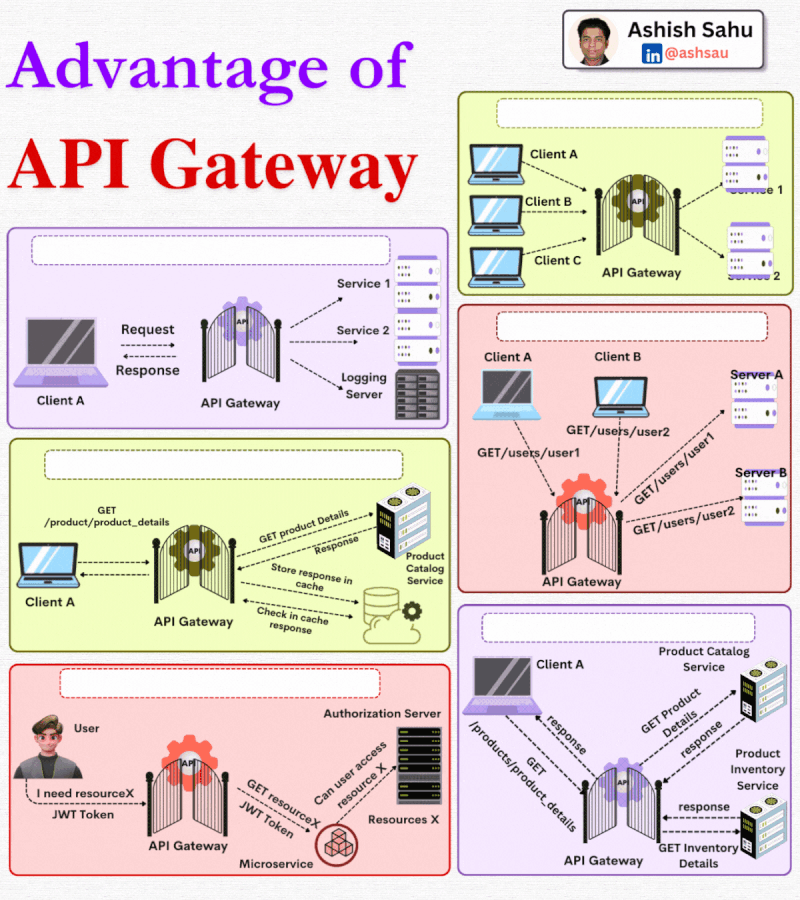


🟢 Peer-to-Peer (P2P): Decentralized architecture where each node acts as both client and server, sharing resources and responsibilities. **🌐** API Gateway: Acts as an entry point for all client requests, routing them to appropriate microservices, handling authentication, rate-limiting, etc.  
  
📢 Pub-Sub: A messaging pattern where publishers send messages to a topic, and subscribers receive them without direct communication between the two.  
  
🔄 Request-Response: Traditional synchronous pattern where a client sends a request and waits for a response from the server.  
  
📊 Event Sourcing: Capturing all changes to an application's state as a sequence of events, which can be replayed or queried.  
  
⚙️ ETL (Extract, Transform, Load): A process that extracts data from sources, transforms it to fit business needs, and loads it into a destination system.  
  
📦 Batching: Accumulating data or tasks over a period of time and processing them all at once to optimize performance.  
  
🚀 Stream Processing: Real-time processing of data as it flows in, enabling quick insights and actions.  
  
🎯 Orchestration: Managing and coordinating multiple services, workflows, or tasks to ensure they work together seamlessly**.**

**Resiliency Patterns**

"Fall seven times, stand up eight."  
  
A brief intro.

  
  
[1.] **Circuit Breaker**◾ Acts like an electrical circuit breaker.  
◾ When a service experiences repeated failures, the circuit breaker 'trips' and stops sending requests to that service for a period of time.  
◾ This allows the failing service to recover without being overwhelmed.  
  
The main circuit breaker states -  
◾ Closed - Requests are allowed to pass through.  
◾ Open - Requests are immediately rejected with an error.  
  
◾ Circuit breakers are effective for protecting against cascading failures and isolating problematic services.  
  
[2.] **Retry**  
◾ When a request fails, the system automatically retries it a certain number of times before giving up.  
◾ This can help overcome transient errors like network glitches or temporary unavailability.  
◾ Improves system availability and can mask transient errors.  
◾ Be mindful of retry storms (where excessive retries overload the system) and implement exponential backoff (increasing the time between retries).  
  
[3.] **Timeout**  
◾ Sets a maximum time limit for a request.  
◾ If a response is not received within the timeout period, the request is considered a failure.  
  
[4.] **Bulkhead**  
◾ The Bulkhead pattern isolates different parts of an application into pools or compartments.  
◾ This isolation limits the impact of failures or overload in one compartment, preventing it from cascading and affecting the entire system.  
  
[5.] Rate Limiting  
◾ Controls the rate of incoming requests to protect a system from being overwhelmed.  
◾ Protects against denial of service attacks, ensures fair usage and helps maintain system stability.  
  
[6.] Fallback  
◾ Provides an alternative (often less ideal) response or action when the primary one fails.  
◾ Improves system availability and user experience by providing some level of service even when the primary function is unavailable.  
  
[7.] Hedging (Redundancy)  
◾ Sends duplicate requests to multiple identical services and uses the fastest response.  
◾ Mitigates the impact of slow responses and failures, improving system responsiveness.  
  
[8.] Load Shedding  
◾ Drops non-critical requests when a system is overloaded to protect its core functionality.  
◾ Helps maintain system stability and availability during peak loads.  
  
[9.] Backpressure  
\*\* shares some similarities with other resilience patterns  
◾ The core mechanism of backpressure is a feedback loop between the producer (sending data) and the consumer (receiving data).  
◾ The consumer signals its capacity to the producer, allowing the producer to adjust its output rate dynamically.  
  
Several backpressure strategies exist -  
◾ Reactive Pull - The consumer explicitly requests data from the producer, pulling data at its own pace.  
◾ Rate Limiting - The producer limits its output rate based on the consumer's feedback.  
◾ Buffering - A buffer is used to temporarily store data when the consumer is slow.



- Single Point of Interaction: API Gateway centralizes communication between services.  
- Load Balancing: Distributes traffic evenly across services.  
- Aggregation of Results: Combines responses from multiple services.  
- Security and Authentication: Manages security centrally.  
- Performance Optimization: Reduces delays with caching.  
- Centralized Logging and Monitoring: Collects logs and metrics from all services.