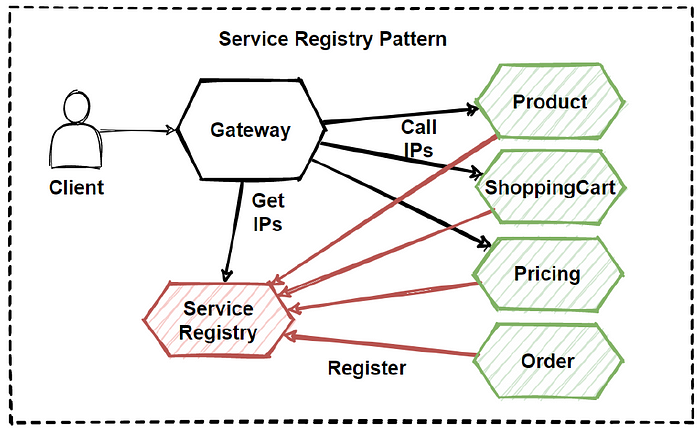
**1. Service Registry**

Tools **Consul, Eureka**(springcloud)



**2. Circuit Breaker**

Tools – Netflixs Hystrix / Spring Cloud CB

Continue function when one or more services fail

**3. API Gateway** ((entry point for all incoming API requests)

Request/response transformation, rate limiting, authentication, authorization and caching

**4. Saga Pattern**

Two types

* Orchestration
* Choreography (without central service)
* Local transactions each updates state of service

Workflow becomes confusing when adding new steps

Risk of cyclic dependencies

Complex if large number of steps

**5. Even Sourcing Pattern**

Persists all events in the application, allow state to be reconstructed. Auditability, Scalability, Flexibility, Fault-tolerance

**6. Command Query Responsibility Segregation (CQRS)**

Separates commands and queries

Command – receives command, write to database

Query – read from database, send to client

**7. Bulkhead Pattern**

Failure in one part, not affect other parts

**8. Backends for Frontends (BFF)**

separate back-end service for each frontend to handle the specific needs of that interface

to optimize the data flow, caching, and authentication mechanisms for the unique needs of the front-end while keeping the back-end services modular and decoupled.

**9. Externalized Configuration**

Storing configuration data outside of the application code, making it easier to manage configuration changes

**10. Event-Driven Architecture**

Services communicate each other by emitting events

**11. Database per Service**

Each service has its own database, which allows services to operate independently

**SOLID**

Single Responsibility | Open Closed | Liskov Substitution

Interface Segregation | Dependency Injection

**ACID**

Atomicity | Consistency | Isolation | Durability

**Cache**

L1 Cache – Hibernate session / transaction cache

L2 Cache – among threads(eg EH cache of hibernate)

**Design Patterns Programming**

**- Creational Patterns** (Object creations)

- Factory

- Abstract Factory

- Builder (simplify complext obj creation)

- Prototype (new obj from copy existing)

- Singleton

- **Structural patterns** (How assemble objects)

- Adapter (for incompatible objects)

- Bridge

- Composite (tree structure)

- Decorator (add behaviour)

- Facade (simplifies subsystem)

- Flyweight (minimise m/y)

- Proxy (Interface to another resource)

- **Behavioural patterns** (Commn and assignments)

- Chain of Respbty (for incompatible objects)

- Command

- State

- Strategy

- Template Method

|  |  |
| --- | --- |
| **SQL** | **NoSQL** |
| RDBMS | Non-relational / Distributed database system. |
| These databases have fixed or static or predefined schema | They have a dynamic schema |
| These databases are not suited for hierarchical data storage. | These databases are best suited for hierarchical data storage. |
| These databases are best suited for complex queries | These databases are not so good for complex queries |
| Vertically Scalable | Horizontally scalable |
| Follows ACID property | Follows CAP(consistency, availability, partition tolerance) |
| Examples: MySQL, PostgreSQL, Oracle, MS-SQL Server, etc | Examples: MongoDB, HBase, Neo4j, Cassandra, etc |

|  |  |
| --- | --- |
| **SQS** | **Kafka** |
| Standard / FIFO (more than once / ordered) | real-time processing of streaming data |
| Lack of support for broadcast messages | Allows broadcasting |