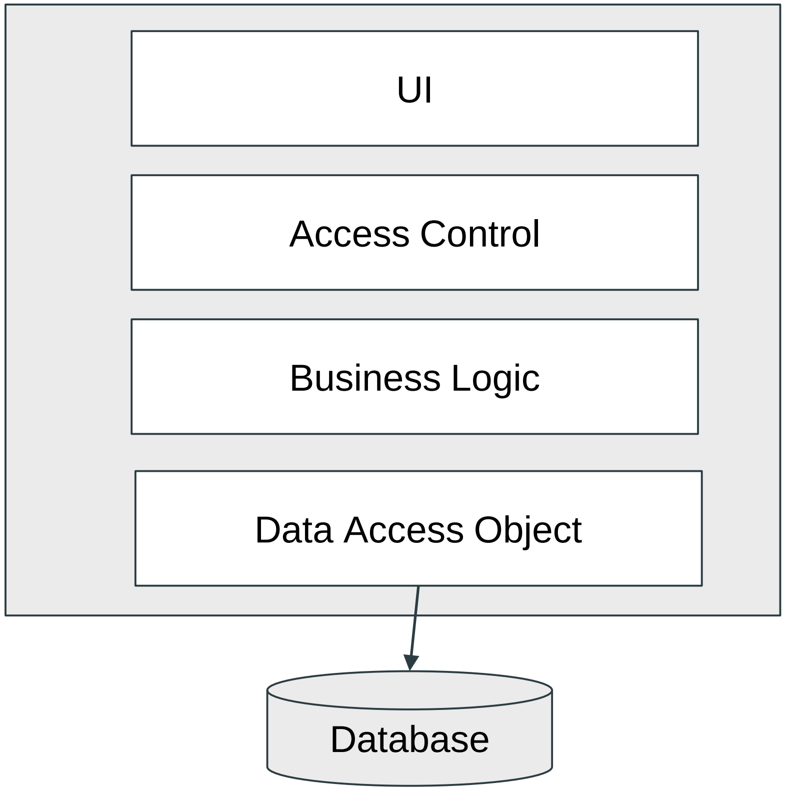
SOA vs MSA

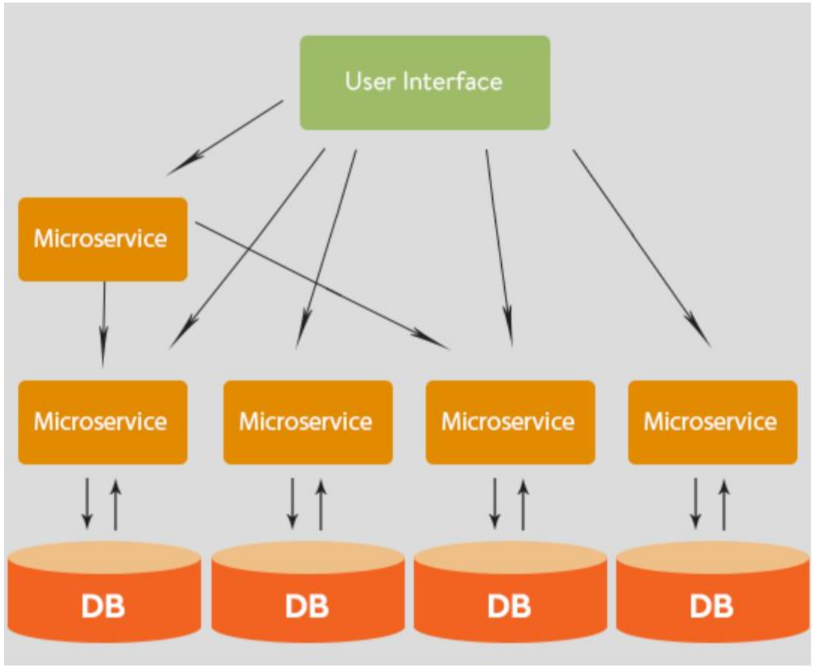
Microservices Architecture – A developer perspective

Monolithic Application Architecture

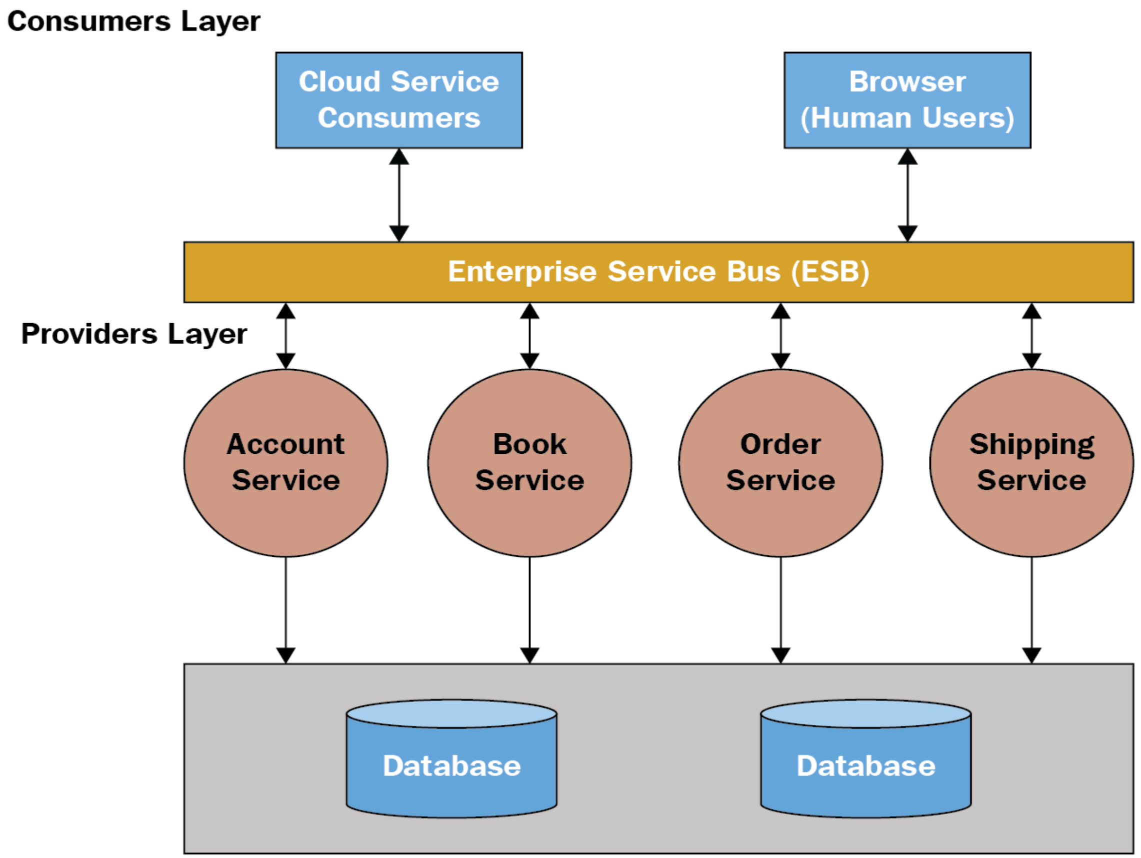


MSA Generic Architecture

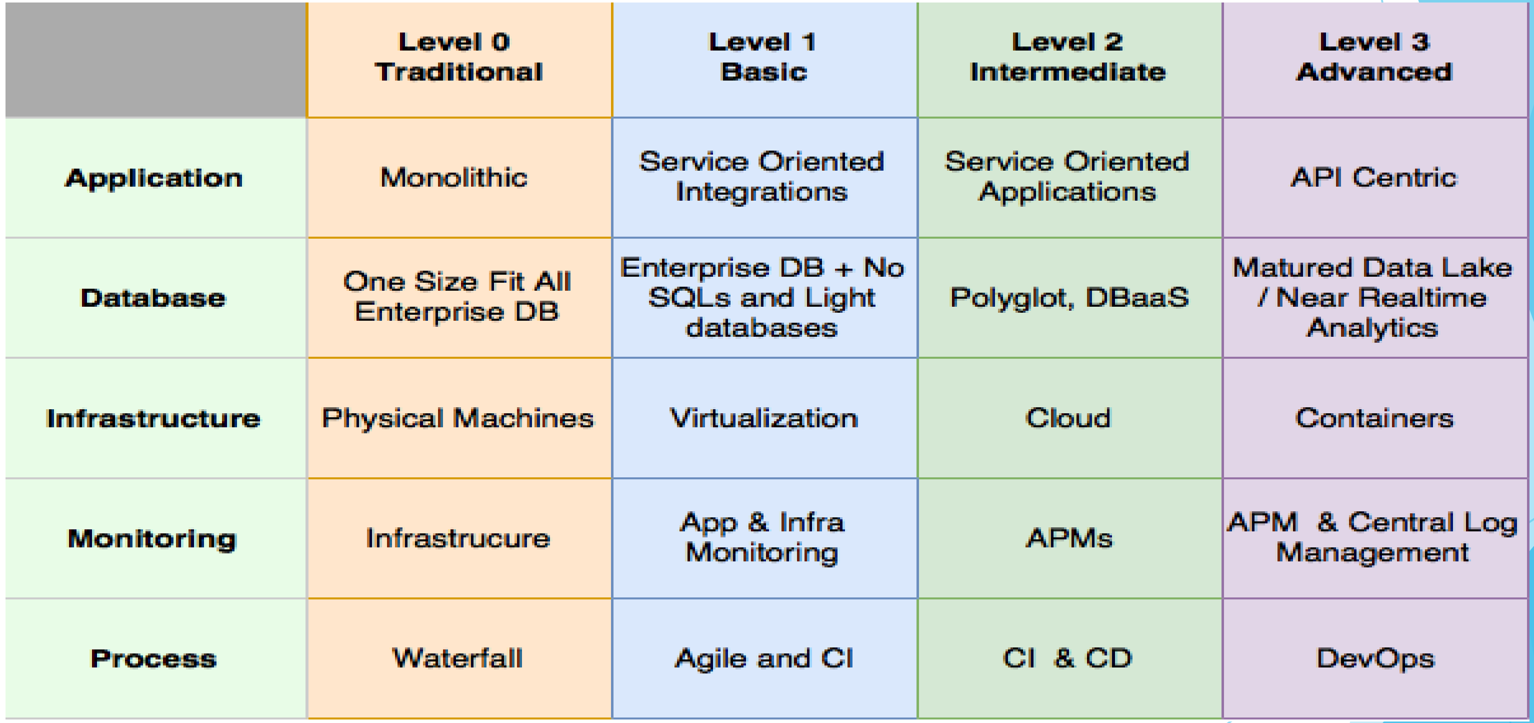
- Note: Ideally there should be a service to render UI as well



SOA Generic Architecture



MSA Maturity Level Matrix



	Level 0 Traditional	Level 1 Basic	Level 2 Intermediate	Level 3 Advanced
Application	Monolithic	Service Oriented Integrations	Service Oriented Applications	API Centric
Database	One Size Fit All Enterprise DB	Enterprise DB + No SQLs and Light databases	Polyglot, DBaaS	Matured Data Lake / Near Realtime Analytics
Infrastructure	Physical Machines	Virtualization	Cloud	Containers
Monitoring	Infrastrucure	App & Infra Monitoring	APMs	APM & Central Log Management
Process	Waterfall	Agile and CI	CI & CD	DevOps

Characteristics of MSA

1. Scalability -- How our services can be scaled on demand?
2. Availability -- How can we ensure that our services are available all the time or meet SLA?
3. Resiliency -- How our services can be made fault tolerant?
4. Independent, autonomous -- Are our services independent and autonomous?
5. Decentralized governance -- Can we manage services end to end in DevSecsOPs independently?
6. Failure isolation-- What happens if one service is not available and some composite service is also using it?
7. Auto-Provisioning -- Can a service be provisioned based on an event?
8. Continuous delivery through DevOps -- Are services using CI/CD for DevSecOps?
9. If we have exposed some services as API are they following 12-factor guidelines of at least implemented versioning and metering?

Problem with traditional capacity planning

Traditional capacity planning:

1. Under provisioned
2. Over provisioned

We should utilize cloud to provision optimum/on-demand provisioning

Async, Sync or event based request processing

From platform point of view:

1. Categorize synchronous and asynchronous processing. Try to move more and more to asynchronous processing to plan for CROPS kind of infrastructure of future as platform
2. Use microservices where on-demand scaling is required

CROPS (Cost-optimized, Resilient, Operationally-excellent, Performant and Secure)

Async vs Sync or event -- new customer registration

<Your Story and Imagination>

Async vs Sync or event – Order fulfilment

<Your Story and Imagination>

The Art of Scalability

Decomposition can have three directions:

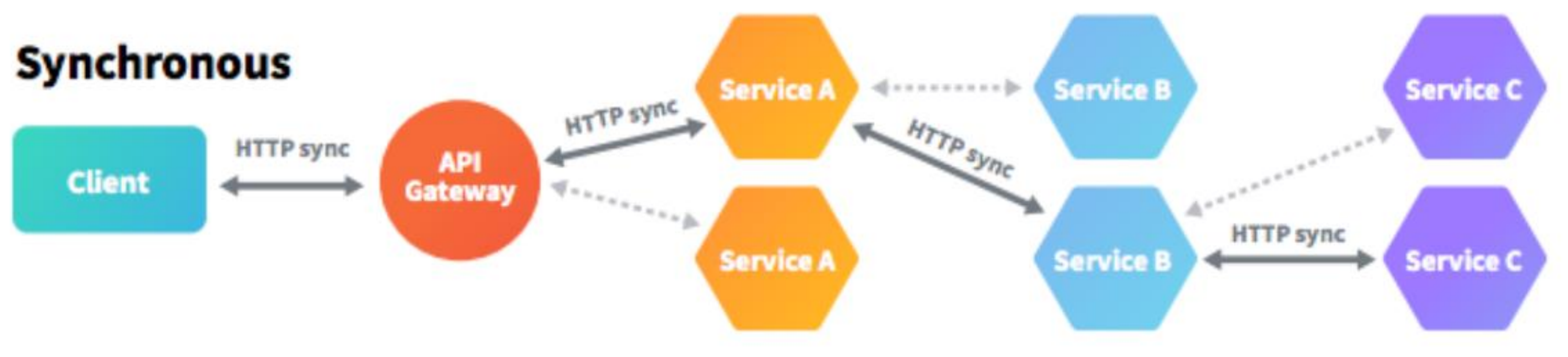
Horizontal duplication: scale by cloning similar components and using load balancing.

Functional decomposition: scale by dividing different logical parts of the system.

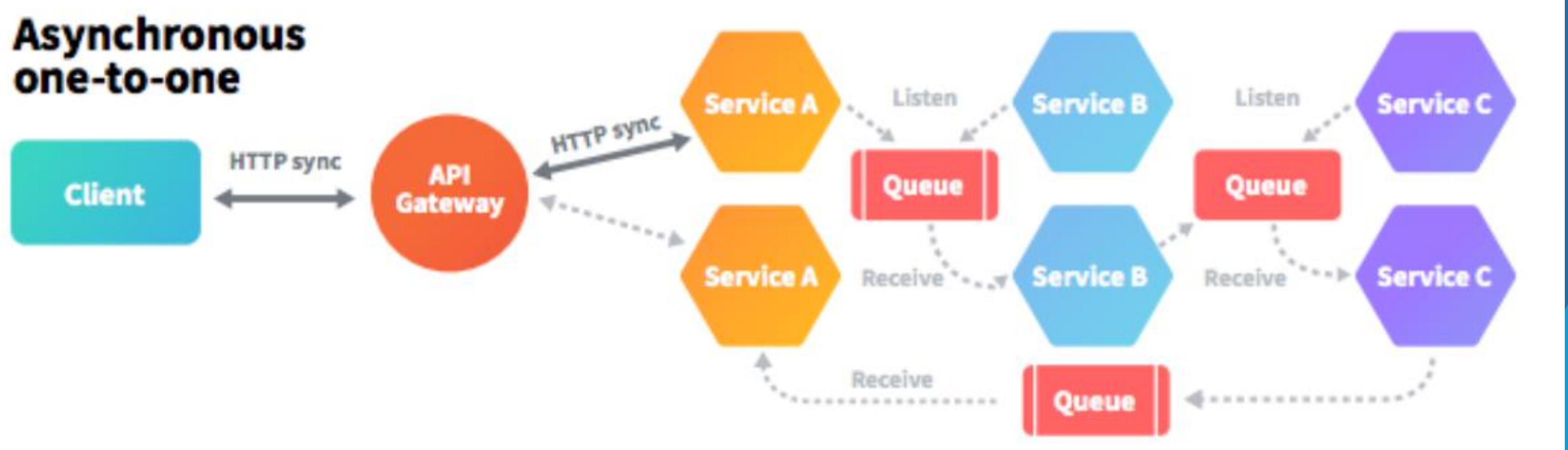
Data partitioning: scale by splitting non-dependent similar data.

Note: Virtually every system/application can be scaled, it is time, effort and resource all that matters while scaling.

Inter Service Communication - Synchronous Request Processing

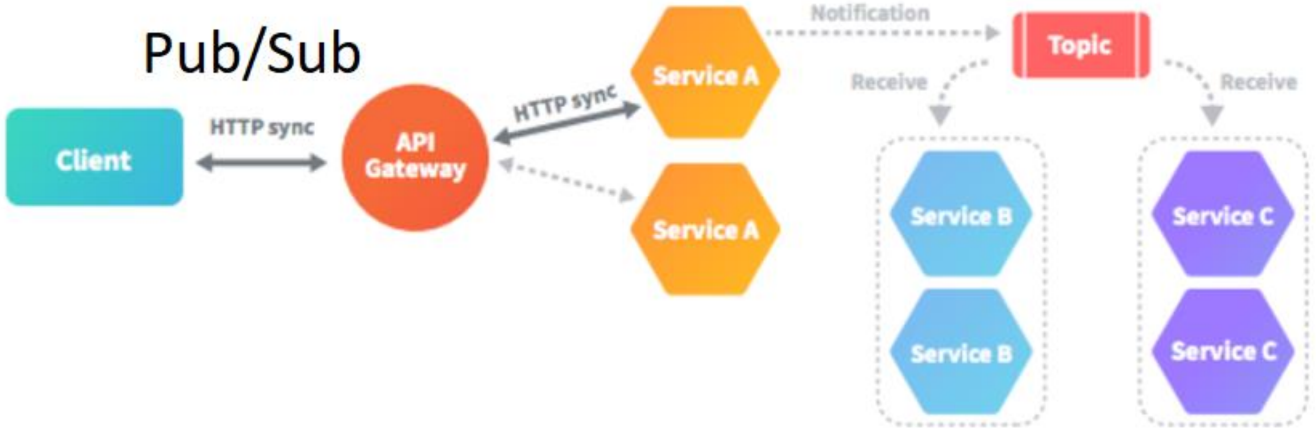


Inter Service Communication - Asynchronous Request Processing



Inter Service Communication - Asynchronous Request Processing

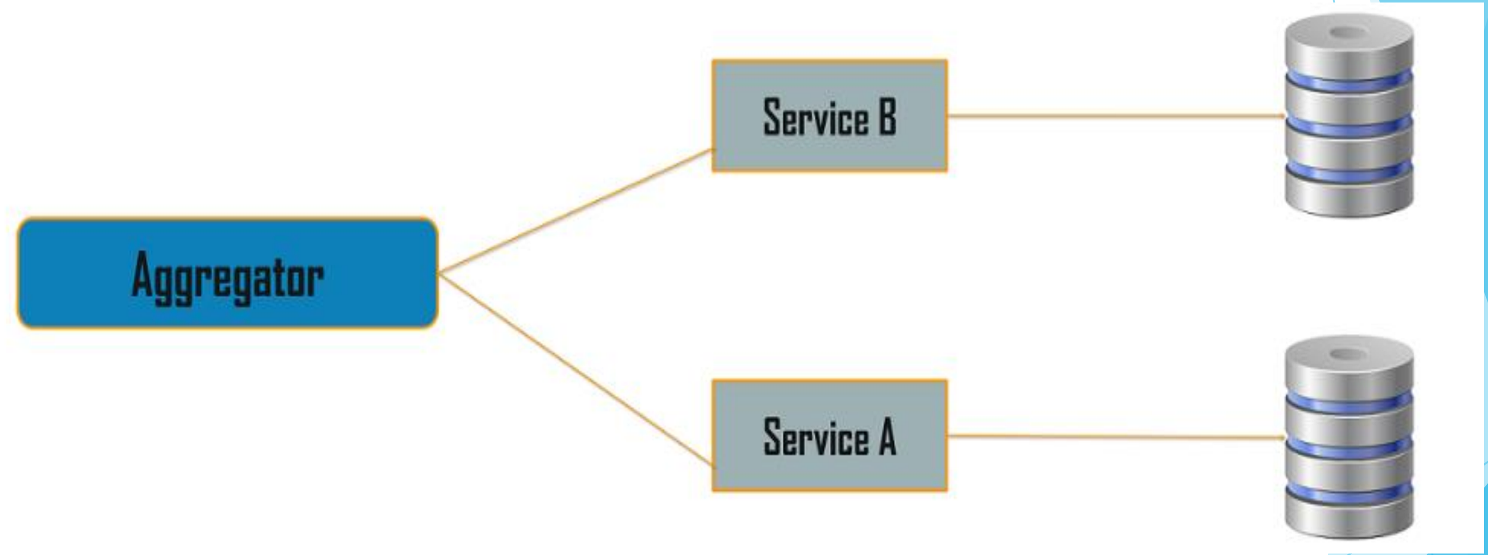
Pub/Sub



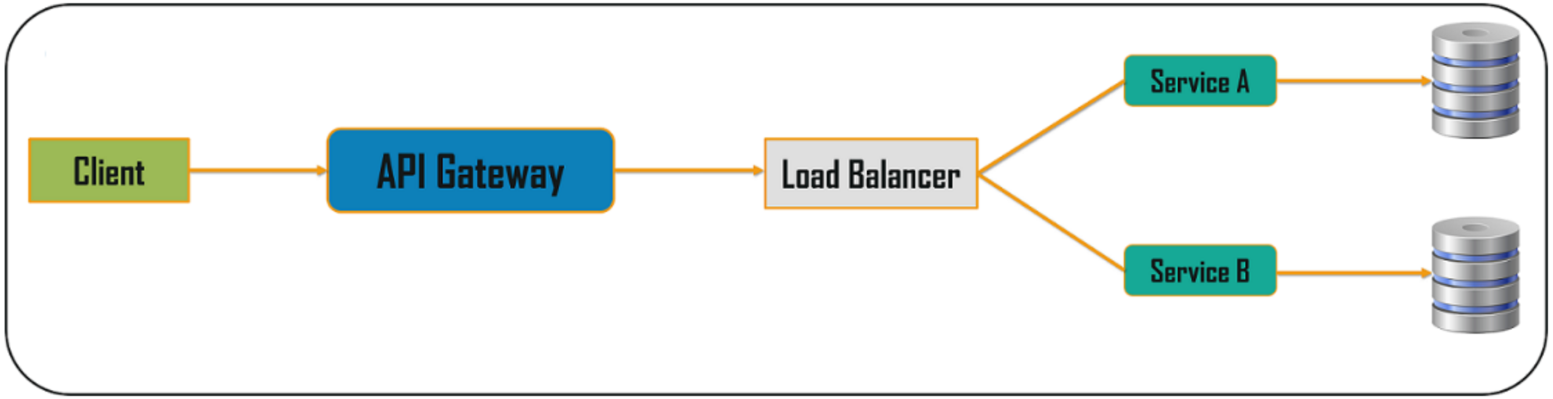
Design Patterns of Microservices

1. Aggregator - development pattern
2. API Gateway -- deployment pattern
3. Chained or Chain of Responsibility -- development pattern
4. Asynchronous Messaging -- development pattern
5. Database or Shared Data -- devops
6. Event Sourcing -- devops
7. Branch -- development pattern
8. Command Query Responsibility Segregation - devops
9. Circuit Breaker -- devops
10. Decomposition - art of breaking monolith into automic services

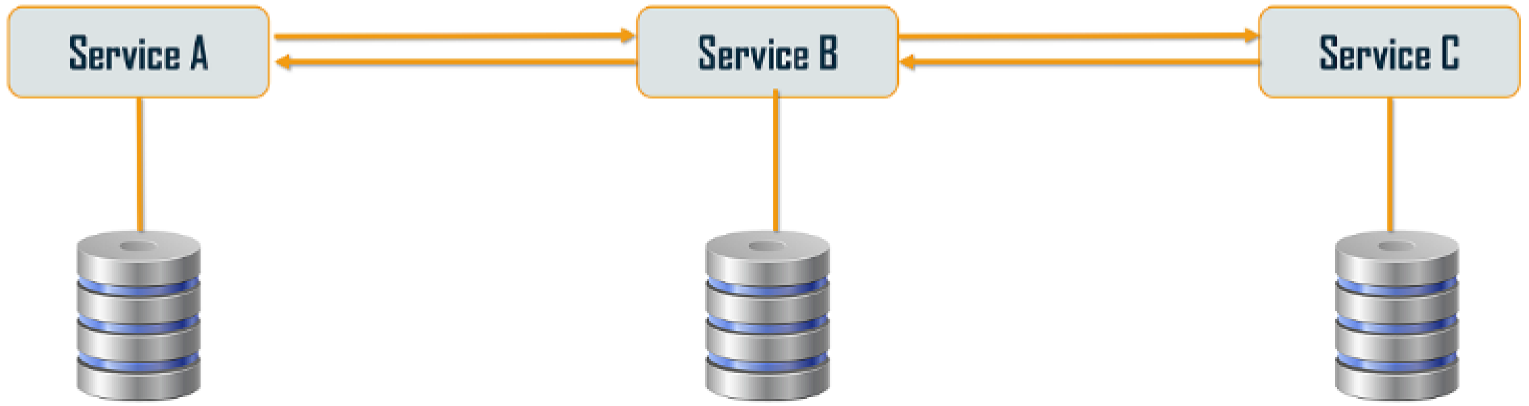
1. Aggregator Pattern



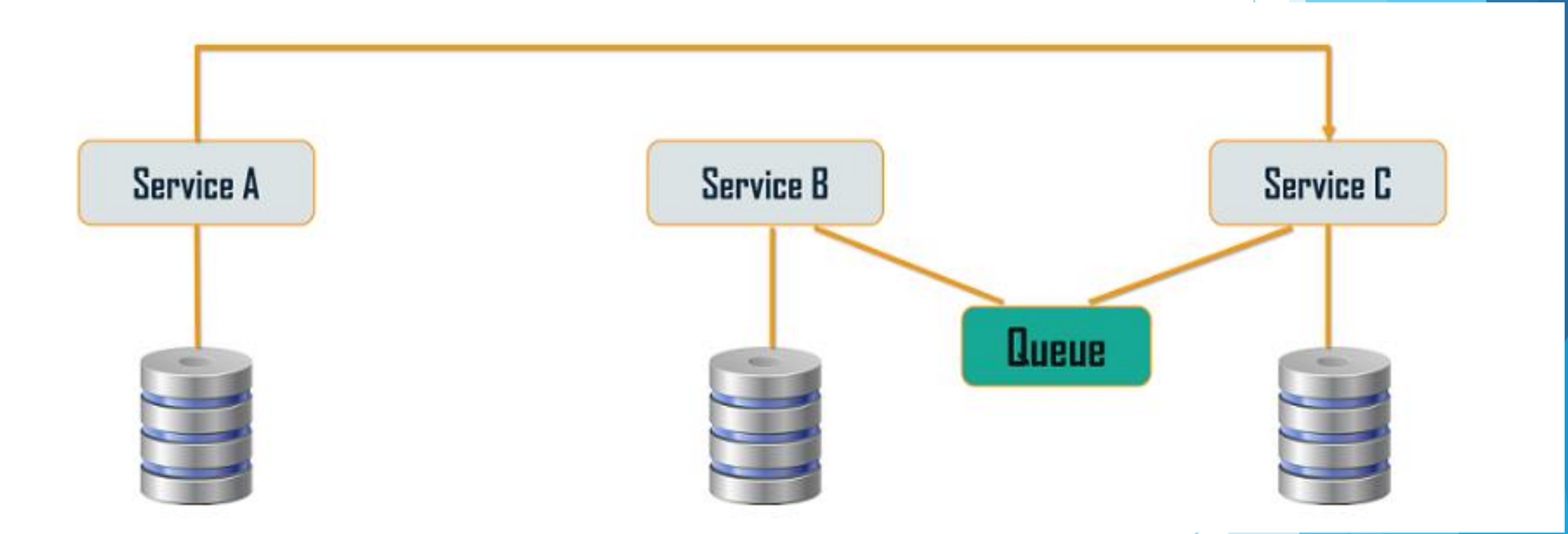
2. API Gateway Pattern



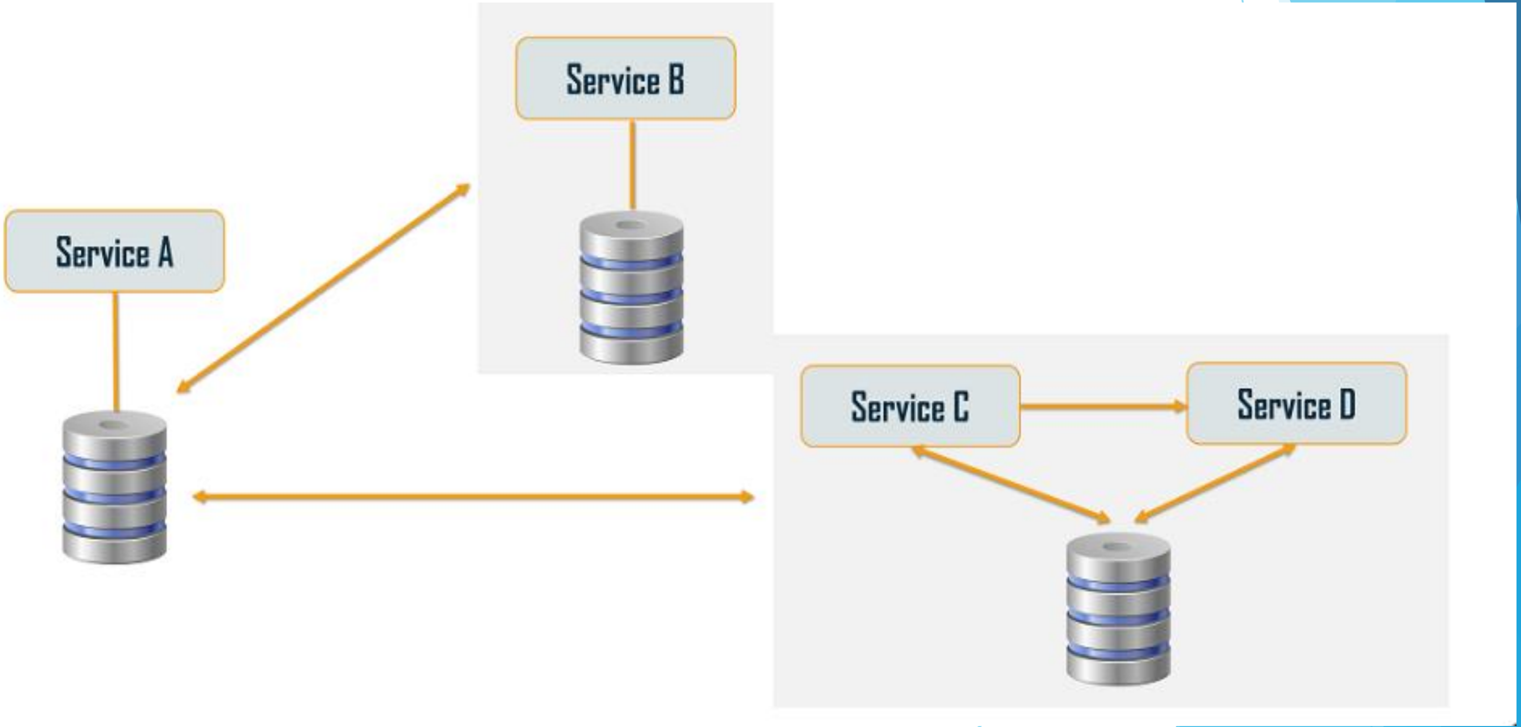
3. Chained or Chain of Responsibility Pattern



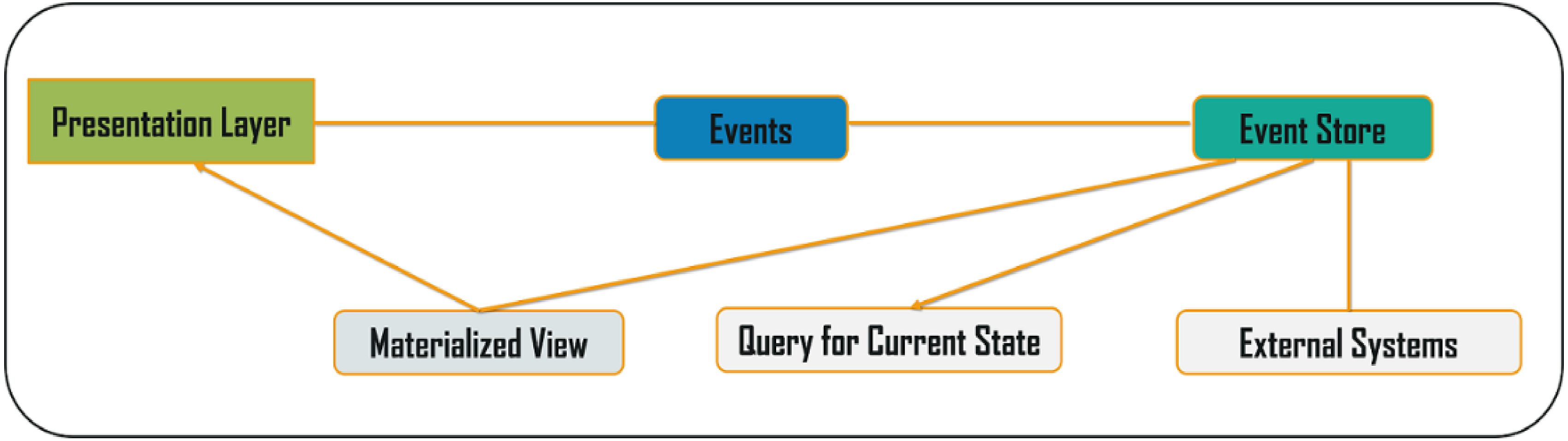
4. Asynchronous Messaging Pattern



5. Database or Shared Data Pattern

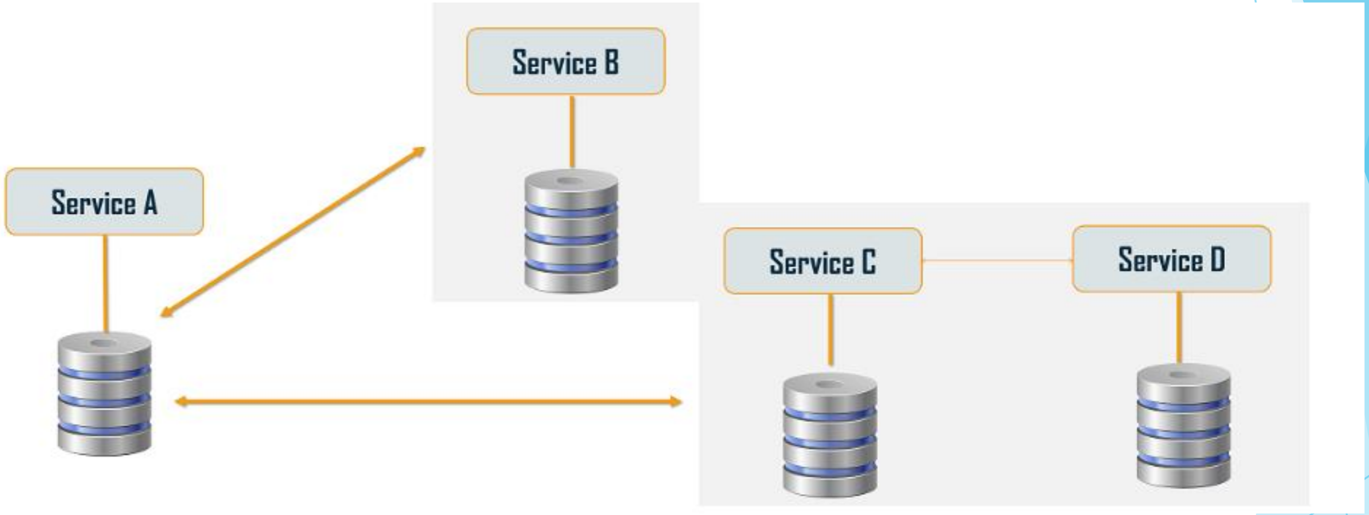


6. Event Sourcing Pattern



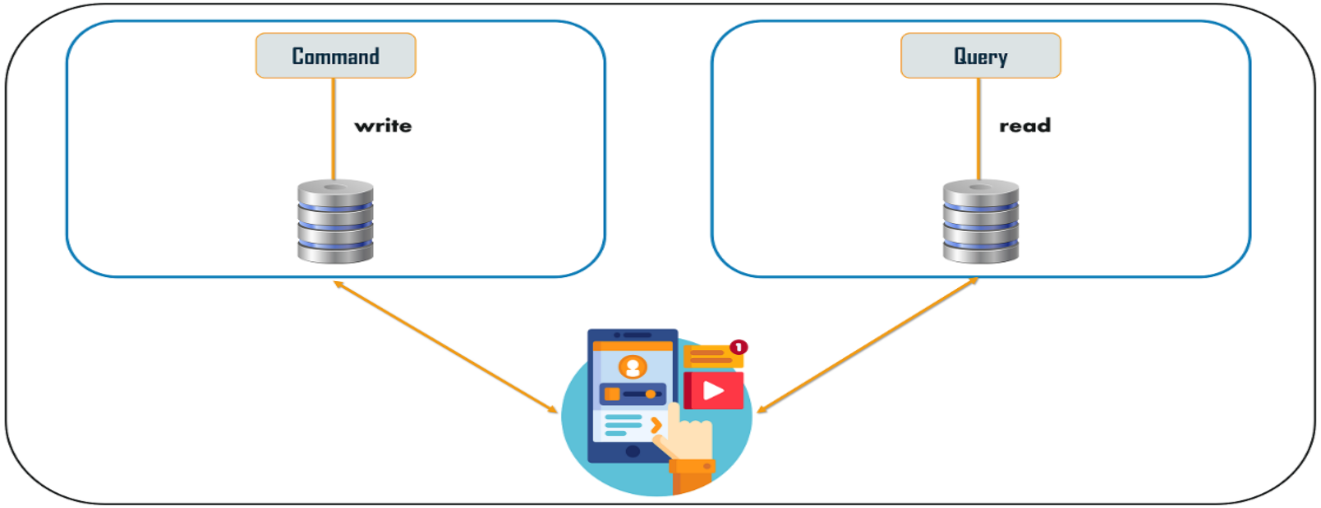
Used with Database pattern to sync individual DB or to provide eventual consistency

7. Branch Pattern



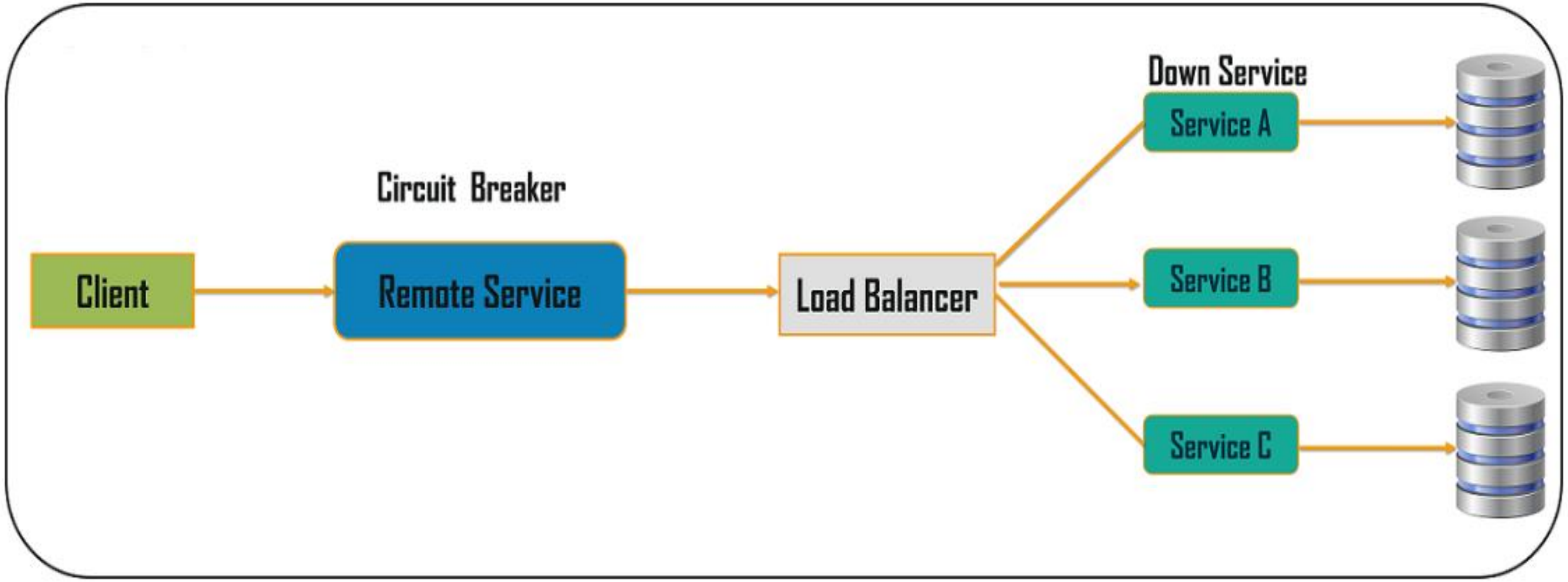
Extension of Aggregator pattern to ward of problem with chaining pattern

8. CQRS Pattern



Works with Database per service and event sourcing pattern when service need to query other database

9. Circuit Breaker Pattern



Tools to work with MSA

To start with microservices development, testing and deployment

1. SpringBoot – Java platform for developing microservices
2. Docker – containerization
3. Swarm – simple stack deployment using docker container
4. MongoDB –NoSQL Database
5. Redis – in memory database for caching
6. RabbitMQ – Messaging platform for managing queue and topics
7. Swagger – API documentation and testing

Thank you