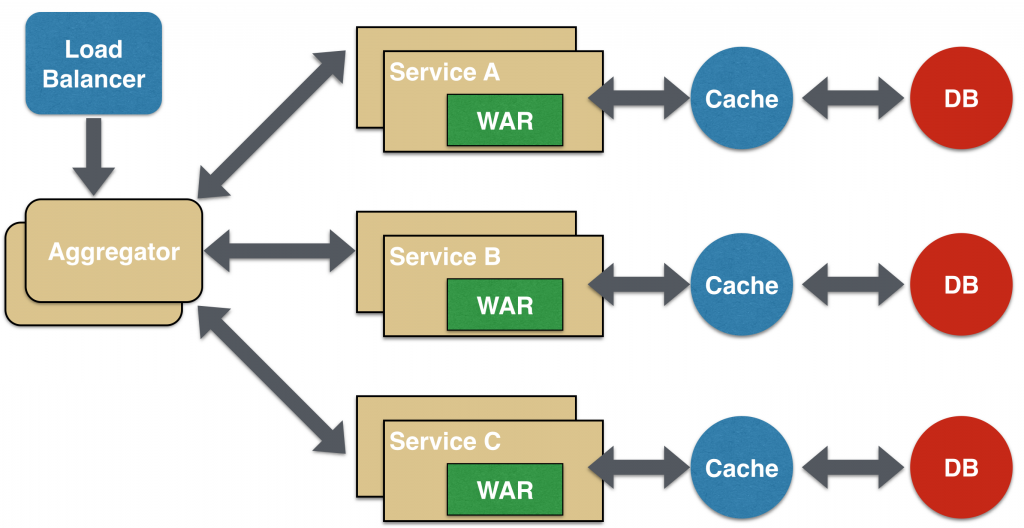
**Microservice Design Patterns – 2022**

Aggregator Microservice Design Pattern

**An aggregator would be a simple web page that invokes multiple services to achieve the functionality**.

Aggregator be a higher level composite microservice which can be consumed by other services.

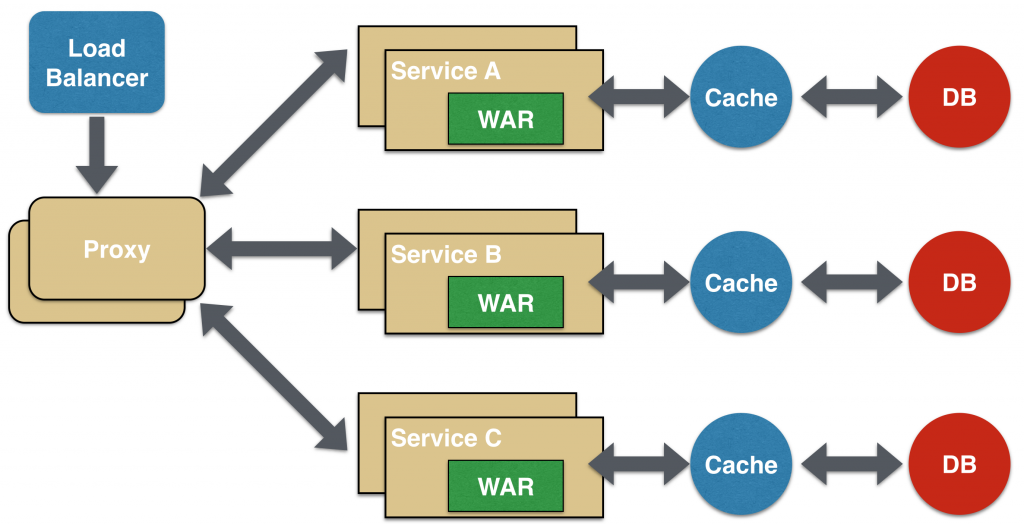
This design pattern follows the DRY principle. If there are multiple services that need to access Service A, B, and C, then its recommended to abstract that logic into a composite microservice and aggregate that logic into one service. An advantage of abstracting at this level is that the individual services, i.e. Service A, B, and C, and can evolve independently and the business need is still provided by the composite microservice.



Proxy Microservice Design Pattern

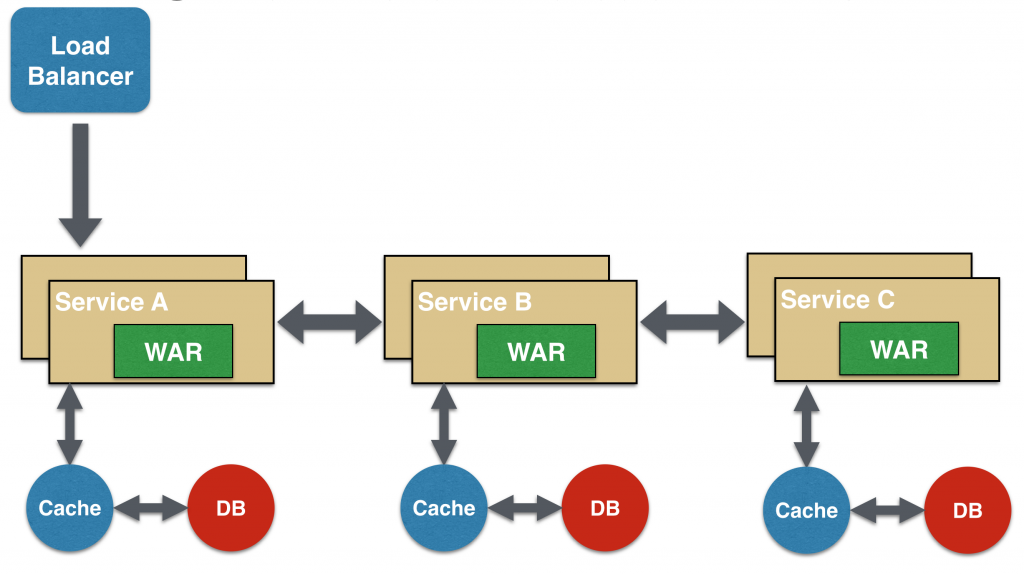
Proxy microservice design pattern is a variation of Aggregator.

The proxy **may be a dumb proxy in which case it just delegates the request to one of the services**. Alternatively, it may be a **smart proxy where some data transformation is applied** before the response is served to the client. A good example of this would be where the presentation layer to different devices can be encapsulated in the smart proxy.



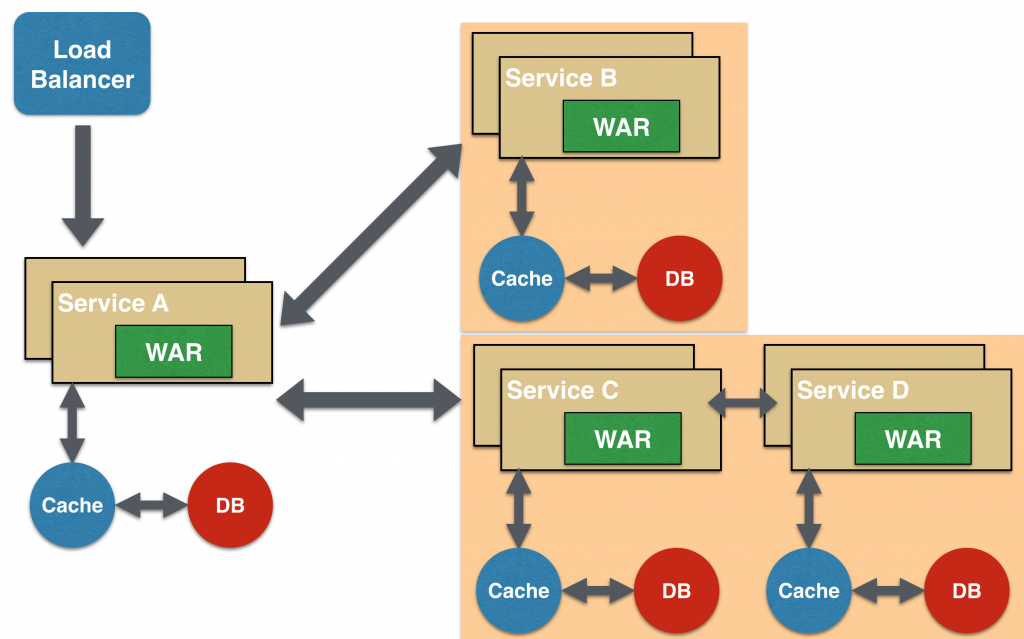
Chained Microservice Design Pattern

**Service A calls Service B and Service B calls Service C**. All the services are likely using a synchronous HTTP request/response messaging. The key part to remember is that the client is blocked until the complete chain of request/response. Another important aspect to understand here is to not make the chain too long.

[](http://blog.arungupta.me/wp-content/uploads/2015/04/microservices-chain.png)

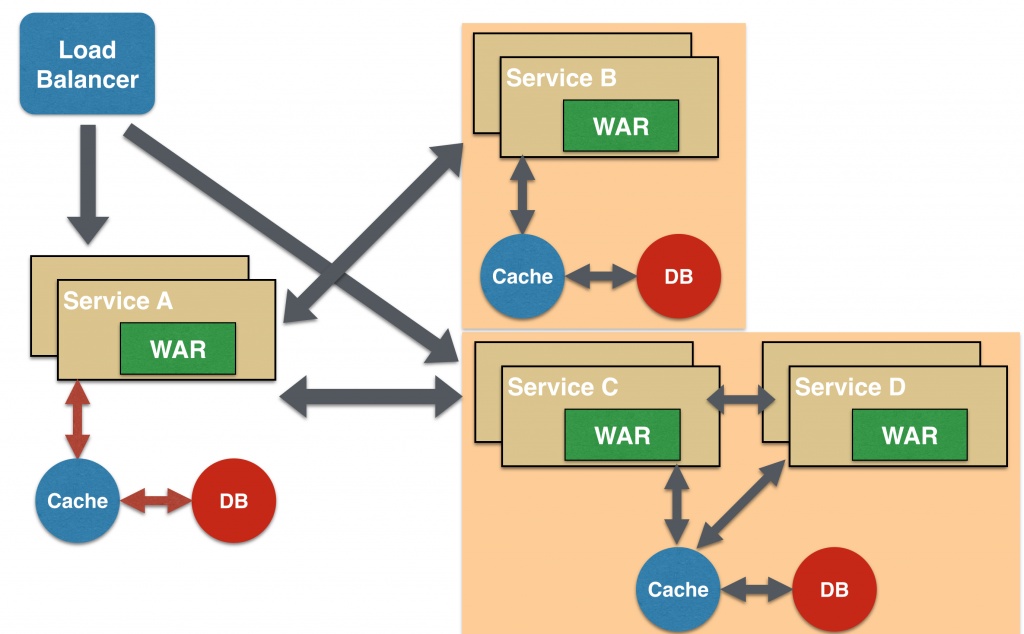
## Branch Microservice Design Pattern

Branch microservice design pattern extends Aggregator design pattern and allows simultaneous response processing from two. **Service A concurrently calls Service B and Service C**.

[](http://blog.arungupta.me/wp-content/uploads/2015/04/microservices-branch.png)

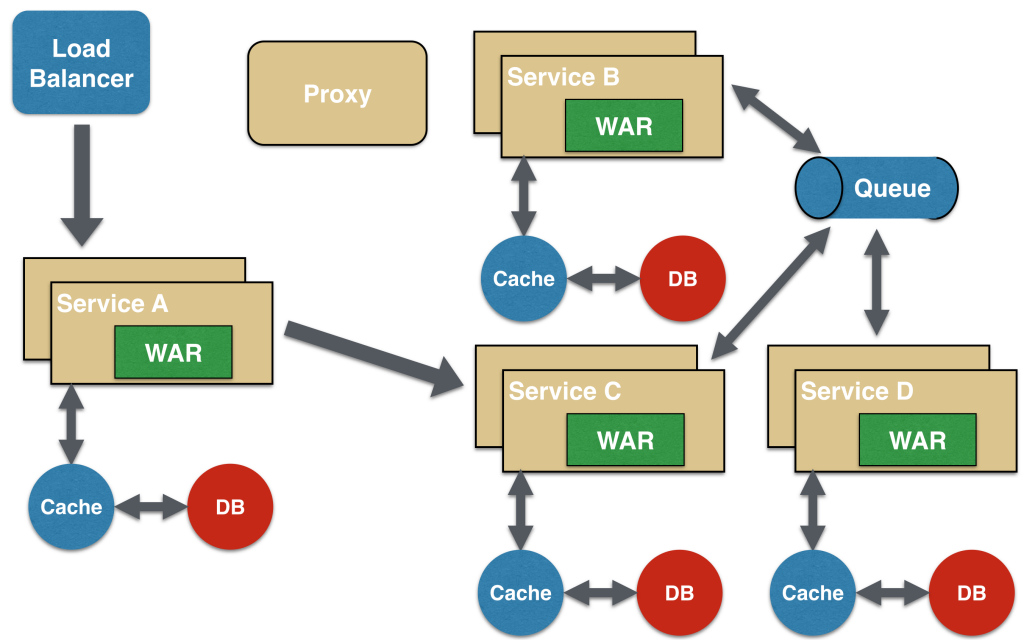
## Shared Data Microservice Design Pattern

In this design pattern, some microservices, likely in a chain, may share caching and database stores. This would only make sense if there is a strong coupling between the two services.

[](http://blog.arungupta.me/wp-content/uploads/2015/04/microservices-branch-shared-data.png)

## Asynchronous Messaging Microservice Design Pattern

In this design pattern, Service A may call Service C synchronously which is then communicating with Service B and D asynchronously using a shared message queue. You can use Kafka or RabbitMQ to achieve.

[](http://blog.arungupta.me/wp-content/uploads/2015/04/microservices-async-messaging.png)