Microservices Software Architecture: Patterns and Techniques

Thursday, June 6, 2019 12:55 PM

Microservice Architecture:

- Based on services and their intercommunication
- The system is structured as a collection of interconnected services
- Each service performs a specific function/task (**fine grained**), and is **lightweight**

Advantages:

- Low coupling
- Improves modularity
- Promotes parallel development
- Promotes scalability

Drawbacks:

- Infrastructure costs are usually higher
- Integration testing complexity
- Service management and deployment
- Nanoservice anti pattern (service is **too** fine grained)

*Note: GUI-based applications may not be suitable candidates for microservice architecture especially if these require a lot of data exchange between services

Assignment 1:

A bank is looking to develop a system for its new web platform that will be a long term project , and will therefore need to be scalable to support future growth

Yes, this would be a suitable application of microservices architecture since it's a short term project , and a well designed microservices based system will support good scalability.

An e-commerce website requires a small application that will be used for a short period to support a temporary promotion scheme that they will have.

This scenario is contrary to that of the previous question, since the application is not complex and will only be used for a short term it can be developed in less development intensive software architectures such as tiered architecture.

A blogging website with complex functionality that is currently built on a monolith architecture, is looking into doing a quick refactoring to improve the code quality .

Although the blogging website has complex functionality and is a project for the long term, the company only has time / resources for a 'quick' refactoring and for that reason microservices architecture is not applicable here as this would take a significant amount of time.

From <https://fidelity.udemy.com/microservices-software-architecture-patterns-and-techniques/learn/practice/1038064/instructor-solution#overview>

Why do many microservice projects fail?

- Lack of:
  - Planning
  - Knowledge
  - Skills
  - Time
  - Underestimation in the additional development, deployment, monitoring, and configuration of microservices and their inter-process communication infrastructure

How to prevent your projection from failing?

- Determine applicability
- Prioritize automation
- Have a clear plan
- Avoid common pitfalls (prior research)

Microservice Template:

Why is it important?

- Significant amount of time setting up
- Similar code for each microservice setup
- Not as effective in a monolithic/multi-tiered architecture compared to a microservice architecture w/ 10s-100s of microservices adding up.

What should the template contain?

- Cross cutting concerns:
    - Logging
    - Metrics
    - Connection setup and configuration to databases and message brokers
- Project structure

Code Repository Setup:

- Mono vs. Discrete

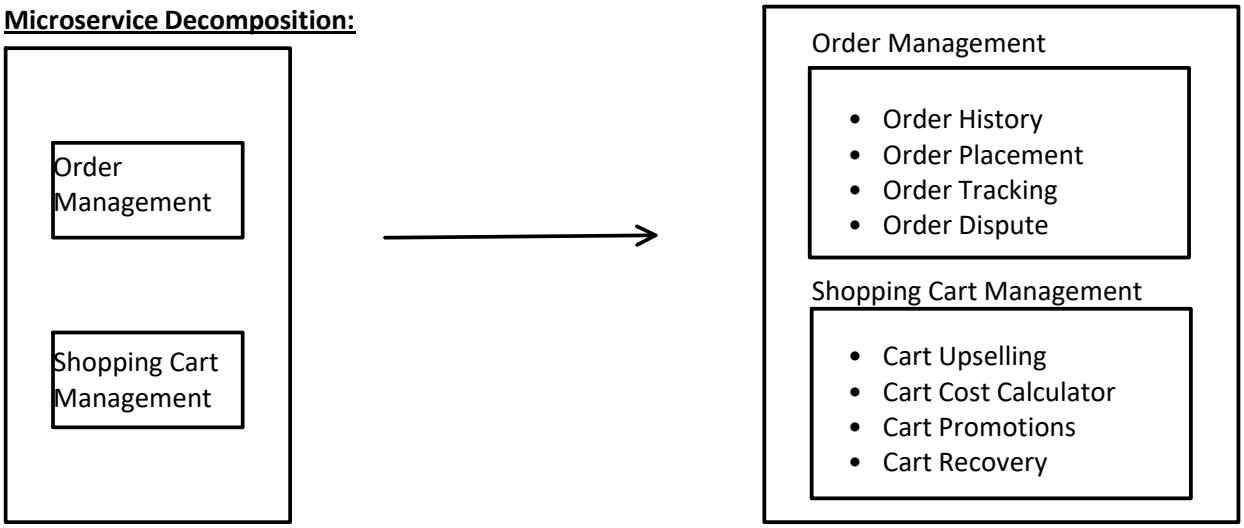
Mono Repo:

Pros	Cons
Easier to keep input/output contracts in sync	Different teams working in the same repo can break the build, disrupting CI/CD for other teams
Can version the entire repo with a build number	Easier to create tight coupling
	Long build times, large code repo to download

Discrete Repo:

Pros	Cons
Different teams can 'own' different repositories	Contract versioning becomes more complex
Scope of a single repo is more clear	Unless managed properly, discrete repositories can easily become monoliths
	More upfront cost in setting up repos and CI/CD pipeline

Microservice Decomposition:



Assignment 2:

A hospital monitoring system, where sensors monitor patients' vitals and raise an alert if these do not match the patient's healthy range of vitals' statistics.

The main microservices you should include are:

- Patient Microservice
- Vitals Monitoring Microservice
- Alert Microservice

An airline booking system where customers create an account and book flights , and allocates seating arrangements to customers.

The main microservices you should include are:

- Customer microservice
- Seating microservice
- Flight booking microservice

A blogging site where users can post articles on their blogs, and other users can comment on these articles.

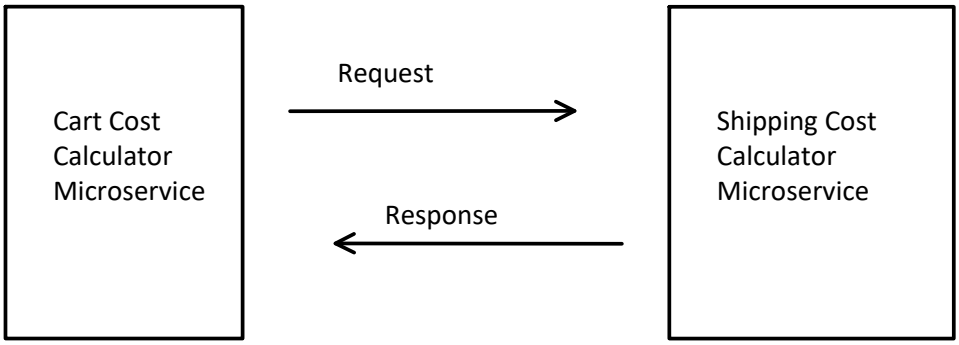
The main microservices you should include are:

- User microservice
- Blog microservice
- Article microservice
- Comment microservice

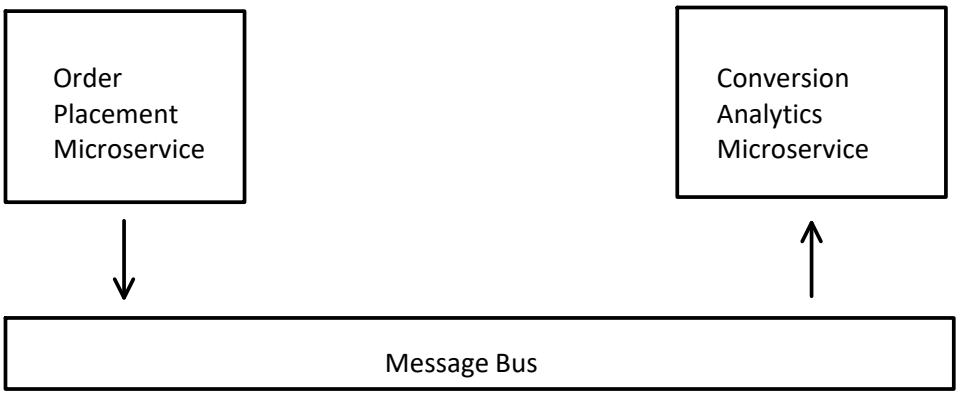
From <https://fidelity.udemy.com/microservices-software-architecture-patterns-and-techniques/learn/practice/1038066/instructor-solution#overview>

Inter-Service Communication:

Remote Procedure Invocation:



Asynchronous Message Based Communication:



Custom or Domain Specific Protocols:

Assignment 3:

In a payment gateway system, when we receive a request to withdraw funds from a customer's account, we check if funds are available and process the request if funds are available. An email is queued to be sent to the customer notifying them of the request.

Checking if the funds are available is a synchronous flow, as this must be completed before we can process the withdrawal request.

Since the email is queued (i.e. the system doesn't wait for this to be sent before responding to the payment request) this is an example of an asynchronous flow.

In an e-commerce system, when a customer places an order successfully we show a successful response to the customer on screen. Data related to the order is also published to analytics components for business insights.

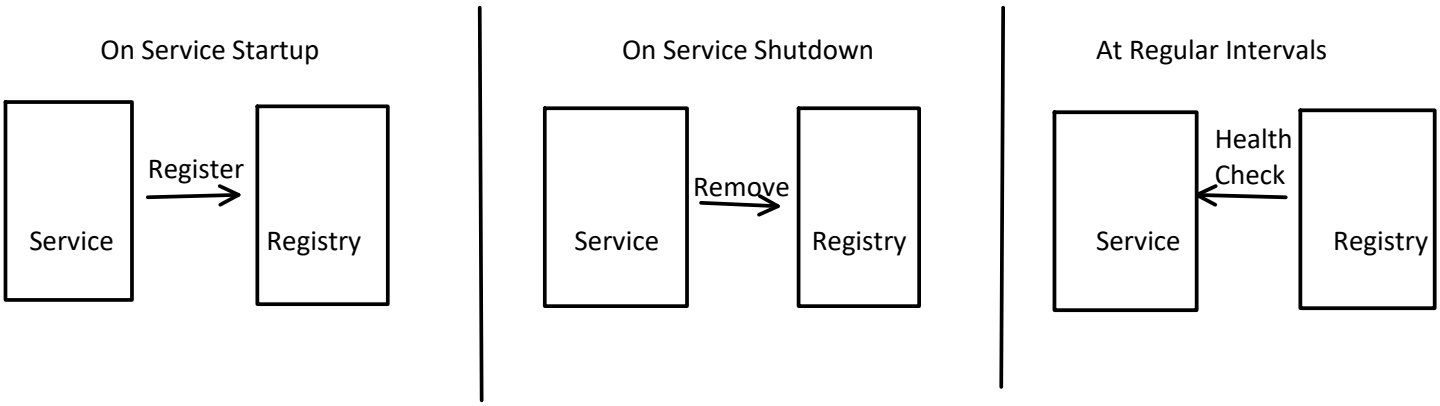
The response being shown to the customer is a synchronous process and the publishing of the order data to analytics components is also a synchronous process.

However, the analytics components processing this order data for analytics insights is done asynchronously.

From <https://fidelity.udemy.com/microservices-software-architecture-patterns-and-techniques/learn/practice/1038072/instructor-solution#overview>

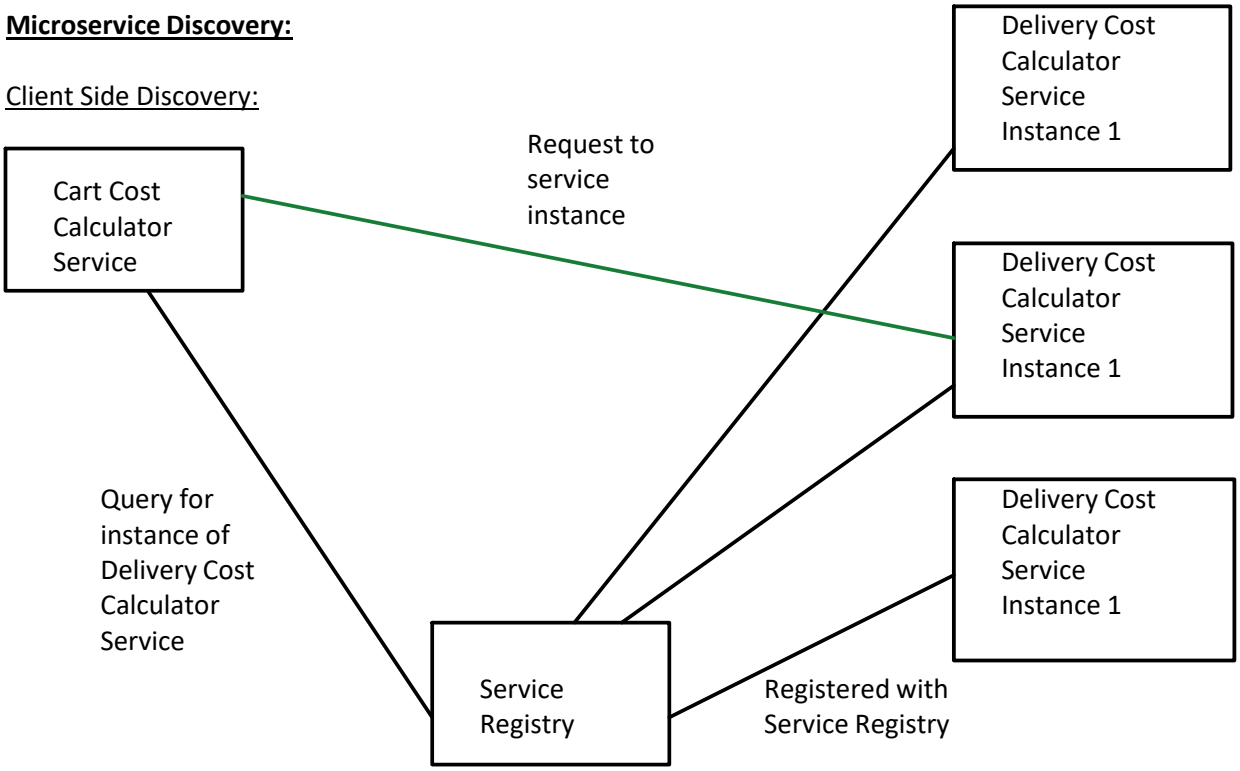
Microservice Registry:

How it works:

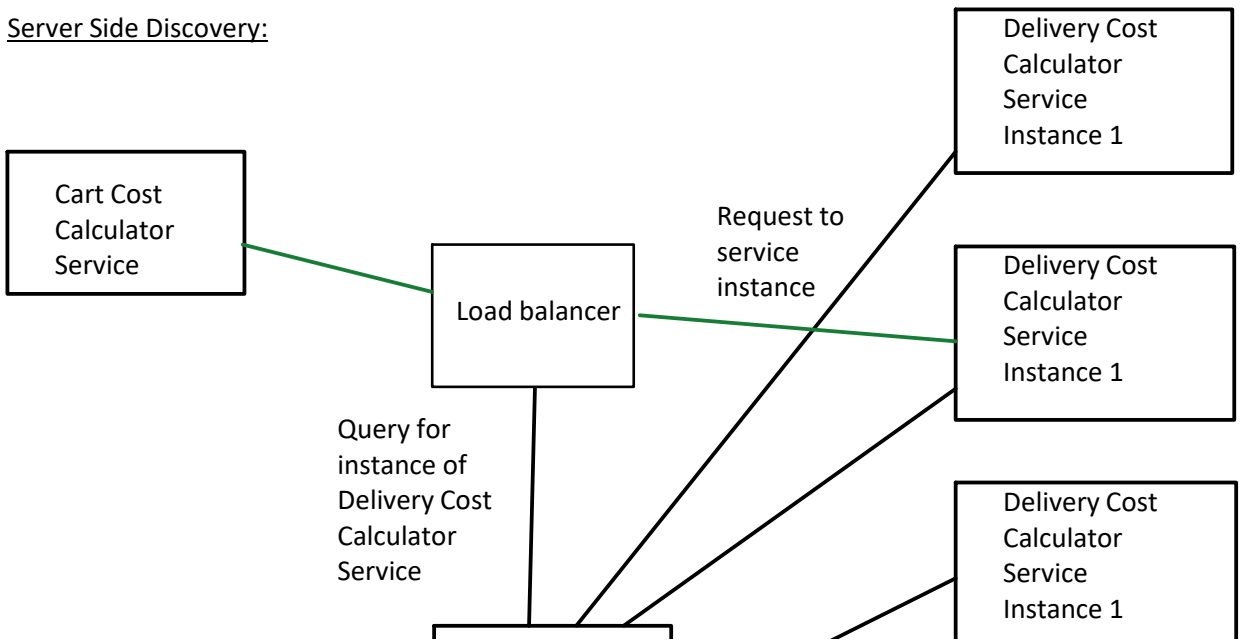


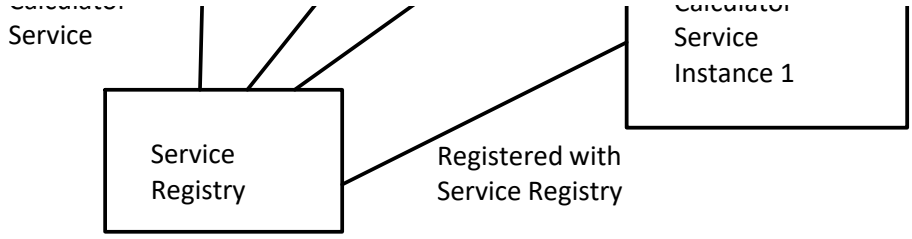
Microservice Discovery:

Client Side Discovery:



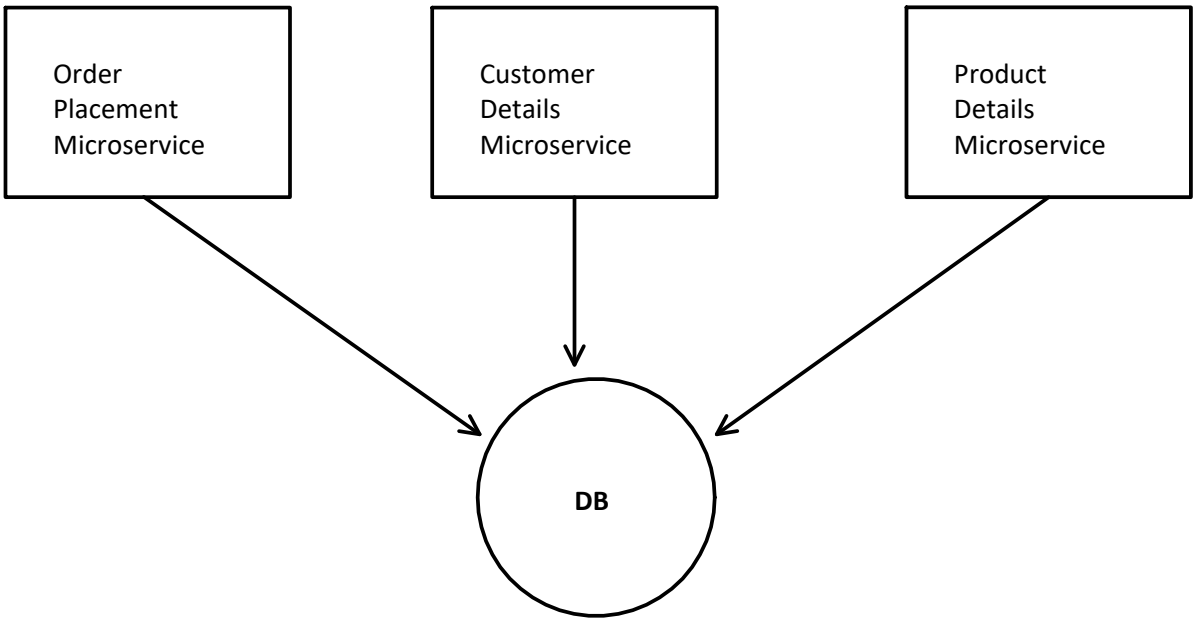
Server Side Discovery:



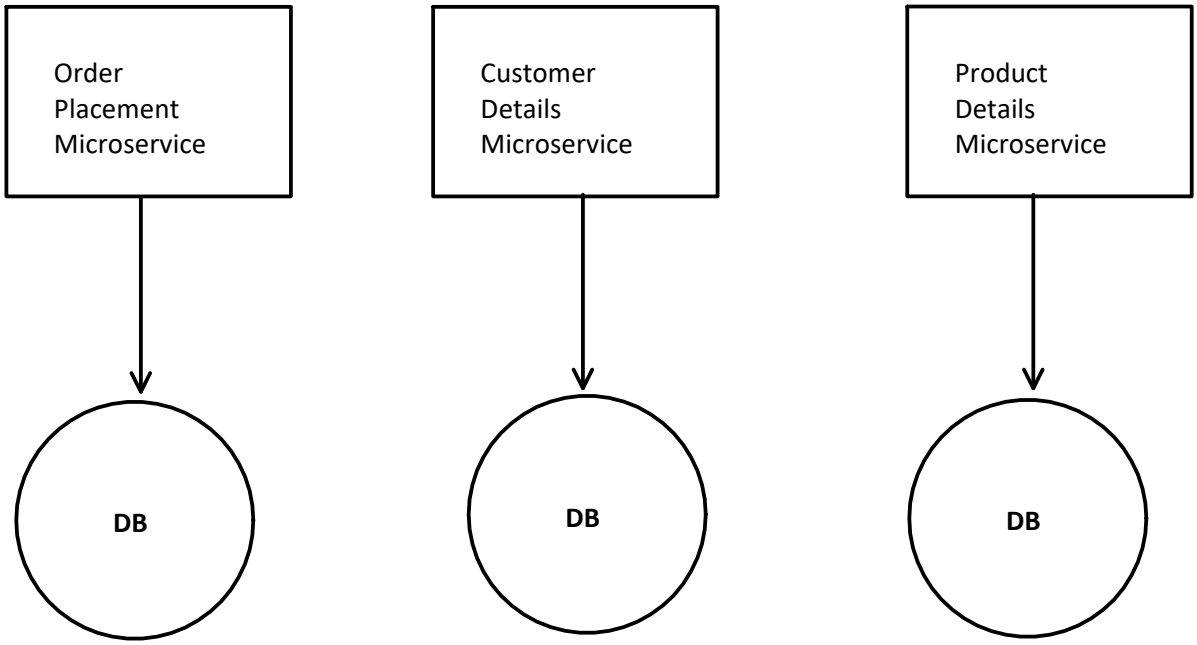


Databases:

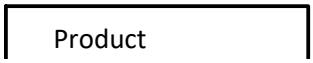
Shared Database:

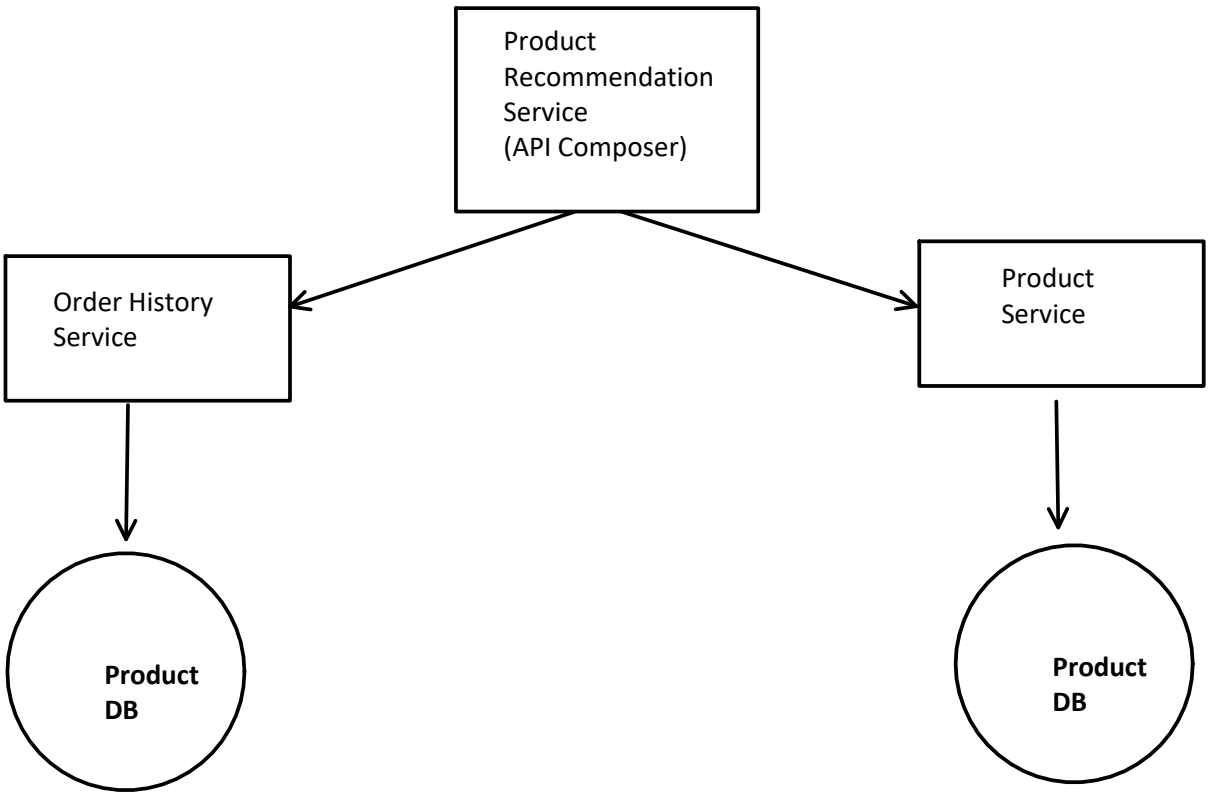


Database per service:

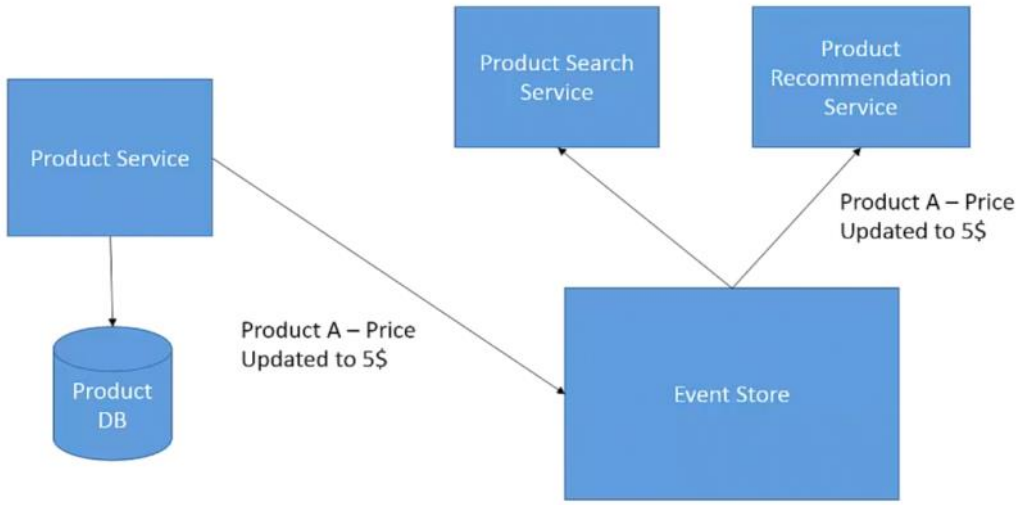


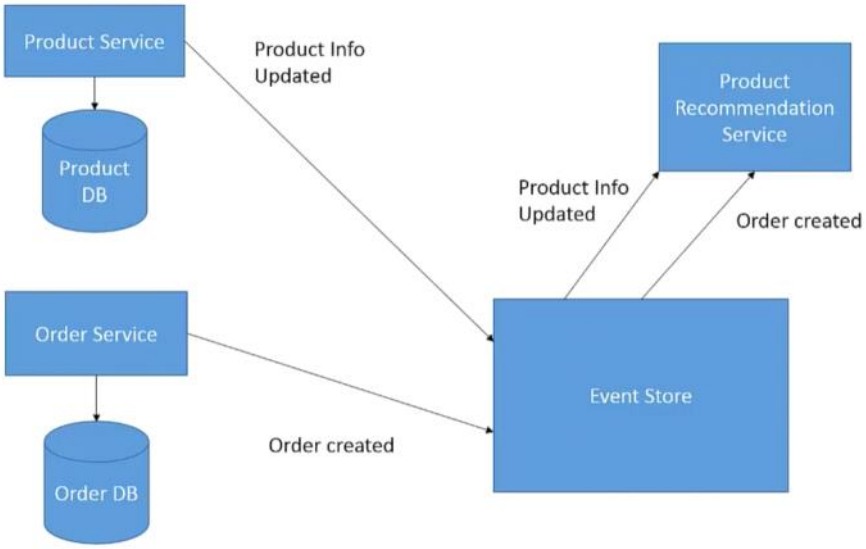
API Composition:





Event Sourcing:

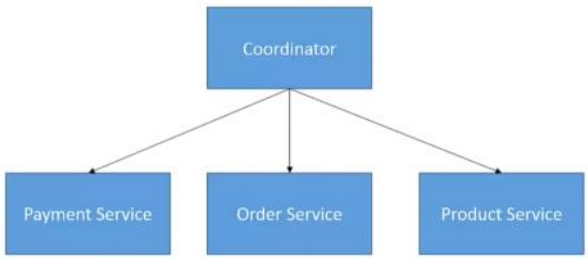




Two Phase Commit:

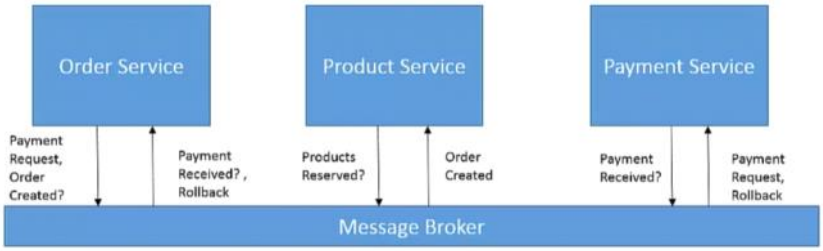
Phase 1 : Commit Request
- Coordinator Sends a query to commit message
- Services execute the transaction but do not commit
- Reply Yes/No depending on if they were successful

Phase 2 : Commit
- If all services replied yes:
  - coordinator sends a commit message
  - Services commit the transaction
  - Reply with an acknowledgement
- If at least one service replied with no:
  - Coordinator sends a rollback message
  - Services rollback the transaction

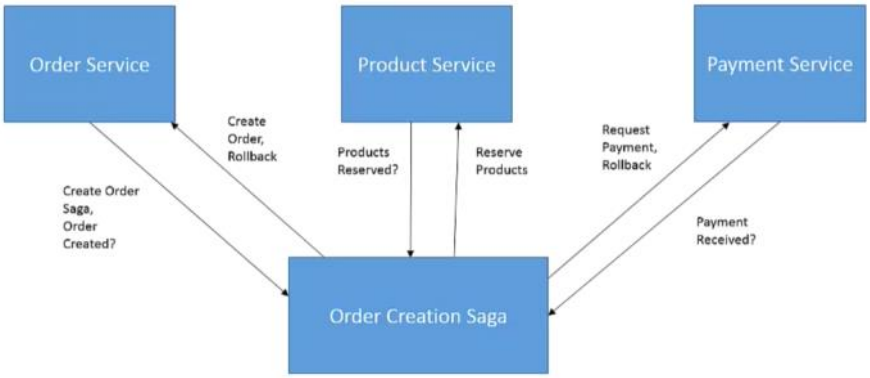


Saga:

Choreography-based sagas:



Orchestrator-based sagas:



Assignment 4:

In an e-commerce system, the order information needs to be combined with the list of warehouses to identify the closest location. The list of warehouses is short and can be cached for long durations as it does not change often.

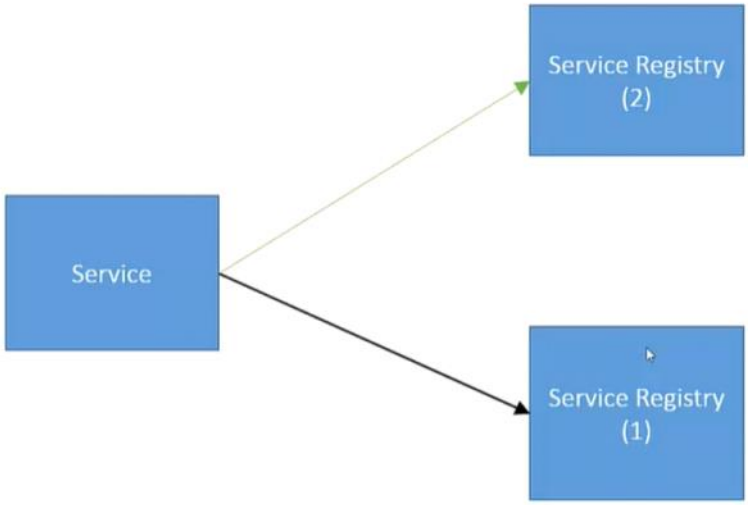
The API composition pattern will fit this scenario well, as the quantity of data being requested from the order microservice and warehouse microservice is small and we can also take advantage of in memory caching for additional optimization.

In an e-commerce store, a service needs to combine data for the hourly product and order data. The business is large , with a high number of orders per hour and a large number of products in its database.

Since the volume of data that needs to be loaded by the service is potentially very large, the API composition pattern is not a suitable candidate. A suitable solution would be to use event sourcing, and have this service subscribe to both product and order related events in order to keep an updated view of the required data whilst avoiding bulk data transfers that are both time consuming and also resource inefficient.

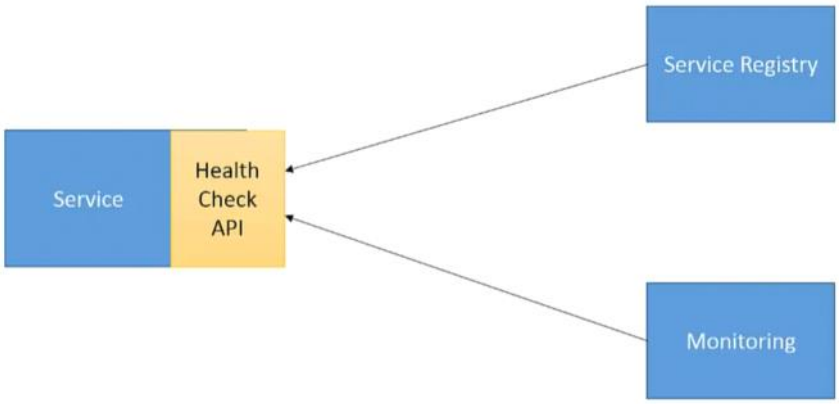
From <https://fidelity.udemy.com/microservices-software-architecture-patterns-and-techniques/learn/practice/1038068/instructor-solution#overview>

Failover Mechanism:

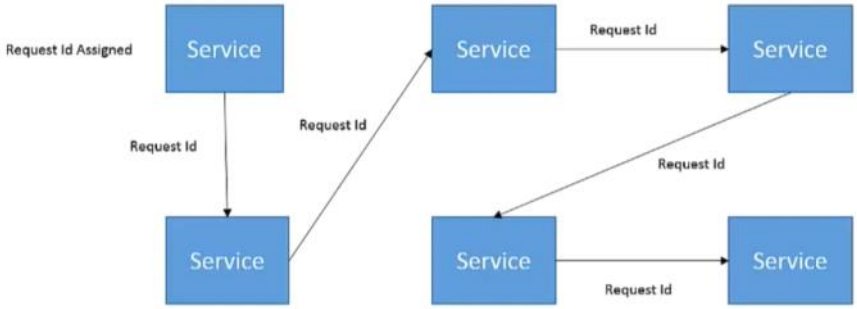


Circuit breaker:

Health Checking API:



Logging Techniques:



- SPLUNK, AWS Cloud!

Assignment 5:

In a payment gateway, a request to withdraw funds has failed unexpectedly. This was reported by the customer , we have the following data available:

- Customer ID, Amount requested, Approximate time of the request

First off, we will rely on the logging aggregate technology using the customer ID, amount and time as filters to query the logs in order to identify the entry point of the withdrawal request into our system.

Once this has been identified, we can use the request identifier related to this request to trace its flow across microservices to see where it is failing.

In a blogging website, a service our error logs indicate that an article creation microservice is failing regularly . You are required to help identify potential causes.

One approach to this scenario would be to query a list of all error logs, use their request identifiers to obtain the original requests as they entered the system and with this data in hand examine the request payloads for common denominators that could immediately pinpoint to the root cause. Failing that, we could get some of these sample payloads that caused the error and replay them to replicate.

In an e-commerce system, a customer is not able to register a new account. We have attempted to trace their details in our aggregate logs however there was no trace of this request.

This was partially a trick question, however one that will help train your logical reasoning for practical situations - if there is absolutely no record of the request in our logs then one of the following is the most probable cause and we'll need to investigate each accordingly:

- The request did not make it to our system at all - there may have been some network error or UI error on the customer's browser that prevented the request from being sent

- The logging aggregation technology may not be picking up all logs, this is unfortunately a possibility especially if resources are under heavy usage and hence monitoring on the logging aggregate is also necessary. It would also be a good idea to ensure that logs are being kept for adequate periods and that they are not being discarded due to storage capacity limitations

From <https://fidelity.udemy.com/microservices-software-architecture-patterns-and-techniques/learn/practice/1038070/instructor-solution#overview>