critique [kri-teek]

noun

1. an article or essay criticizing a literary or other work; detailed evaluation; review.
2. a criticism or critical comment on some problem, subject, etc.

verb (used with object), cri·tiqued, cri·ti·quing.

4. to review or analyze critically.

REST API design

Use HTTP verbs

- Easy to understand and a lot of shared knowledge… but what about cases that don’t map so well to CRUD operations?
- Real life example: restart a server
- Another example: “soft delete” an item (can be restored)

Any guesses?

Source: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

Rest API design

Use HTTP verbs

- Both cases were subjects of debate
- Restarting a server: POST /servers/{id}/restart
- "Soft delete/restore":
  DELETE /items/{id}
  GET /deleted-items
  PUT /deleted-items/{id} "{...deleted: false}"
  database magic happens here to move an item under /items

Source: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

Rest API design

Return appropriate HTTP codes and errors

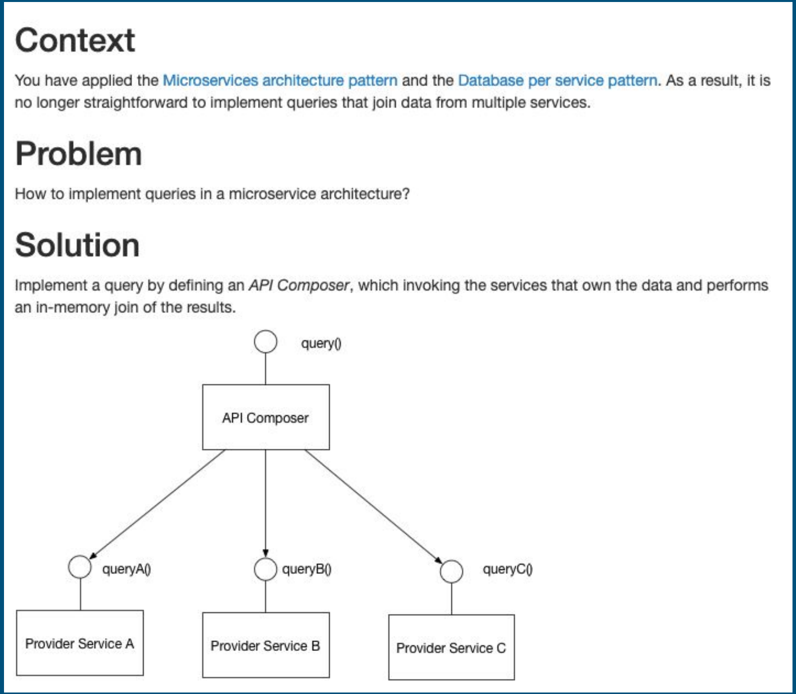
- KISS is very important here: don’t use every single code known to man. Error 500 with a descriptive error message is better than an obscure special error code
- Real life example: to indicate that server is restarting, developer wants to use 423 Locked (I’ll let you find out what it is…)

Source: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/

Rest API design

- In my opinion, most important API design best practice: **be consistent**
- Things like naming fields (e.g. `country` vs `country_name`), pagination, sorting, etc. should work consistently across different APIs
- Especially important for **microservices** where different people are working on different things

API Composition



Context

You have applied the Microservices architecture pattern and the Database per service pattern. As a result, it is no longer straightforward to implement queries that join data from multiple services.

Problem

How to implement queries in a microservice architecture?

Solution

Implement a query by defining an *API Composer*, which invoking the services that own the data and performs an in-memory join of the results.

Context

Implement queries that join data from multiple services.

Solution: API Composer

Invoking the services that own the data and performs an in-memory join of the results.

Source: https://microservices.io/patterns/data/api-composition.html?source=post_page

But which component is responsible for API Composition?

Option 1

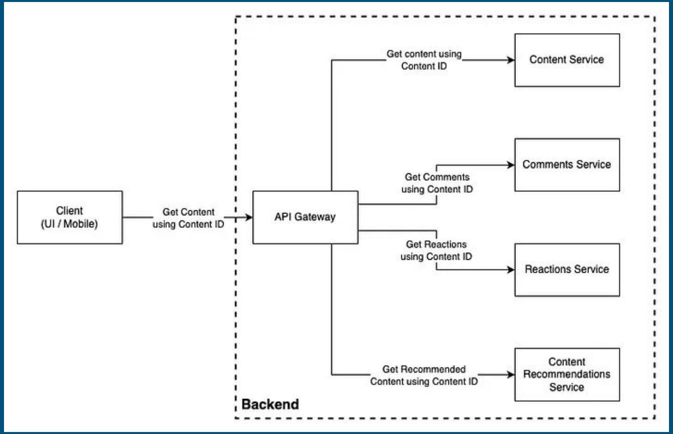
Have a new component service for API Composition

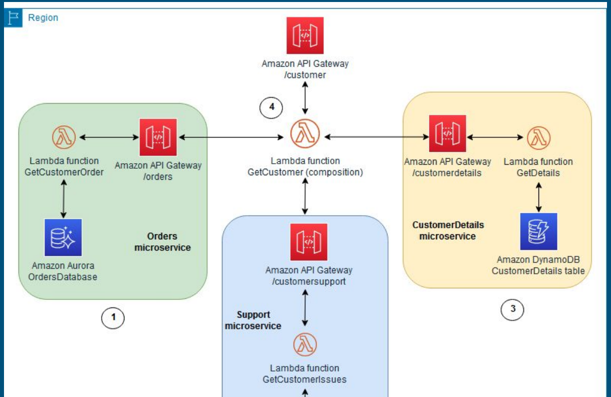
Pros:

- Better separation of concerns
- Flexibility, Centralized Management

Cons:

- Increased Complexity
- Potential Performance & Communication Overhead
- Potential for Inconsistency


Client (UI / Mobile)
Get Content using Content ID
API Gateway
Get content using Content ID
Content Service
Get Comments using Content ID
Comments Service
Get Reactions using Content ID
Reactions Service
Get Recommended Content using Content ID
Content Recommendations Service
Backend


Region
Amazon API Gateway /customer
4
Lambda function GetCustomer (composition)
Lambda function GetCustomerOrder
Amazon API Gateway /orders
Amazon Aurora OrdersDatabase
Orders microservice
1
Amazon API Gateway /customersupport
Support microservice
Lambda function GetCustomerIssues
Amazon API Gateway /customerdetails
Lambda function GetDetails
CustomerDetails microservice
Amazon DynamoDB CustomerDetails table
3

Which component is responsible for API Composition?

Option 2

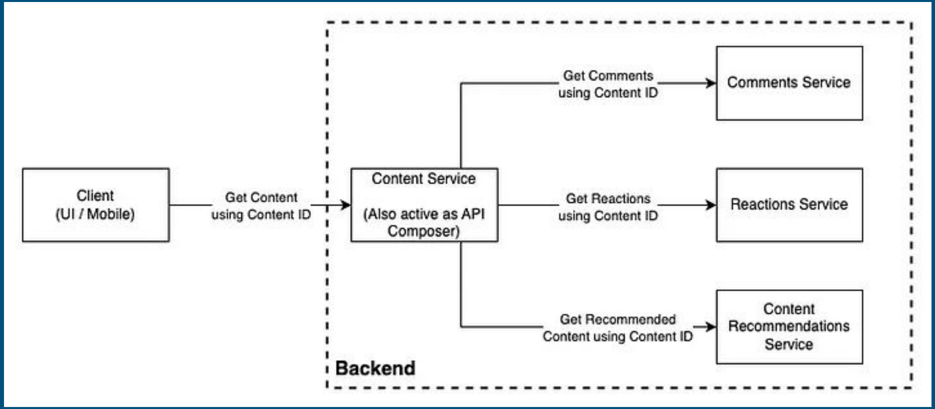
One of the Domain services owns this responsibility.

Pros

- Quick to set-up
- Utilizing resources
- Smaller blast radius

Cons

- Introduce domain coupling
- Resource intensive tasks (aggregation of large dataset) might affect service availability


Client (UI / Mobile)
Get Content using Content ID
Content Service (Also active as API Composer)
Get Comments using Content ID
Comments Service
Get Reactions using Content ID
Reactions Service
Get Recommended Content using Content ID
Content Recommendations Service
Backend

Source: https://microservices.io/patterns/data/api-composition.html?source=post_page

API Composition

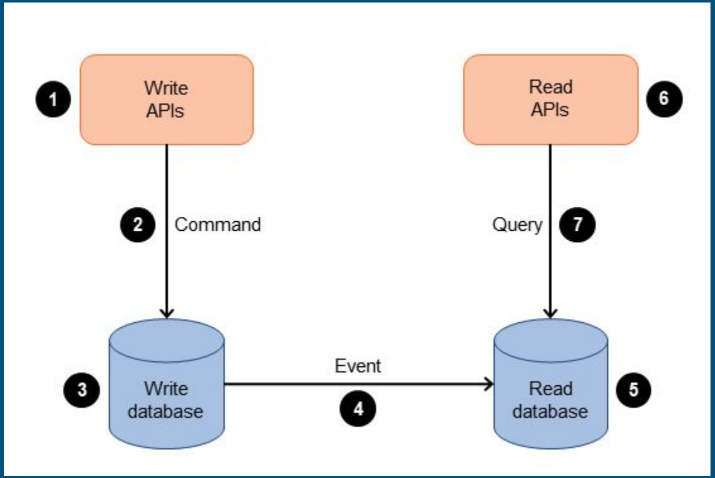
Pros

- Simple way to query data in a microservice architecture.

Cons

- When dealing with large amounts of data (to join, transformation, etc), which technically could be doable but may not be efficient and would lead to high latency.

CQRS pattern



Source: https://microservices.io/patterns/data/api-composition.html?source=post_page