**API Composition Design Pattern**

**Introduction**

* When dealing with 2 or more microservice, it’s often a problem to implement queries that join data from multiple services.
* For instance, we have a catalog and inventory microservice. The catalog provides the API in managing the catalog models such as product while inventory handles the stock of a certain product. In the inventory service, product data such as code is not maintained.
* Now how do we generate a list of product stock with product code and quantity?
* One answer is the API Composition Design Pattern.

**Advantages:**

* This pattern is easy to use as we are just calling multiple microservices to get the data.
* Clients can get all the data in a single hit
* Microservices can focus on business logic

**Drawbacks:**

* Increased overhead - More DB requests, more network traffic...

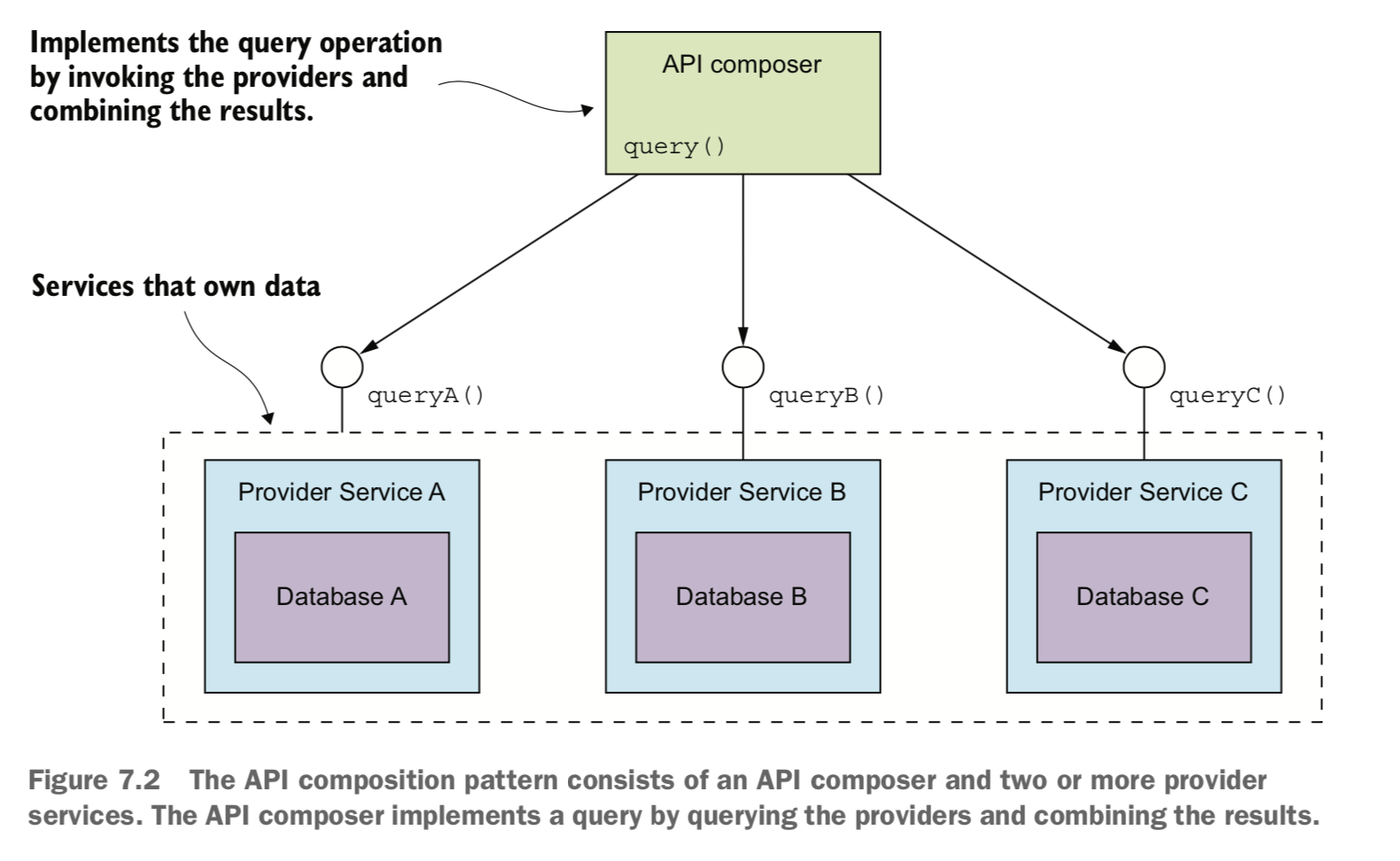
Ex: For example, if we have a list of 10 product stocks with different product id, that means we have to query the catalog service to get the product code 10x.

* Risk of reduced availability - The more services you're connected to, the more likely one can take you down
* Lack of transactional data consistency
* To avoid memory problems when loading a large dataset, always implement paging.

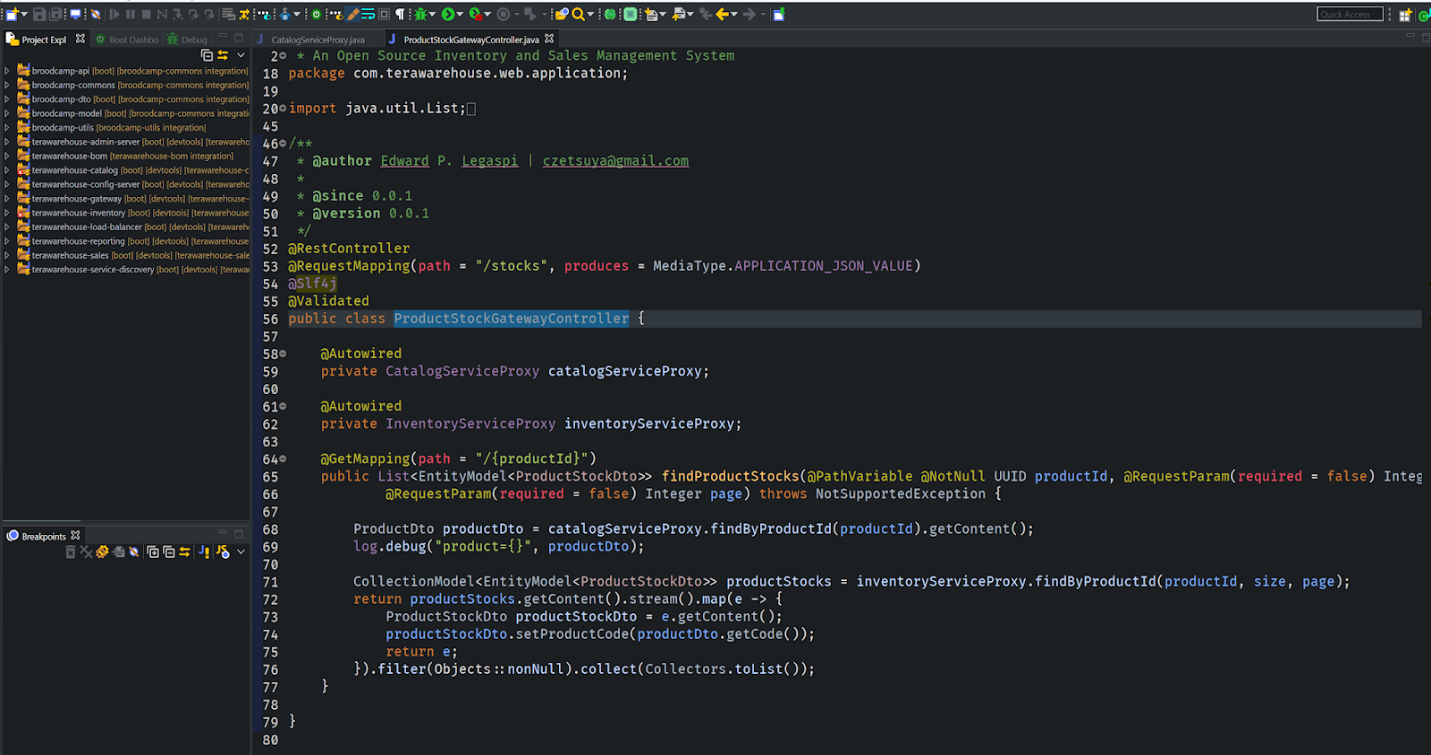
**How can we create the view database?**

* If you are using the same database type such as Postgresql then you can simply join 2 tables from a different database.
* Use Apache Nifi to populate a reporting table on a database whenever the source tables from different databases are modified.
* Use a messaging queue such as Apache Kafka to update multiple databases.

### Catalog and Inventory Microservice Architecture



### The API Composer Class



**References:**

* https://livebook.manning.com/book/microservices-patterns/chapter-7

**Codes are available at Github**

* <https://github.com/terawarehouse>
* <https://github.com/broodcamp>