# Legacy Adapter Microservice

You are designing an application using a *Microservices Architecture*. You want it to incorporate existing services (e.g. SOAP, JMS, or mainframe-based services) but their APIs do not use the RESTful or queue-based approach that is consistent with the microservices architecture.

**How do you handle translation of existing service implementations into good microservices APIs?**

There are several things that make this rearchitecture challenging:

- If you have existing services that run well enough as is, you don't want to rewrite them, you want to reuse them instead.
- Changing the APIs of existing services can break existing clients.
- If an existing service doesn't have a good microservice API and that's inconvenient to change, it's also inconvenient for each client microservice to use that existing service via its legacy API. The components in a microservices architecture should integrate via good microservice APIs.

**Build simple *Legacy Adapter Microservice* that converts the existing service's non-microservice API to an API that client microservies will expect. A Legacy Adapter Microservice is a type of *Adapter Microservice* that follows the Adapter pattern from Gamma in that it converts one API to another.**

In many cases, it's a straightforward exercise to convert a function-based interface (for instance one built using SOAP) into a business-concept-based interface. In many ways this can be thought of as moving from a verb-based (functional) approach to a noun-based (entity) approach. Often, the functions exposed in a SOAP endpoint correspond one-to-one to CRUD (create, read, update, delete) operations on a single business object type, and therefore can map easily to a REST interface where the URL represents the type of object, and the GET, POST, PUT and DELETE methods correspond to read, create, update and delete for that type respectively. These operations would then simply send the corresponding SOAP messages to the existing SOAP endpoint and then translate the XML datatypes from the corresponding SOAP operations to JSON datatypes for the new REST interface.

However, there are cases where this mapping falls down that must also be handled. An example of this case is a bank transfer - which is an operation between two bank accounts that does not correspond to a CRUD operation on either account. In this case, the simplest solution is often the same one taken in relational databases for the corresponding problem - you create a new type that represents the transfer itself - analogous to creating a relationship table in a relational database.

So in this case you would build a new REST interface whose URL represents the Transfer type, with a POST operation corresponding to performing a transfer, a GET operation that could return information about a transfer, and a DELETE operation that is a (hopefully well-controlled) undo.

Legacy Adapter Microservices are a special type of microservice in that they are often transient solutions, which only last until the underlying existing services can be replaced by a natively implemented microservice. The Command Query Responsibility Separation pattern is often a part of this transformation in that a Legacy Adapter Microservice may only represent one side (usually the write side) of the process.

Due to their special nature of converting one interface to another, *Legacy Adapter Microservices* often use Results Caches in order to reduce the number of times they have to invoke the underlying SOA service that they convert.