**OAUTH 2**

An API itself is an interface. It’s the interface for clients that interact with the system. Clients should only know about the interface and nothing about its implementation. There can be more than one implementation for a given interface; the clients written against the interface can switch between implementations seamlessly and painlessly

The client application and the API implementation can be running in the same process or in different processes.

If they’re running in the same process, then the call between the client and the API is a native one—if not, it’s a

remote call.

APIs can also be exposed for remote access. To invoke an API remotely, you need to have a protocol defined for interprocess communication

Java RMI, CORBA, .NET Remoting, SOAP, and REST (over HTTP) are some protocols that facilitate interprocess communication

SOAP-based web services provide a way to build and invoke a hosted API in a language- and platform-neutral

manner. A message from one end to the other is passed as an XML payload. SOAP is very structured and is backed by a large number of specifications. The request/response protocol between the client and the server is defined in the

SOAP specification. The way you describe a SOAP service is defined in Web Services Description Language (WSDL).

The WS-Security, WS-Trust, and WS-Federation specifications describe how to secure a SOAP-based service. WS-

Policy provides a framework to build quality-of-service expressions around SOAP services. WS-SecurityPolicy

defines the security requirements of a SOAP service in a standard way, built on top of the WS-Policy framework.

The list goes on and on. Due to the nature of SOAP-based services, which are highly decoupled, standardized, and

governed based on policies, they’re the preferred ingredient to build a service-oriented architecture (SOA).

**API vs. Managed API**

Security, rate limiting (throttling), and monitoring are key aspects of a managed

business API. It also must have the ability to scale up and down for high availability based on traffic.

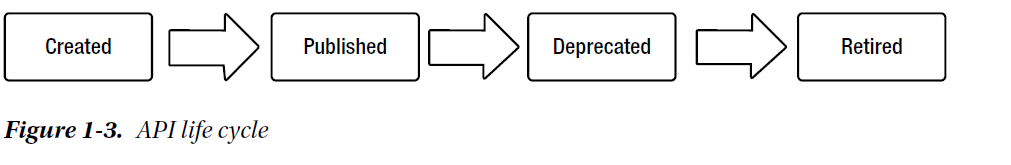
Life-cycle management is another key differentiator between a naked API and a managed API. A managed API has a life cycle from its creation to its retirement. A typical API life cycle might flow through Created, Published, Deprecated, and Retired stages

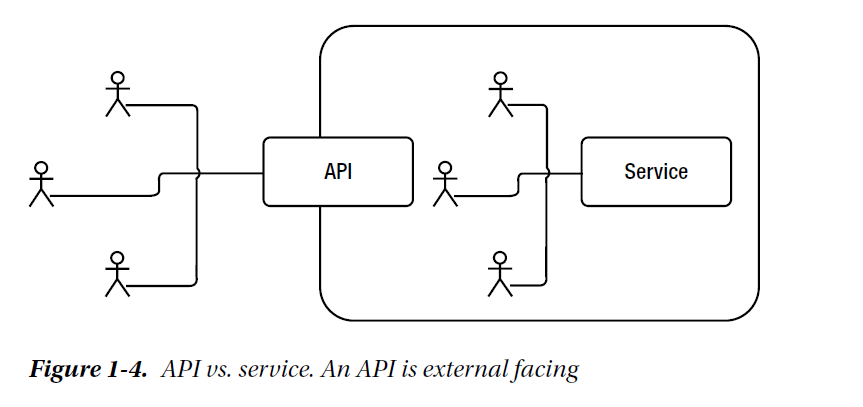
For example, to promote an API from Created to Published, you need to make sure the API is secured

properly, the documentation is ready, throttling rules are enforced, and so on. A naked business API, which only

worries about business functionalities, can be turned into a managed API by building these quality-of-service aspects

around it.





Discovering and Describing APIs

APIs are public facing, and that raises the need for the API description to be extremely useful and meaningful. At the same time, APIs need to be published somewhere to be discovered. A comprehensive API management platform needs to have at least three main components:

* a publisher,
* a store,
* a gateway.

The API publisher provides tooling support to create and publish APIs. When an API is created, it needs to be associated with API documentation and other related quality-of-service controls. Then it’s published into the API store and deployed into the API gateway