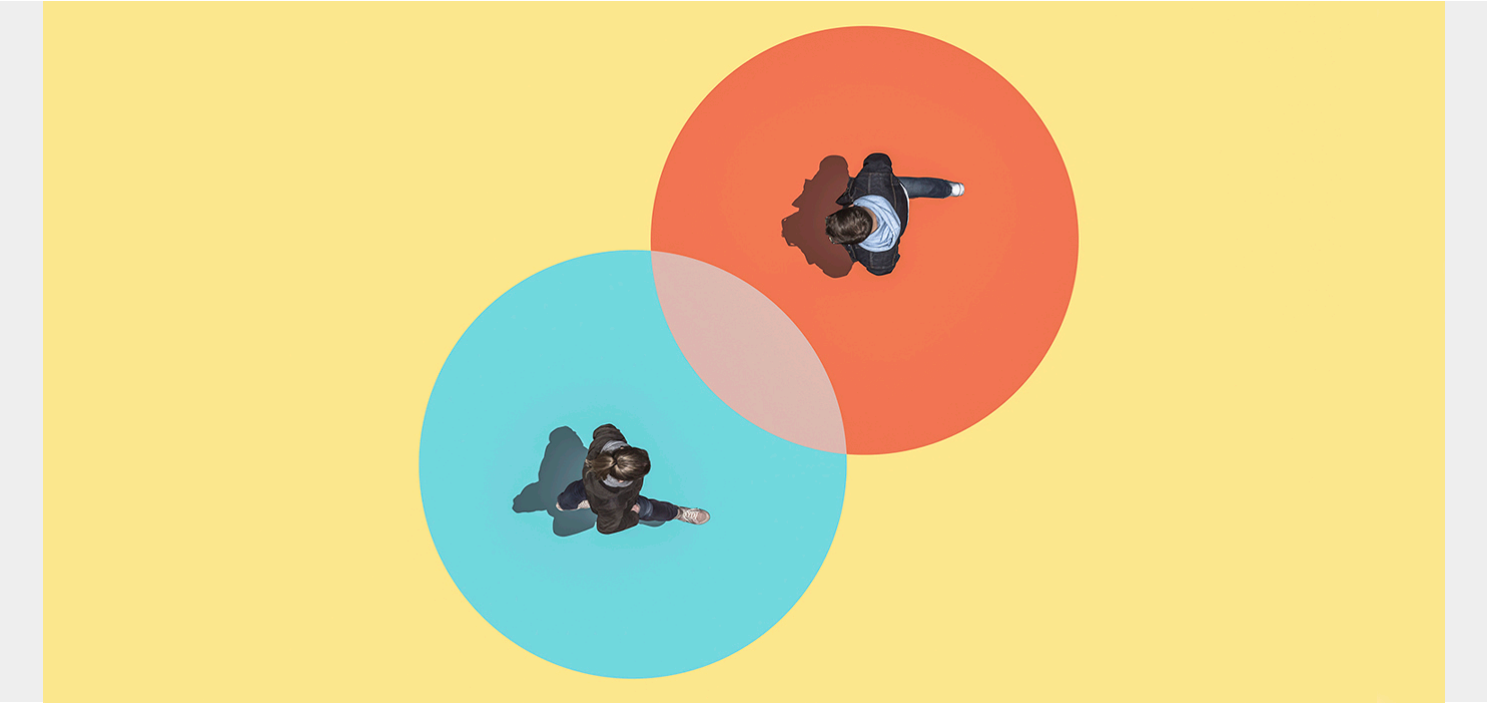
MICROSERVICES VS APIS: WHAT'S THE DIFFERENCE?



A microservice is a small, single service offered by a company. It derives from the distributed computing architecture that connects many small services, rather than having one large service. The microservice can then be delivered through an application programming interface (API).

An API is a method of communication between a requester and a host, most often accessible through an IP address. The API can communicate multiple types of information to users, such as:

- Data you want to share
- A function you want to provide

In short, talk of a microservice has to do more with the software's architecture, and the API has to do with how to expose the microservice to a consumer.

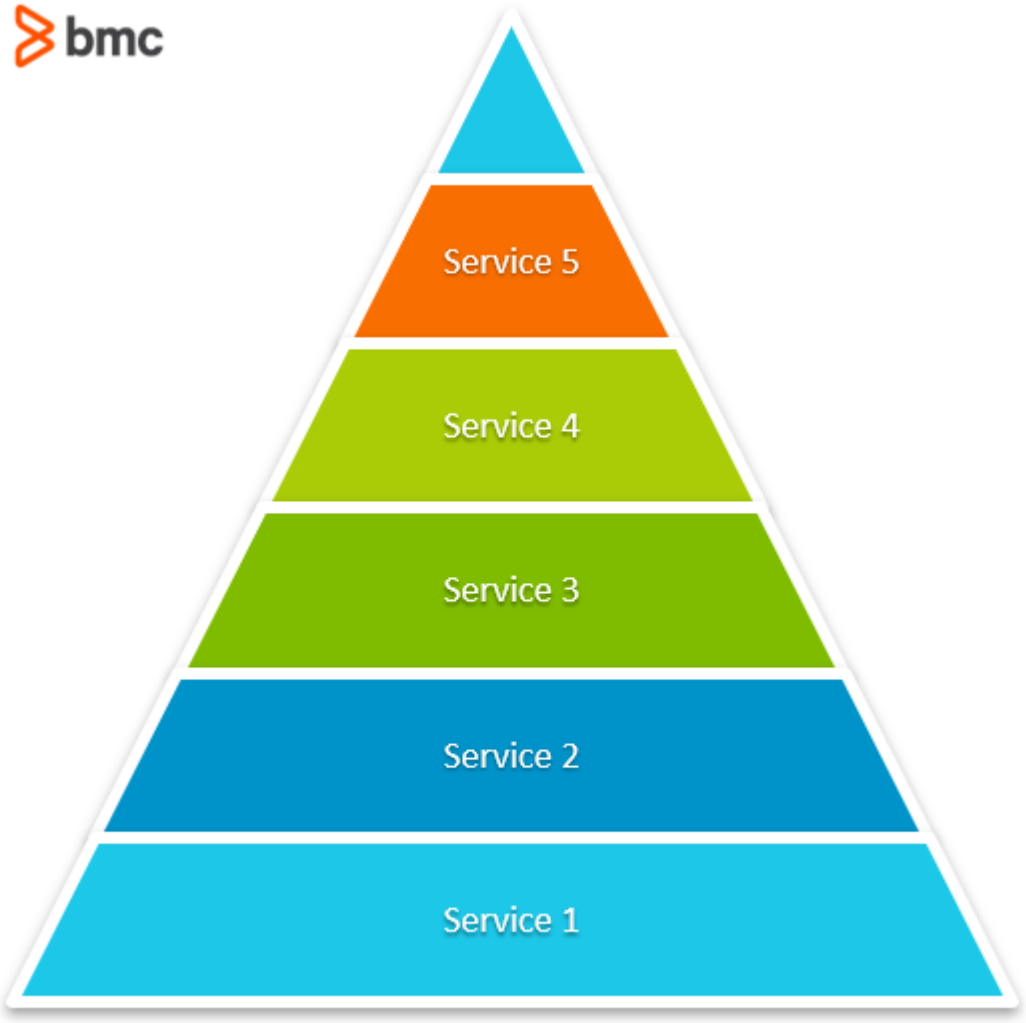
How microservices work

Microservices extend from the idea that a company provides a large, single service. Microservices come as individual functions. If Microsoft Word were to be split into microservices, perhaps there would be one offered as the blank sheet of paper, one as a spell checker, one service as a formatting tool.

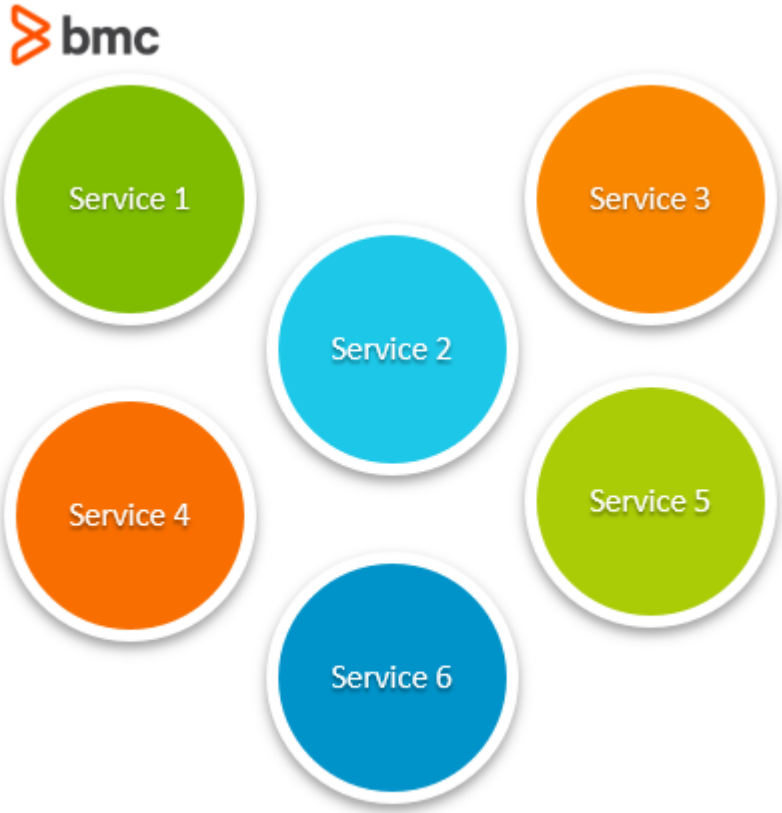
Kubernetes has allowed computer software to adapt. While Kubernetes has its own advantages, it has also pushed software design away from a single monolith of services—and towards a combination of many, small services working together. That's because of the Kubernetes design, which can:

- Efficiently orchestrate the use of single containers on servers

- Increase system reliability and scalability
- Decrease associated management and resource costs



Monolith architecture



Distributed microservices

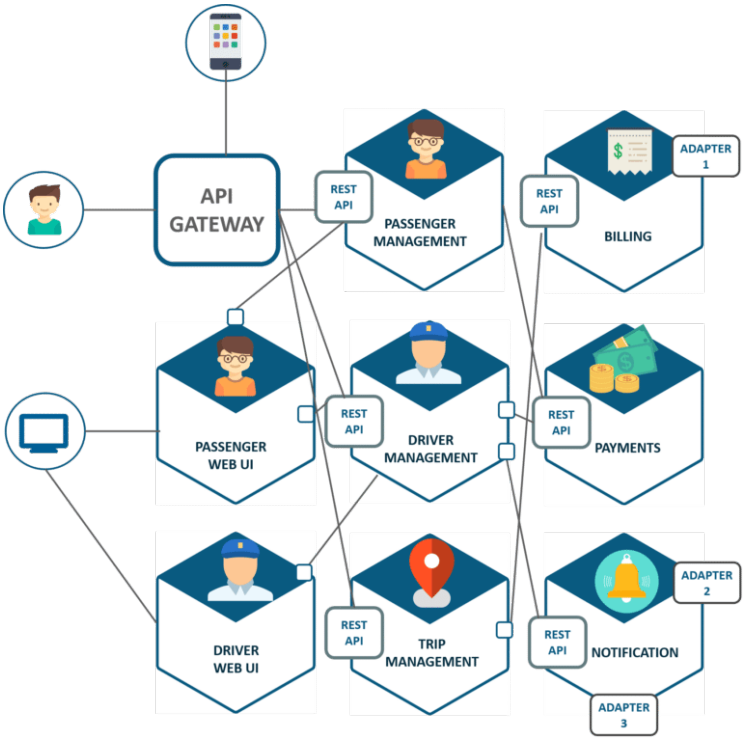
(Learn more about microservice vs monolith architecture.)

Examples of microservices

Microservices are very simple. Simplicity is a primary goal. They can be thought of as roles in a company; one microservice serves a very particular role and has just one job to do.

DZone put together an excellent graph of different microservices that Uber offers, communicating with one another through APIs and performing different tasks. Uber builds different services for each task:

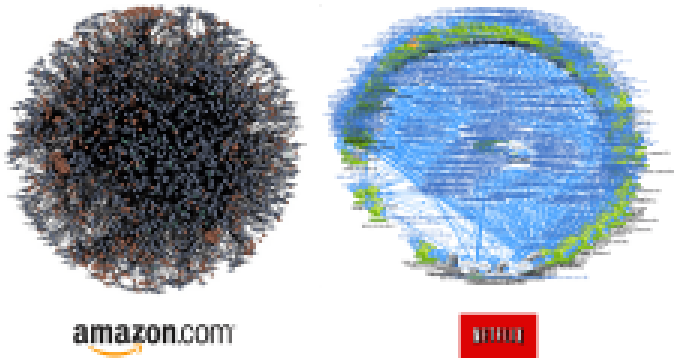
- Passenger management
- Passenger web UI
- Billing
- Driver management
- Driver web UI
- Payment
- Trip management
- Notifications



(*Source*)

Microservices can also be illustrated through graphs, where one microservice is a single node that communicates to another service via an API. The architecture can grow and grow as more services are tacked onto the system.

As you can imagine, the graphs of the large companies can be extensive, like a small city. Here are the supposed graphs of Amazon and Netflix:



(Source)

Microservices rely on APIs

The API is a communication tool—it lets one service interact with another. An API itself cannot do anything unless it is connected to something, like a cell phone that just sits there. The API becomes useful when it is connected to services and microservices such as:

- Function as a Service
- Machine Learning as a Service
- Software as a Service

The API is the way you can distribute the microservice to users. Instead of downloading software or popping in a disc, the API distributes your service.

The API is necessary for the microservice architecture to function because the API is the communication tool between its services. Without an API, there would be a lot of disconnected microservices. Technically, the microservice would just build to be a monolith again.

How APIs work

APIs are extremely versatile. You can:

- Create APIs on any containerized service
- Use many different languages—Java, Python, Go, to name a few
- Deploy APIs on any of the major cloud providers

APIs can increase both the usability and the exposure of your service. With the distribution made a lot easier, you can offer smaller services. (After all, you don't have to build a whole Adobe Suite just to prove viability).

Many APIs are RESTful and exposed through an endpoint like an HTTP endpoint. This means accessing information from an API is as easy as pinging a URL. GET, POST, PUT, DELETE commands, in conjunction with the URL, work as expected, fetching data or giving data to the API. Though REST APIs are the most common in modern web applications, other options include:

- SOAP
- RPC
- GraphQL

As a product, the API endpoint is usually served alongside a developer portal that informs developers how to use it and assigns devs an API key. If the goal of a microservice is to provide data on the registered vehicles in a given county, then the dev portal will explain:

- What the service does
- How the data is structured (i.e.; a data schema)
- What is required for a developer to use the API

Microservices vs APIs

Most good microservices have some type of API. If you want your microservice to be used, then you're going to create an API.

The API is to developers what having a social media account is to artists and creators. If you want people to use it, you use an API so they can receive it.

Related reading

- BMC DevOps Blog
- Getting Started with Containers and Microservices for Enterprise Leaders
- An Introduction To Micro Frontends
- Microservices vs SOA: How Are They Different?

- The Death (or Not) of Microservices