Microservices vs. API: The Benefits of Using Microservices

snipcart.com/blog/microservices-vs-api

Microservices and APIs are often talked about as if they're the same thing, but they're, in fact, complementary concepts.

While microservices let users access API, not every microservice has to be part of an API.

Microservices break large applications into independent parts, giving developers more flexibility to create lean, performant applications. On the other hand, an API is a bridge that enables microservices to communicate with the server and other APIs, making it possible for microservices to work independently yet in sync.

Many microservices only do internal tasks or help out the API functions. However, there are many ways to make an API without using microservices, so these technologies are more complementary than competing.

If this still sounds a bit confusing, don't worry; by the end of this article, you will be able to differentiate between the two and when to use microservices along with APIs.

What is the difference between a microservice and an API?

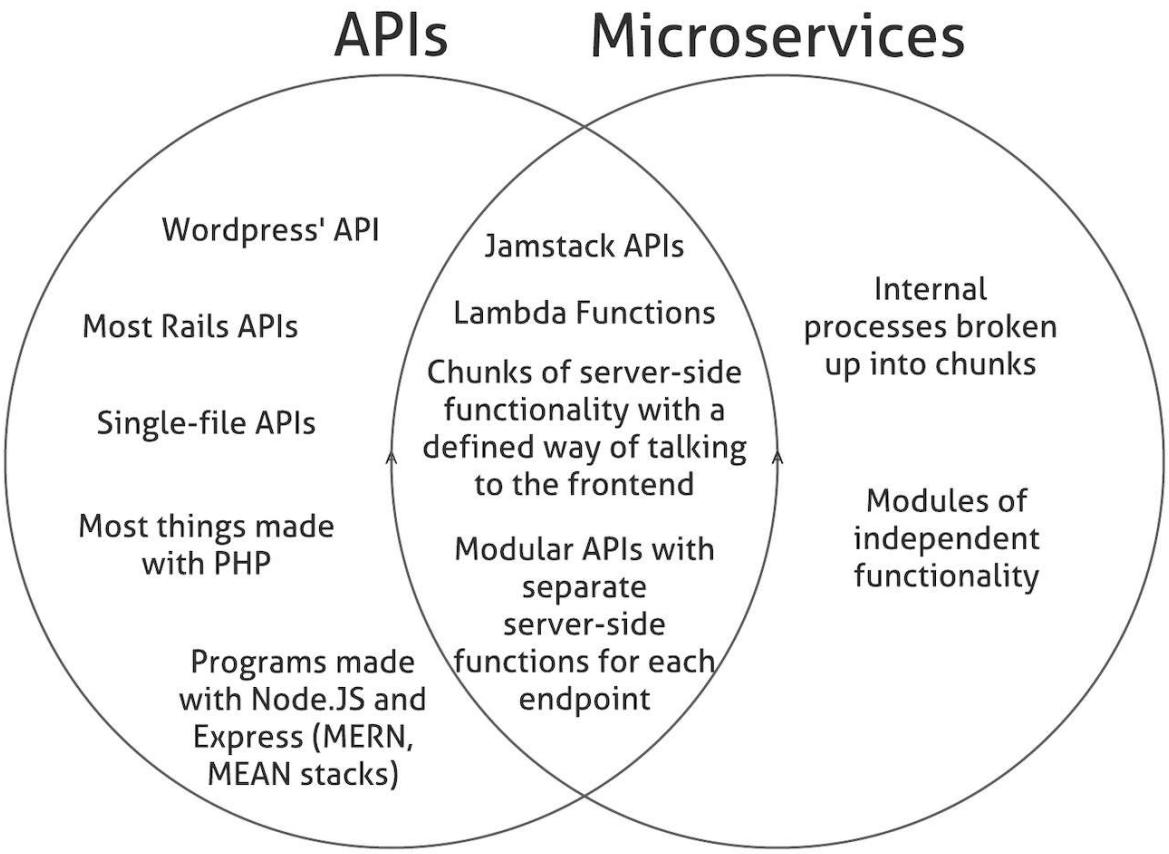
The difference between microservices and APIs is that microservices make up a single application from small, interconnected services that *communicate* via APIs.

APIs are a method of communication between a requester and a host, allowing services and products to leverage each other's data and functionalities through a documented interface.

An application can contain a series of microservices, each using an API to connect with one another.

APIs serve as the connective tissue that makes the interaction between different parts of your technology stack possible.

Here's a Venn diagram if you'd like another way to visualize the differences.



As you can see, microservices can be an API (the overlapping of the circles), but APIs aren't necessarily a microservice.

Some microservices aren't part of APIs (the right section on the Venn diagram) because they don't have to be accessible to the outside world. On the other hand, there are a *ton* of ways to make an API, and most of them don't involve microservices.

Here's a handy chart showing how microservices and APIs differ:

	Microservices	**APIs**
Scope	Architectural style that structures an application as a collection of interconnected services.	Standard or protocol used to communicate between applications and services.
Ways of accessing	Use APIs to maintain communication between every packaged business capability (PCB)	Offered by one application to another and accessible via the internet.
How they work	Consume APIs to connect services	Expose services to make them consumable
Goal	Provide components to an application	Serve as interfaces

What are APIs?

API (application programming interface) defines how two pieces of software can connect and talk to each other via their endpoints. For example, your APIs job could be used to keep in touch with external parties (customers or company partners). Most APIs are organized around rules or standards, like REST or GraphQL, so everybody knows how to use them.

An API call works like this:

1. APIs take a request from an application user and send it to the server.
2. The server retrieves that data and interprets it.
3. Finally, it performs the action the user requested.

Since this data is usually presented in JSON (JavaScript Object Notation) format, the application takes this information and formats it in a readable way for the user.

An online store API, for example, takes data from the order you've just placed using your browser, sends it to the store's server for processing, then returns a response to the customers notifying them whether the order was successful or not.

A web API call for an eCommerce store could look like this:

```
{
        "customerName" : "Jane Doe",
        "item" : "Cat Mug",
        "price" : 25
}
```

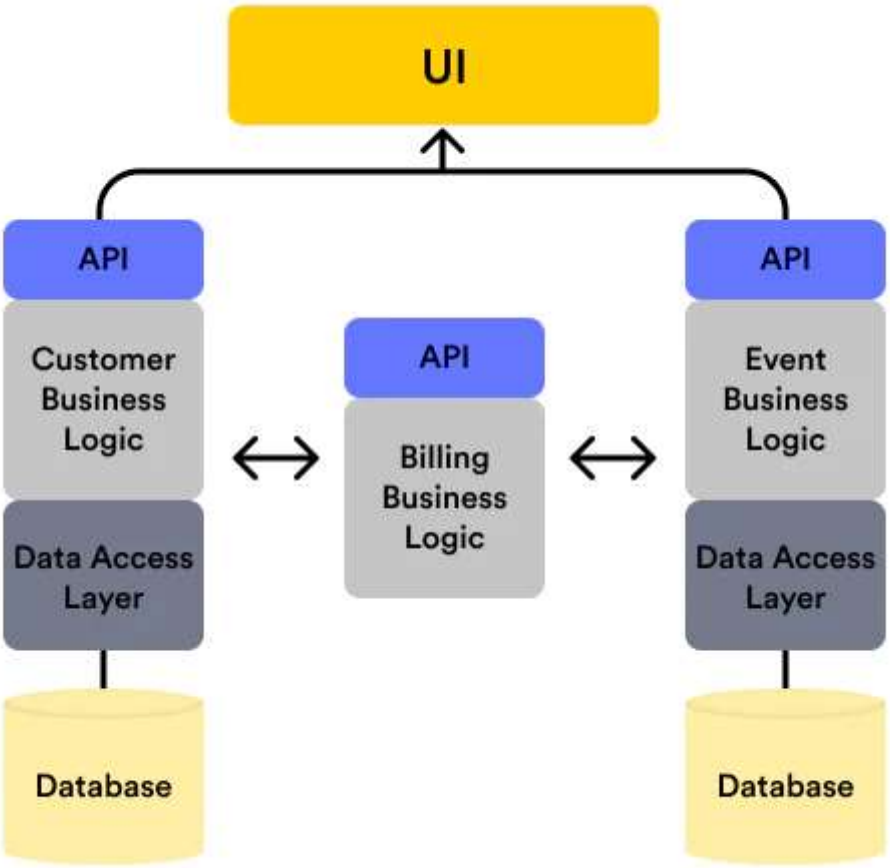
The API sends a request with this data to the server, where it checks that there is stock, saves the order information, and returns with a response for the customer. An example response may look like this:

```
{
        "status": true,
        "message": "Thank you, we've received your order. We will contact you as soon as your
package is shipped."
}
```

The app receives this response and delivers the message to the client.

What are microservices?

Microservices are pieces of software that perform a single, independent task within a more extensive application. They contrast with monolithic applications because, rather than building web applications as one unit made up of a UI, a server-side application, and a database, they break each part of the application into several PCBs connected via APIs.



The microservices architecture lets you make changes to individual microservices without affecting the rest of the application. Building your application with microservices reduces complexity and makes maintenance a lot easier because you can edit these little pieces individually.

Also, the microservices architecture allows for software teams to streamline communication, prepare for failure, and ensure better integration with other features. Microservices also enable distributed development, which means that you can develop multiple microservices simultaneously. resulting in faster sprints.

It's like building a site out of Legos: if you don't like one of them, you can just replace it and leave the rest of the site intact. That means your technical debt goes down to almost nothing, and if you keep with this approach, you'll never run into one of those lose-lose architecture dilemmas that we all hate.

What are microservices used for?

Microservices improve the flexibility of an application. This can be used in a variety of ways like:

1. Legacy application refactoring: If you're still using a legacy architecture, leveraging microservices to move to the cloud, change functionalities, and add new features would enable you to build incrementally and reduce technical complexity.
2. Real-time data processing: For instance, banking platforms and online booking services use microservices to execute operations in real-time and deliver an immediate output.
3. Applications providing third-party services: Since third-party applications like plugins require extra CPU power to operate, you can use microservices to make them more efficient.

What are the benefits of microservices?

The microservices architecture represents a paradigm shift when compared to the monolithic architecture. Microservices decentralize software development and enable agile methodologies, resulting in faster testing and deployment.

Here's a handy table with the benefits of microservices to help you wrap your mind around what makes microservices so compelling for developers.

Benefits of microservices architecture	
Resilience Since every service is independent, they do not impact one another, which means that if one goes down, the others will remain up.	**High scalability** Microservices can be scaled or downscaled across different servers and infrastructures depending on your needs.
Faster time to market The microservices architecture enable shorter development cycles, resulting in faster updates and less time-to-feature.	**Greater accessibility** Microservices enable developers to understand and enhance their code, delivering value faster quickly.
Ease of deployment Microservice-based applications are smaller and easier to deploy than monolithic applications.	**Open standards** APIs enable developers to build their microservices using the programming language and technology they prefer.
Use of APIs Microservices rely heavily on APIs and API gateways to make communication between different microservices possible.	**Increased Security** Microservices enable data separation. Each service has its own database, making it harder for hackers to compromise your application.

Microservices vs. APIs: how they work together

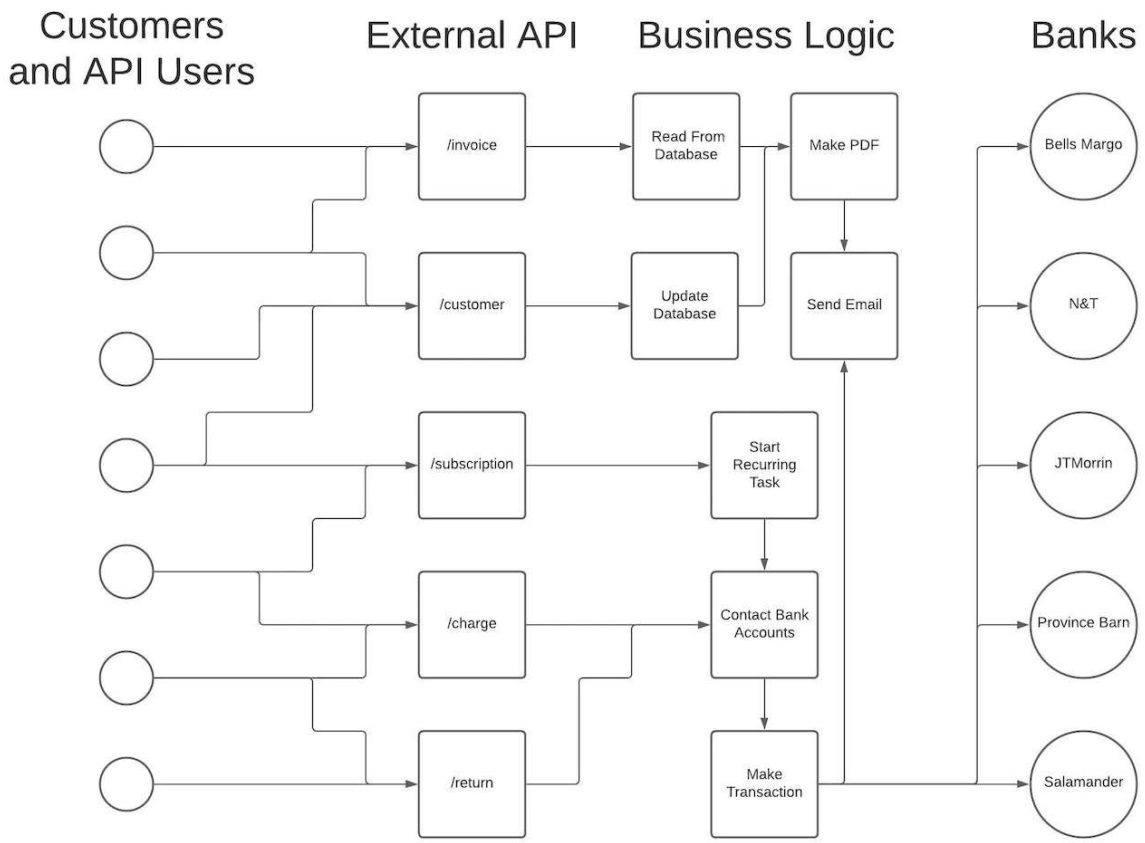
Imagine we're building a payment processor service from scratch. There are plenty of functionalities that your application would need. Such as:

- Sending emails
- Contacting the banks
- Running a transaction
- Creating invoice PDFs
- Reading from the database
- Inserting into or updating the database
- Scheduling a recurring task for subscriptions

All these functionalities work separately; they're also independent of each other and communicate using APIs. In this example, to make all these interactions possible, you need APIs to trigger functions such as creating invoices, customer profiles, subscriptions, charges, and returns.

To build a payment processor like the one we're describing, you would need to create five new microservices, one for each of the new actions you're making available for the users.

Our payment processor architecture would look something like this:

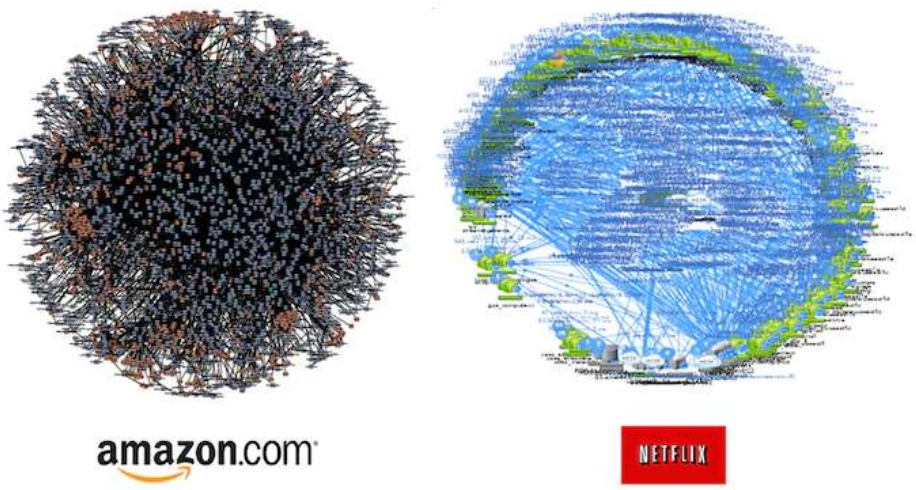


It can look a little complex on the surface, but it's a payment processor; after all, some complexity is to be expected.

To understand better how this payment processor works, let's go through the columns one by one.

- On the far left, we have the users. They only have access to the microservices in the External API column.
- The external APIs and the API users make up the API that interacts with the business logic in the third column.
- The third column shows microservices accomplishing tasks independently and communicating with one another to trigger other tasks.
- The fourth column shows the banks that finally receive the information from the API user in the first column.

Our payment processor graph looks pretty complex because of all the lines. Still, it is relatively simple when matched up against the same chart for more prominent companies like Amazon or Netflix:



Instead of working on the entire application codebase, the DevOps team only has to work with one manageable chunk of code at a time, just one of the dots on that graph.

Conclusion

APIs and microservices are complementary.

APIs and microservices are now massive parts of the modern web development process, but there's still a lot of confusion about them.

The easiest way to understand microservices is that they break down an application into smaller parts that work simultaneously.

Microservices are the blocks of your application and perform different services, while REST APIs work as the glue or the bridge that integrates these separate microservices.

APIs can be made up, wholly or partially, out of microservices. Developers can use Microservices for a lot more, though. Each service performs one function in the overall app, which is backed by an API that enables it to communicate with the rest. This allows each service to be scaled independently of the others to meet demand spikes or decrease in activity on any given area of your product.

Have you ever implemented a microservices architecture? Let us know in the comments.