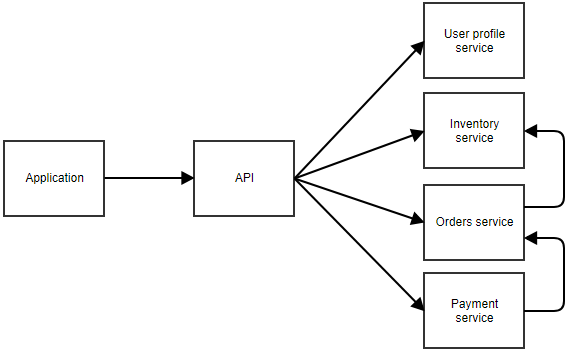
**Microservices**

When you write your application as a set of microservices, you’re actually writing multiple applications that will work together. Each microservice has its own distinct responsibility and teams can develop them independently of other microservices. The only dependency between them is the communication. As microservices communicate with each other, you will have to make sure messages sent between them remain backwards-compatible (Backward compatibility is a property of a system, product, or technology that allows for interoperability with an older legacy system, or with input designed for such a system, especially in telecommunications and computing.). This requires some coordination, especially when different teams are responsible for different microservices. A diagram can explain.

میکروسرویس

وقتی برنامه خود را به عنوان مجموعه میکروسرویسها می نویسید ، در واقع چندین برنامه می نویسید که با هم کار خواهند کرد. هر میکروسرویس وظیفه متمایز خود را دارد و تیم ها می توانند آنها را بطور مستقل از سایر میکروسرویس ها توسعه دهند. تنها وابستگی بین آنها ارتباط است. از آنجا که میکرو سرویس ها با یکدیگر ارتباط برقرار می کنند ، شما باید مطمئن شوید که پیام های ارسالی بین آنها backwards-compatible (backwards-compatible خاصیت یک سیستم ، محصول یا فناوری است که امکان همکاری با سیستم قدیمی تریا ورودی را برای چنین سیستمی بخصوص در ارتباطات از راه دور و محاسبات فراهم می کند.) باقی می مانند. این امر به برخی از هماهنگی ها نیاز دارد ، به ویژه هنگامی که تیم های مختلف مسئولیت میکروسرویس های مختلف را بر عهده دارند. یک نمودار می تواند توضیح دهد.

[](https://blog.ndepend.com/wp-content/uploads/layered-6.png)

In the above diagram, the application calls a central API that forwards the call to the correct microservice. In this example, there are separate services for the user profile, inventory, orders, and payment. You can imagine this is an application where the user can order something. The separate microservices can call each other too. For example, the payment service may notify the orders service when a payment succeeds. The orders service could then call the inventory service to adjust the stock.

در نمودار فوق ، برنامه با یک API مرکزی تماس می گیرد که تماس را به سمت میکروسرویس صحیح هدایت می کند. در این مثال ، سرویس های جداگانه ای برای پروفایل کاربر ، موجودی ، سفارشات و پرداخت وجود دارد. شما می توانید تصور کنید این یک برنامه کاربردی است که در آن کاربر می تواند چیزی سفارش دهد. خدمات میکرو سرویس جداگانه نیز می توانند با یکدیگر تماس بگیرند. به عنوان مثال ، سرویس پرداخت می تواند در صورت موفقیت در پرداخت ، خدمات سفارشات را به شما اطلاع دهد. سپس خدمات سفارشات می توانند برای تنظیم سهام ، از سرویس موجودی تماس بگیرند.

There is no clear rule of how big a microservice can be. In the previous example, the user profile service may be responsible for data like the username and password of a user, but also the home address, avatar image, favorites, etc. It could also be an option to split all those responsibilities into even smaller microservices.

هیچ قانون روشنی وجود ندارد که میکروسرویس چقدر میتواند بزرگ باشد. در مثال قبلی ، سرویس پروفایل کاربر ممکن است مسئول داده هایی مانند نام کاربری و رمزعبور کاربر باشد ، اما همچنین آدرس منزل ، تصویر نماد ، موارد دلخواه و غیره نیز می تواند گزینه ای برای تقسیم همه آن مسئولیت ها به میکروسرویسهای حتی کوچکتر باشد.

**Advantages**

* You can write, maintain, and deploy each microservice separately.
* A microservices architecture should be easier to scale, as you can scale only the microservices that need to be scaled. There’s no need to scale the less frequently used pieces of the application.
* It’s easier to rewrite pieces of the application because they’re smaller and less coupled to other parts.

مزایا

• می توانید هر میکروسرویس را بطور جداگانه بنویسید ، نگهداری و استقرار دهید.

• یک معماری میکروسرویس آسان تر مقیاس گذاری می شود ، زیرا فقط میکروسرویس های مورد نیاز برای مقیاس گذاری را می توانید مقیاس کنید. نیازی به مقیاس کردن قطعات کمتر استفاده شده از برنامه نیست.

• بازنویسی بخش هایی از برنامه ساده تر است زیرا آنها کوچکتر و کمتر با سایر قسمت ها همراه هستند.

**Disadvantages**

* Contrary to what you might expect, it’s actually easier to write a well-structured monolith at first and split it up into microservices later. With microservices, a lot of extra concerns come into play: communication, coordination, backward compatibility, logging, etc. Teams that miss the necessary skill to write a well-structured monolith will probably have a hard time writing a good set of microservices.
* A single action of a user can pass through multiple microservices. There are more points of failure, and when something does go wrong, it can take more time to pinpoint the problem.

معایب

• برخلاف آنچه ممکن است انتظار داشته باشید ، در ابتدا نوشتن یک monolith خوب ساختار یافته و بعداً تقسیم آن در میکروسرویس آسان تر است. با استفاده از microservices ، نگرانی های اضافی زیادی به وجود می آید: ارتباطات ، هماهنگی ، backward compatibility ، ورود به سیستم و غیره. تیم هایی که مهارت لازم برای نوشتن یک monolith خوب ساختار یافته را از دست داده اند ، احتمالاً در نوشتن مجموعه خوبی از microservices کار سختی دارند.

• یک عمل واحد از یک کاربر می تواند از چند microservices عبور کند. نقاط شکست بیشتر وجود دارد و وقتی چیزی پیش آمد ، می تواند زمان بیشتری را برای مشخص کردن مشکل به طول بینجامد.

**Ideal for:**

* Applications where certain parts will be used intensively and need to be scaled
* Services that provide functionality to several other applications
* Applications that would become very complex if combined into one monolith
* Applications where clear bounded contexts can be defined

مناسب برای:

• برنامه هایی که قطعات خاصی به صورت فشرده استفاده می شوند و نیاز به مقیاس بندی دارند

• خدماتی که عملکرد چندین برنامه دیگر را ارائه می دهند

• برنامه هایی که اگر در یکپارچه سازی قرار گیرند بسیار پیچیده می شوند

• برنامه هایی که در آنها زمینه های مرز مشخص را می توان تعریف کرد

<https://dzone.com/articles/software-architecture-the-5-patterns-you-need-to-k>