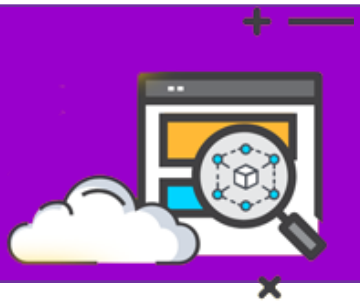


عنوان مقاله: میکروسرویس چیست؟
نویسنده مقاله: علیرضا ارومند
تاریخ انتشار: مرداد ماه ۱۳۹۸
منبع: https://nikamooz.com/what-micro-service/

مقدمه

اگر اهل توسعه نرم افزار و پیگیری اخبار روز دنیای توسعه نرم افزار باشید حتما اسم میکروسرویس به گوشتان خورده است. تقریبا هیچ بلاگ و کتابی نیست که این روزها یکی دوتا پست یا فصل خودش را به این موضوع اختصاص نداده باشد. اغلب فریم‌ورک ها و ابزارهای توسعه هم سعی می‌کنند برای توسعه میکرو سرویس ها راهکارهایی ارائه بدهند و به کمک این راهکارها طرفداران بیشتری در دنیای توسعه نرم افزار برای خودشان به دست بیاورند.

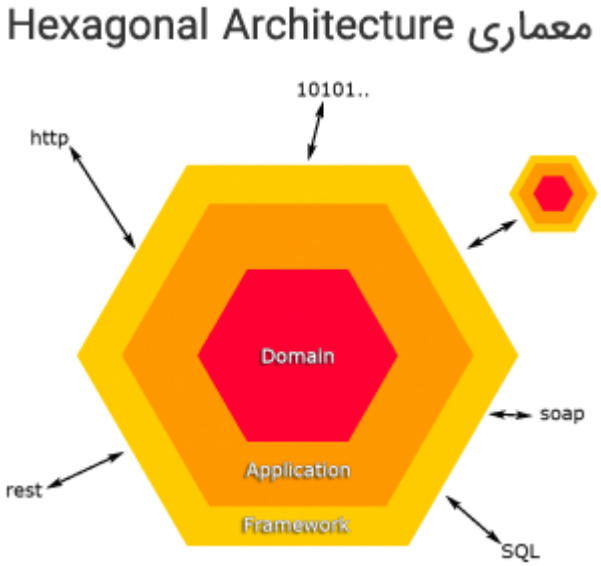
برای همین تصمیم گرفتم در قالب چند مقاله درباره این روش توسعه نرم افزار و بایدها و نبایدهای آن و تکنیک های پیاده سازی Micro Service دانسته‌های خودم را با شما به اشتراک بگذارم. بخش زیادی از مطالب این نوشته‌ها ترجمه مطالب آقای کریس ریچاردسون هست که بعضا با تجربه‌های شخصی خودم سعی کردم مطالب ساده تر و کامل تر بیان بشود و اینکه قطعا به نظرم هیچ کاری در دنیای نرم افزار بدون پیاده سازی و کد نویسی به درد نخواهد خورد برای همین توی این سری مطالب بعضا دست به کد می‌شویم و پیاده سازی‌هایی خواهیم داشت که برای این کار از C#.NET و .NET Core استفاده خواهیم کرد.

دنیا قبل از میکروسرویس به چه صورت بود؟

فرض کنید شما برای توسعه یک نرم افزار دعوت به همکاری شده اید. از شما خواسته می شود یک CMS خبری توسعه بدهید که کارش بسیار ساده است. خبرنگارها امکان ثبت اخبار متنی دارند، دبیرها و سردبیرها امکان انتخاب و انتشار اخبار در صفحات سایت دارند و در نهایت کاربران سایت امکان مشاهده‌ی مطالب سایت را خواهند داشت.

از آنجا که شما یک توسعه دهنده حرفه‌ای هستید احتمالا سراغ یک ابزار خوب و یک ساختار خوب و تمیز خواهید رفت، مثلا معماری Hexagonal یا معماری Onion برای توسعه انتخاب می‌کنید و در نهایت یک نرم افزار بسیار تمیز توسعه می‌دهید. در همچین شرایطی احتمالا منطق برنامه در مرکز برنامه قرار گرفته و جاهایی که نیاز به ارتباط با زیرساخت و استفاده از ابزار دارید به جای وابسته شدن به تکنولوژی و زیرساخت از اینترفیس‌ها استفاده می‌کنید و

وابستگی‌ها را به خارج از دامنه اصلی برنامه هدایت می‌کنید خروجی مناسب برای برنامه خودتان طراحی و پیاده سازی می‌کنید.



هر چند در این روش از نظر منطقی برنامه ما کاملا ماژولار طراحی و پیاده سازی شده است اما در واقع کل این ماژول ها کاملا وابسته به هم هستند و در قالب یک بسته نرم افزاری روی خروجی قرار می‌گیرند. اینکه این یک بسته نرم افزاری دقیقا چه چیزی است بستگی به تکنولوژی‌های مورد استفاده شما دارد اما در محیط .NET شما یک یا چند اسمبلی دارید که در نهایت به عنوان خروجی برنامه شما شناخته خواهند شد و هر جایی نیاز به نرم افزار داشته باشید کل این اسمبلی‌ها باید در کنار هم بسته بندی بشوند و خدمات خودشان را ارائه بدهند.

توسعه برنامه به این روش بسیار فراگیر و مرسوم است. البته این فراگیری دلیل خوبی دارد و آن سادگی در توسعه و نصب برنامه است. ابزارها و IDEها به سادگی این قابلیت را در اختیار شما قرار می‌دهند که یک برنامه‌های خودتان را توسعه بدید و نصب و راه اندازی کنید. با یک نصب ساده قابلیت تست کل برنامه را دارید و برای راه اندازی نسخه جدید برنامه فقط کافیه بسته‌های نرم افزاری خودتان را کپی کنید و همه چیز آماده است. این روش توسعه همون چیزی است که در اصطلاح ما به آن Monolith می‌گوییم.

پیش به سوی جهنم!

از قدیم گفتند: "هیچ ارزانی بی حکمت نیست" اگر بخواهیم به زبان برنامه نویسی ترجمه کنیم می‌شود "هیچ سادگی بدون محدودیت نیست." هر نرم افزار و پروژه‌ای در صورتی که موفق بشود قطعا تمایل به رشد دارد و موفقیت بیشتر پروژه موجب رشد بیشتر پروژه خواهد شد. در نهایت بعد از مدتی پروژه کوچک و ساده ما تبدیل به هیولایی بزرگ و وحشتناک خواهد شد.

اما ممکن است به این فکر کنید که زیاد شدن حجم کدها در طول دوران توسعه نرم افزار و تغییرات آن چه مشکلی می‌تواند داشته باشد؟ در ادامه به چند مورد از این مشکلات خواهیم پرداخت.

مشکل اول: با بزرگ و پیچیده شدن نرم افزار تیم توسعه شما با مشکلات فراوانی دست و پنجه نرم خواهد کرد. هر تصمیمی برای تغییر در برنامه یا ایجاد سریع یک ویژگی و ارائه آن احتمالا به بن بست خواهد رسید. بزرگترین مشکل این نرم افزارها پیچیدگی بیش از حد این برنامه‌ها است. معمولا نرم افزارها در طول سالیان آنقدر بزرگ و پیچیده می‌شوند که درک درست و دقیق عملکرد آنها برای یک توسعه دهنده نرم افزار غیر ممکن می‌شود.

در نتیجه این عدم توانایی شناخت درست و صحیح باعث می‌شود رفع خطاهای موجود یا اضافه کردن یک ویژگی جدید هم بسیار سخت و هم بسیار پیچیده باشد. نکته اصلی در این شرایط این است که با گذشت زمان این مشکل به شکل نمایی بیشتر و بیشتر می‌شود. هر چقدر فهم کد سخت‌تر بشود احتمال اشتباه بیشتر و احتمال پیدا کردن اشتباه‌ها کمتر و توان رفع آنها هم کمتر می‌شود، در نهایت به سورس کدی خواهیم رسید که اصطلاحا به اون big ball of mud می‌گوییم.

مشکل دوم: سورس کد حجیم می‌تواند باعث پایین آمدن سرعت توسعه نرم افزار بشود. هر وقت تصمیم به بازکردن پروژه بگیرید دقایق زیادی طول خواهد کشید تا IDE شما باز بشود و آماده به کار باشد. در کنار این شرایط احتمالا این سیستم عظیم بهره‌وری کل بستر شما رو هم پایین خواهد آورد.

مشکل سوم: احتمالا با داشتن یک سیستم بزرگ شما با عدم توانایی در انتشار بهبودها و توسعه‌های کوچک روبرو خواهید شد. مسلما سیستمی که باز کردن آن چند دقیقه طول بکشد، Build شدنش بعضا ۱ ساعت طول خواهد کشید و روال نصب راحتی هم نخواهد داشت. همچنین با توجه به اینکه در زمان نصب احتمالا سیستم از دسترس خارج خواهد بود و قاعدتا از دسترس خارج کردن یک سیستم عظیم به خاطر یک تغییر کوچک یا رفع باگ احتمالی جزئی، مقرون به صرفه نیست به همین خاطر نصب نسخه‌های جدید با فاصله‌های زمانی طولانی انجام خواهد شد.

مشکل چهارم: عدم استفاده بهینه از منابع یکی دیگر از مشکلات اساسی توسعه نرم افزار به این شکل هست. در نرم افزارهای متفاوت نیازهای متفاوتی وجود دارد. ممکن است بخشی از برنامه مصرف RAM زیادی داشته باشه و بخشی دیگر هم مصرف CPU اما در این روش امکان تخصیص یک منبع خاص به بخش خاصی که نیاز به آن منبع دارد، وجود نخواهد داشت. در نتیجه هنگامی که بخشی با مصرف CPU زیاد ارتقا داده بشود این امکان در اختیار همه بخش‌ها قرار خواهد گرفت و برای همه قسمت ها امکان استفاده از این منبع، بی رویه و ناصحیح وجود خواهد داشت.

مشکل پنجم: مشکلی که توسعه به روش Monolith به همراه داره انتشار مشکلات هست. کل برنامه در یک سیستم و در قالب یک پروسه اجرا خواهد شد. پس اگر ایرادی در هر یک از قسمت‌های برنامه به وجود بیاید، تمامی قسمت‌های

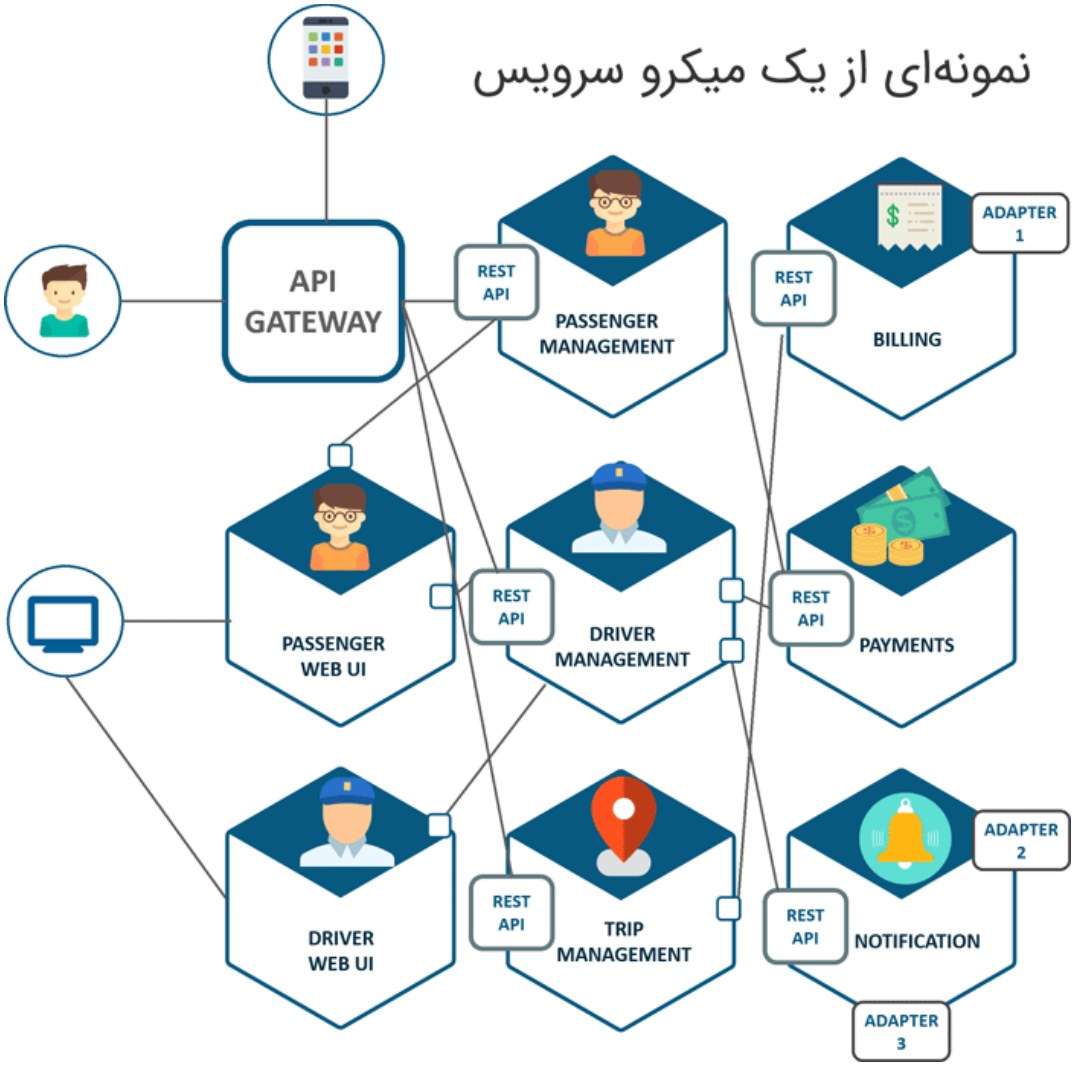
برنامه از کار خواهند افتاد و کل برنامه تا زمان رفع مشکل و نصب نسخه جدید از دسترس خارج خواهد بود. البته در این شرایط باید امیدوار بود که نصب نسخه جدید موجب ایجاد مشکلات جدید در سیستم نشود.

مشکل ششم: عدم توانایی در به کارگیری ابزارها و تکنولوژی‌های جدید هم یکی دیگر از ایرادات این روش توسعه نرم افزار هست. در شرایطی که شما یک نرم افزار بزرگ و پیچیده را سالها توسعه داده‌اید احتمالا وقتی با یک ابزار، تکنولوژی یا فریمورک جدید مواجه بشوید به جزء حسرت امکانات موجود کار دیگری از دستتان بر نمیاد. معمولا امکان اینکه سالها تلاش تیم دور ریخته بشود نه پذیرفته می‌شود نه قابل اجرا هست. در حال حاضر در یکی از شرکت‌های بزرگ درگیر مشاوره در یک پروژه ERP هستم که سال‌ها پیش و با تکنولوژی سال ۲۰۰۵ توسعه داده شده است در این ابزار از Net Framework ۲.۰ استفاده شده و برای توسعه وب هم از ASP.NET Web Form استفاده شده است و اینقدر سیستم بزرگ و پیچیده شده که در طی این سال‌ها کسی جرات دست زدن به سیستم و تغییر تکنولوژی را نداشته است در صورتی که خودتان جای یکی از مدیران پروژه و شرکت بگذارید چقدر احتمال دارد قبول کنید که ثمره ۱۵ سال توسعه یک تیم برنامه نویسی دور ریخته بشود و یک پروژه جدید استارت زده شود؟ به نظرتان در این ۱۵ سال که یک تیم ۱۵-۲۰ نفره به طور میانگین درگیر این پروژه بوده است چقدر هزینه تولید همچین ابزاری شده است؟ آیا تغییر تکنولوژی با این شرایط امکان پذیر هست؟
خوب به نظر میرسه کم کم داریم به پایان دنیا نزدیک می‌شیم. اما واقعا راه حلی برای این مشکلات نیست؟

بهشت گمشده: میکروسرویس [Micro service]

بسیاری از شرکتهای بزرگ نرم افزاری دنیا مثل eBay, Amazon, Netflix, Facebook و ... برای حل این مشکل به سراغ توسعه به روش میکرو سرویس رفتند. این شرکت‌ها تصمیم گرفتن که به جای داشتن هیولای غیرقابل کنترل تعداد زیادی مینی اپلیکیشن تولید کنند که خیلی خوب با هم تبادل اطلاعات می‌کنند و هر کدام از این مینی اپلیکیشن‌ها یک وظیفه خاص و دقیق را به انجام می‌رسانند.

در این روش هر سرویس مجموعه‌ای از وظایف مرتبط با هم را به طور کامل و بدون وابستگی به بخش دیگری به انجام می‌رساند. برای مثال در سیستم تولید محتوا و مدیریت خبر می‌توان به مواردی چون مدیریت نظرات، مدیریت ثبت خبر و محتوا، مدیریت فایل، مدیریت انتشار در بستر وب و ... اشاره کرد. در این روش هر مینی اپلیکیشن از یک معماری تمیز مثل Hexagonal یا Onion به طور داخلی استفاده می‌کند. هر مینی اپلیکیشن این توانایی را خواهد داشت که در صورت نیاز بخشی از خدمات و داده‌های خودش را برای سایر قسمت ها به صورت API در اختیار قرار بدهد. برای نصب و راه اندازی هم هر کدام از این مینی اپلیکیشن‌ها توانایی این را خواهند داشت که در یک VM جداگانه به کار خودشان ادامه بدهند یا به عنوان Docker Image در اختیار تیم زیرساخت برای نصب و راه اندازی قرار بگیرند.



در این شرایط هر کدام از بخش‌های عملیاتی برنامه به عنوان یک مینی اپلیکیشن توسعه داده شدند. مهم تر از آن حالا به جای یک وب اپلیکیشن بزرگ مجموعه ای از اپلیکیشن‌های کوچک داریم که به طور دقیق و حساب شده ای برای انجام کار اصلی و رسیدن به هدف اصلی با هم تعامل و همکاری خواهند کرد. برای مثال در بخش تولید خبر این امکان برای این مینی اپلیکیشن فراهم هست که برای نمایش تصاویر از مینی اپلیکیشن مربوط به مدیریت فایل‌های عکس کمک بگیرد و از طریق APIهای ارائه شده توسط بخش مدیریت تصاویر به عکس‌های ذخیره شده در سیستم دسترسی داشته باشد. برای ارتباط بین سرویس‌ها راهکارهای مختلفی وجود دارد و اگر با این روش‌ها آشنا نیستید اصلا نگران نباشید. در قسمت‌های بعد در مورد این روش‌ها کامل صحبت خواهیم کرد.

خوب تا اینجا همه چیز به نظر خوب میاد اما احتمالا به این موضوع فکر بکنید که اگر تعداد مینی‌اپلیکیشن‌ها زیاد بشود و هر برنامه هم API خودش را ارائه بدهد و برنامه ها نیاز داشته باشند که با هم ارتباط برقرار کنند تعداد زیادی آدرس بوجود خواهد آمد که هر برنامه باید آنها را مدیریت کند و این خودش شروع مشکلات می‌شود. اما خبر خوب

اینکه این مشکل به کمک API Gateway حل خواهد شد و در مورد جزئیات این بحث در قسمت آینده کامل صحبت خواهیم کرد.

مزایای میکروسرویس‌ها

گویا این روش توسعه جدید نرم افزار بسیاری از مشکلات روش Monolith را حل خواهد کرد. اما برای اینکه دقیق‌تر بدانیم به چه شکلی این مشکلات حل خواهند شد بیایید با هم نگاهی به برخی ویژگی‌های مفید میکرو سرویس‌ها بندازیم.

احتمالا همه شما با روش تقسیم و غلبه برای حل کردن مشکلات آشنا هستید، به جای حل کردن یک مشکل بزرگ بهتر است مشکل به قطعات کوچک شکسته بشود و هر قطعه به طور جداگانه حل بشود و در نهایت جواب نهایی از مجموع جواب‌های به دست آمده تشکیل خواهد شد. خوب همین فلسفه در مورد میکرو سرویس‌ها هم صدق می‌کند. هیولای Monolith به تعداد زیادی مینی اپلیکیشن با قابلیت مدیریت و نگهداری بهتر شکسته می شود و حالا تعداد زیادی صورت مسئله جزئی برای حل کردن خواهیم داشت. قطعا با کوچک شدن برنامه‌ها فهم و توسعه برنامه ها هم به شدت ساده تر از قبل خواهد شد.

اغلب شرکت‌های نرم افزاری از تعدد stackهای توسعه نرم افزار وحشت دارند و معمولا محدودیت‌های زیادی در انتخاب ابزارها و فریم‌ورک‌های توسعه نرم افزار وجود دارد با این حالا در شرایطی که تعداد زیادی مینی اپلیکیشن داریم به راحتی می‌توانیم در هر کدام از این مینی اپلیکیشن‌ها از ابزارهایی که دقیقا مناسب با نیاز آن‌ها است استفاده کنیم. به راحتی اگر داده‌های ما ساختار گراف داشته باشید به سراغ یک گراف دیتابیس میرویم. برای هر برنامه زبان توسعه خاص آن را انتخاب می‌کنیم و به راحتی به هر تکنولوژی و فریم ورکی سلام خواهیم کرد.

یکی دیگر از ویژگی‌های توسعه میکرو سرویس قابلیت نصب و انتشار مستقل هر کدام از این میکرو سرویس‌ها است. دیگر نیازی نیست برای رفع یک باگ کوچک در یک قسمت کوچک برنامه کل سیستم از دسترس خارج شود. بلکه به سادگی همان قسمتی که نیاز به رفع ایراد دارد در مدت کوتاهی راه اندازی مجدد خواهد شد.

اما استفاده بهینه و صحیح از منابع هم شاید یکی از مهم ترین ویژگی‌های توسعه میکرو سرویس باشد. در این روش هر مینی اپلیکیشن به اندازه نیاز خود منابع و امکانات دریافت خواهد کرد و در صورت نیاز می‌توان منابع یک سرویس را افزایش داد یا یک سرویس خاص را در چند نسخه مختلف اجرا کرد.

خیلی هم خوب و خیلی هم عالی تا اینجای کار تمام معایب سیستم‌های قدیمی رفع شده و می‌توانیم به راحتی به کار خود ادامه دهیم. اما طبق اصل Pay for Play قطعا به دست آوردن این همه مزیت بدون هزینه و ایراد نخواهد بود. پس در ادامه بعضی از این ایرادات را با هم بررسی خواهیم کرد.

در این شرایط هر کدام از بخش‌های عملیاتی برنامه به عنوان یک مینی اپلیکیشن توسعه داده شدند. مهم تر از آن حالا به جای یک وب اپلیکیشن بزرگ مجموعه ای از اپلیکیشن‌های کوچک داریم که به طور دقیق و حساب شده ای برای انجام کار اصلی و رسیدن به هدف اصلی با هم تعامل و همکاری خواهند کرد. برای مثال در بخش تولید خبر این امکان برای این مینی اپلیکیشن فراهم هست که برای نمایش تصاویر از مینی اپلیکیشن مربوط به مدیریت فایل‌های عکس کمک بگیرد و از طریق APIهای ارائه شده توسط بخش مدیریت تصاویر به عکس‌های ذخیره شده در سیستم دسترسی داشته باشد. برای ارتباط بین سرویس‌ها راهکارهای مختلفی وجود دارد و اگر با این روش‌ها آشنا نیستید اصلا نگران نباشید. در قسمت‌های بعد در مورد این روش‌ها کامل صحبت خواهیم کرد.

خوب تا اینجا همه چیز به نظر خوب میاد اما احتمالا به این موضوع فکر بکنید که اگر تعداد مینی‌اپلیکیشن‌ها زیاد بشود و هر برنامه هم API خودش را ارائه بدهد و برنامه ها نیاز داشته باشند که با هم ارتباط برقرار کنند تعداد زیادی آدرس بوجود خواهد آمد که هر برنامه باید آنها را مدیریت کند و این خودش شروع مشکلات می‌شود. اما خبر خوب اینکه این مشکل به کمک API Gateway حل خواهد شد و در مورد جزئیات این بحث در قسمت آینده کامل صحبت خواهیم کرد.

معایب میکروسرویس‌ها

شاید بزرگترین ایراد توسعه به روش میکرو سرویس نام این روش باشد. به طور جدی در این نام روی اندازه کوچک و یا بهتر بگویم بسیار کوچک این سرویس‌ها تاکید شده است. اما این کوچک بودن به چه معناست؟ چقدر کوچک؟ اندازه یک مورچه به نسبت یک گربه بسیار کوچک است. اما همان گربه به نسبت یک فیل کوچک به حساب می‌آید و در مجموع به اندازه کل کره زمین به نسبت کل کائنات کمی فکر کنید ببنید کدام یک از این مواردی که اسم بردیم کوچک به حساب می‌آید. پس تعیین مقیاس برای سرویس‌ها یکی از مشکلات ما خواهد بود. هنگامی که به اندازه سرویس‌ها فکر می‌کنید حتما این نکته را به یاد داشته باشید که کوچک بودن اندازه سرویس‌ها وسیله ای برای رسیدن به هدفی بزرگ است نه یک هدف اصلی و بزرگ.

پیچیدگی برقراری ارتباط بین سرویس‌های مختلف و سایر پیچیدگی‌های فنی دیگری که هنگام توسعه یک سیستم توزیع شده با آن مواجه خواهیم شد بسیار بیشتر از زمانی است که یک نرم افزار Monolith توسعه می‌دهیم و قبل از انتخاب این روش توسعه، باید به این پیچیدگی‌ها و راهکارهایی که برای برخورد با این پیچیدگی‌ها داریم فکر کنیم. برای مثال برای استفاده از امکانات یک زیرسیستم دیگر در روش توسعه Monolith به راحتی از کلاسی نمونه سازی می‌کنیم و تابعی از آن کلاس را صدا می زنیم اما این کار را در یک سیستم توزیع شده نمی‌توانیم انجام دهیم.

در سیستم‌هایی که به روش Micro Service توسعه می‌دهیم تاکید فراوانی بر توانایی عملکرد هر سیستم به تنهایی وجود دارد و این بدان معنا است که هر میکرو سرویس دیتابیس اختصاصی خود و داده‌های اختصاصی خود را خواهد داشت و این توزیع شدگی داده‌ها در چند دیتابیس و مدیریت نسخه‌های موجود از یک داده در دیتابیس‌های مختلف

می‌تواند مشکلات زیادی را برای تیم توسعه به وجود آورد. شاید این مشکل زمانی بیشتر به چشم بخورد که اتفاقی در سیستم رخ دهد که نیاز به تغییر در پایگاه داده در چند سرویس مختلف داشته باشد و نیاز داشته باشیم یک Transaction را بین چند سرویس مختلف مدیریت کنیم.

یکی دیگر از شمشیرهای دولبه در روش توسعه میکرو سرویس نصب و راه اندازی آن است. شاید اگر به چند پاراگراف بالاتر برگردید مشاهده کنید که امکان نصب و راه اندازی جداگانه سرویس‌ها را به عنوان یکی از برتری‌های این روش توسعه نرم افزار شمردیم. اما با کمی دقت خواهید دید که همین مورد می‌تواند ایرادات زیادی را ایجاد کند. بعضا ممکن است یک سیستم بزرگ به بیش از ۱۰۰ سرویس کوچک تقسیم شود و نصب و راه اندازی و تنظیم ارتباطات این ۱۰۰ سرویس می‌تواند بسیار زمان‌گیر و پیچیده و پرخطا باشد.

برای بسیاری از این مشکلات راهکارهای عملیاتی زیادی تدارک دیده شده است که در مقالات آتی بررسی خواهیم کرد.
جمع بندی [میکروسرویس‌ها]

در این مطلب سعی کردیم با دو روش توسعه Monolith و میکروسرویس و مزایا و معایب هر کدام از این روش‌ها آشنا شویم و با مزایا و معایب هرکدام از این روش‌ها آشنا شدیم. به عنوان یک توسعه دهنده با نزدیک به ۱۷ سال سابقه توسعه نرم افزارهای مختلف یک روحیه مشترک بین همه توسعه دهنده‌ها سراغ دارم و این روحیه علاقه به استفاده از روش‌ها و ابزارهای جدید بدون توجه به نیاز واقعی کار است. پس پیشنهاد می‌کنم به جای تصمیم به توسعه تمام برنامه‌ها به روش میکروسرویس، قبل از انتخاب این روش، کاملا نیازهای اصلی سیستم را بررسی کنید و هر کدام از این روش‌ها را که مناسب سناریوی شما است انتخاب کنید. هر چند میکرو سرویس بسیار مطرح و پر طرفدار است اما با یک بررسی سریع می‌توانید نمونه‌های شکست خورده زیادی از سناریوهایی که برای توسعه میکروسرویس را انتخاب کرده اند پیدا کنید.