

گزارش کار پروژه پایانی معماری نرم افزار

**موضوع Microservice extraction based on knowledge graph from monolithic applications**

استاد دکتر اشتیانی

گرداورندگان

حاتم ابوالقاسمی ,حسین اله وردی, امیر اقابیکلو

الگو: معماری میکروسرویس

چهار چوب

شما در حال توسعه یک برنامه سازمانی سمت سرور. باید از مشتریان مختلف از جمله مرورگرهای رومیزی، مرورگرهای تلفن همراه و برنامه های کاربردی بومی تلفن همراه پشتیبانی کند. این برنامه همچنین ممکن است API برای احزاب 3 به مصرف افشا. همچنین ممکن است با برنامه های کاربردی دیگر از طریق خدمات وب یا یک کارگزار پیام ادغام شود. برنامه رسیدگی به درخواست (درخواست HTTP و پیام) با اجرای منطق کسب و کار؛ دسترسی به یک پایگاه داده؛ تبادل پیام با سیستم های دیگر؛ و بازگشت پاسخ HTML / طولاني / XML. اجزای منطقی متناظر با حوزه های عملکردی مختلف کاربرد وجود دارد.

مشکل

معماری استقرار برنامه چیست

* یک تیم از توسعه دهندگان کار بر روی برنامه وجود دارد
* اعضای تیم جدید باید به سرعت مولد شوند
* برنامه باید آسان به درک و تغییر
* شما می خواهید به تمرین استقرار مداوم از برنامه
* شما باید نمونه های متعددی از برنامه را بر روی ماشین های متعدد اجرا به منظور برآورده کردن مقیاس پذیری و الزامات در دسترس بودن
* شما می خواهید برای استفاده از فن آوری های در حال ظهور (چارچوب ها، زبان های برنامه نویسی، و غیره)

راه حل

تعریف معماری است که ساختار برنامه به عنوان مجموعه ای از سست همراه، خدمات همکاری. این رویکرد با محور Y مکعب [مقیاس مطابقت دارد](https://microservices.io/articles/scalecube.html). هر سرویس این است:

* بسیار قابل نگهداری و قابل آزمایش - توسعه و استقرار سریع و مکرر را قادر می سازد
* شل همراه با خدمات دیگر -- قادر می سازد یک تیم به طور مستقل کار اکثریت زمان در خدمات خود (ها) بدون اینکه تحت تاثیر تغییرات در خدمات دیگر و بدون تاثیر بر خدمات دیگر
* به طور مستقل قابل استقرار - یک تیم را قادر می سازد تا خدمات خود را بدون نیاز به هماهنگی با تیم های دیگر مستقر کند
* قادر به توسعه توسط یک تیم کوچک -- ضروری برای بهره وری بالا با اجتناب از رئیس ارتباطات بالا از تیم های بزرگ

سرویس ها با استفاده از پروتکل های سنکرون مانند HTTP/REST یا پروتکل های ناهمزمان مانند AMQP ارتباط برقرار می کنند. خدمات را می توان به طور مستقل از یکدیگر توسعه و مستقر کرد. هر سرویس پایگاه داده [خاص خود را](https://microservices.io/patterns/data/database-per-service.html) دارد تا از سرویس های دیگر جدا شود. ثبات داده ها بین خدمات با استفاده از الگوی ساگا [حفظ می شود](https://microservices.io/patterns/data/saga.html)

برای کسب اطلاعات بیشتر در مورد ماهیت یک سرویس، لطفا این [مقاله را بخوانید](https://microservices.io/post/microservices/general/2019/02/16/whats-a-service-part-1.html).

برنامه ساختگی تجارت الکترونیکی

بیایید تصور کنید که شما در حال ساخت یک برنامه تجارت الکترونیکی است که طول می کشد سفارشات از مشتریان، تایید موجودی و اعتبار در دسترس، و کشتی آنها را. این برنامه شامل اجزای متعددی از جمله StoreFrontUI، که پیاده سازی رابط کاربری، همراه با برخی از خدمات backend برای چک کردن اعتبار، حفظ موجودی و سفارشات حمل و نقل است. برنامه شامل مجموعه ای از خدمات است.



لطفا برنامه های کاربردی [مثال توسعه یافته توسط کریس ریچاردسون را ببینید](http://eventuate.io/exampleapps.html). این نمونه ها در Github جنبه های مختلفی از معماری میکرو سرویس را نشان می دهد.

این راه حل دارای تعدادی مزایا است:

* تحویل مداوم و استقرار برنامه های کاربردی بزرگ و پیچیده را قادر می سازد.
  + بهبود حفظ پذیری - هر سرویس نسبتا کوچک است و بنابراین درک و تغییر آسان تر است
  + تست پذیری بهتر - خدمات کوچکتر و سریع تر برای آزمایش
  + استقرار بهتر - خدمات را می توان به طور مستقل مستقر کرد
  + این شما را قادر می سازد برای سازماندهی تلاش های توسعه در اطراف تیم های متعدد، خودمختار است. هر یک (به اصطلاح دو پیتزا) تیم صاحب و مسئول یک یا چند خدمات است. هر تیم می تواند خدمات خود را مستقل از تمام تیم های دیگر توسعه، آزمایش، استقرار و مقیاس بندی کند.
* هر میکرو سرویس نسبتاً کوچک است:
  + آسان تر برای یک توسعه دهنده به درک
  + IDE سریع تر ساخت توسعه دهندگان سازنده تر
  + برنامه سریع تر شروع می شود، که باعث می شود توسعه دهندگان سازنده تر، و سرعت استقرار
* بهبود انزوای گسل. به عنوان مثال، اگر نشت حافظه در یک سرویس وجود داشته باشد آنگاه تنها آن سرویس تحت تأثیر قرار خواهد گرفت. سرویس های دیگر به رسیدگی به درخواست ها ادامه خواهند داد. در مقایسه، یک جزء بدرفتاری از یک معماری یکپارچه می تواند کل سیستم را پایین بیاورد.
* هر گونه تعهد بلند مدت به یک پشته فناوری را از بین می برد. هنگامی که در حال توسعه یک سرویس جدید شما می توانید یک پشته فن آوری جدید را انتخاب کنید. به طور مشابه، هنگام ایجاد تغییرات عمده در یک سرویس موجود می توانید آن را با استفاده از یک پشته فناوری جدید بازنویسی کنید.

این راه حل دارای تعدادی اشکال است:

* توسعه دهندگان باید با پیچیدگی اضافی ایجاد یک سیستم توزیع شده برخورد کنند:
  + توسعه دهندگان باید مکانیسم ارتباطی بین سرویس را پیاده سازی کنند و با شکست جزئی مقابله کنند
  + اجرای درخواست هایی که سرویس های متعدد را در بر می گیرد دشوارتر است
  + آزمایش تعاملات بین خدمات دشوارتر است
  + اجرای درخواست هایی که خدمات متعدد را در بر می گیرد نیاز به هماهنگی دقیق بین تیم ها دارد
  + ابزارهای توسعه دهنده/IDEs بر روی کاربردهای یکپارچه ساختمان گرا هستند و پشتیبانی صریحی برای توسعه برنامه های کاربردی توزیع شده ارائه نمی دهند.
* پيچيدگي استقرار . در تولید، پیچیدگی عملیاتی استقرار و مدیریت یک سیستم متشکل از بسیاری از خدمات مختلف نیز وجود دارد.
* افزایش مصرف حافظه. معماری میکروخوشه جایگزین نمونه های برنامه یکپارچه N با نمونه های خدمات NxM می شود. اگر هر سرویس در JVM (یا معادل) خود اجرا می شود، که معمولاً برای جداسازی مصادیق لازم است، آنگاه سربار M بار به اندازه زمان اجراهای JVM وجود دارد. از این گذشته، اگر هر سرویس بر روی VM خود اجرا می شود (به عنوان مثال EC2 نمونه)، به عنوان مورد در نت فلی، سربار حتی بالاتر است.

مسائل

مسائل زیادی وجود دارد که باید به آنها بپردازی.

یکی از چالش ها با استفاده از این رویکرد تصمیم گیری در مورد زمانی است که استفاده از آن منطقی است. در هنگام توسعه نسخه اول یک برنامه کاربردی، شما اغلب مشکلاتی را که این رویکرد حل می کند، داري. همچنین با استفاده از یک معماری استادانه و توزیع شده، توسعه را کند خواهد کرد. این می تواند یک مشکل عمده برای استارتاپ هایی باشد که بزرگترین چالش آنها اغلب این است که چگونه به سرعت مدل کسب و کار و برنامه همراه را تکامل دهند. با استفاده از انشعاب محور Y ممکن است آن را بسیار دشوارتر به سرعت ادامه. بعدها، با این حال، هنگامی که چالش این است که چگونه به مقیاس و شما نیاز به استفاده از تجزیه عملکردی، وابستگی های درهم ممکن است آن را دشوار به تجزیه برنامه یکپارچه خود را به مجموعه ای از خدمات است.

چگونه برنامه را به خدمات تجزیه کنیم؟

یکی دیگر از چالش ها تصمیم گیری در مورد چگونگی تقسیم سیستم به میکرو سرویس ها است. این بسیار یک هنر است، اما تعدادی استراتژی وجود دارد که می تواند کمک کند:

* [تجزیه با قابلیت کسب و کار و](https://microservices.io/patterns/decomposition/decompose-by-business-capability.html) تعریف خدمات مربوط به قابلیت های کسب و کار.
* [تجزیه توسط دامنه محور زیر دامنه طراحی](https://microservices.io/patterns/decomposition/decompose-by-subdomain.html).
* تجزیه توسط فعل و یا استفاده از مورد و تعریف خدمات که مسئول اعمال خاص است. به عنوان مانند خدمات حمل و نقل است که مسئول حمل و نقل سفارشات کامل است.
* تجزیه توسط اسم ها و یا منابع با تعریف یک سرویس است که مسئول تمام عملیات بر روی نهادها / منابع از نوع داده شده است. به عنوان مانند سرویس حساب که مسئول مدیریت حساب های کاربری است.

در حالت ایده آل، هر سرویس باید تنها مجموعه کوچکی از مسئولیت ها را داشته باشد. (عمو) باب مارتین در مورد طراحی کلاس ها با استفاده از اصل مسئولیت [واحد (SRP) صحبت می کند](http://www.objectmentor.com/resources/articles/srp.pdf). SRP یک مسئولیت یک کلاس را دلیلی برای تغییر تعریف می کند و بیان می کند که یک کلاس فقط باید یک دلیل برای تغییر داشته باشد. این منطقی به درخواست SRP به طراحی خدمات نیز هست.

قیاس دیگری که به طراحی سرویس کمک می کند طراحی تاسیسات یونیکسی است. یونیکسی تعداد زیادی آب و برق مانند grep، گربه و پیدا کردن را فراهم می کند. هر ابزار دقیقا یک چیز، اغلب فوق العاده خوب، و در نظر گرفته شده است با تاسیسات دیگر با استفاده از یک اسکریپت پوسته برای انجام وظایف پیچیده ترکیب شده است.

چگونه داده ها را حفظ کنیم؟

به منظور اطمینان از جفت شدن سست، هر سرویس پایگاه داده خاص خود را دارد. حفظ ثبات داده ها بین خدمات یک چالش است زیرا ۲ تراکنش مرحله ای/توزیع شده گزینه ای برای بسیاری از برنامه های کاربردی نیست. یک برنامه باید به جای آن از الگوی [ساگا استفاده کند](https://microservices.io/patterns/data/saga.html). یک سرویس یک رویداد را هنگامی که داده هایش تغییر می کند منتشر می کند. خدمات دیگر آن رویداد را مصرف می کنند و داده های خود را به روز می کنند. راه های متعددی برای به روز رسانی قابل اعتماد داده ها و انتشار وقایع از جمله منابع رویداد و [تراکنش ورود به سیستم Tailing وجود دارد](https://microservices.io/patterns/data/transaction-log-tailing.html).

چگونه پرس و جوها را پیاده سازی کنیم؟

چالش دیگر پیاده سازی پرس و جوهایی است که نیاز به بازیابی داده های متعلق به سرویس های متعدد دارند.

* ترکیب [API و](https://microservices.io/patterns/data/api-composition.html) فرماندهی [پرس و جو مسئولیت تفکیک (CQRS)](https://microservices.io/patterns/data/cqrs.html)

الگوهای مرتبط

الگوهای زیادی در ارتباط با الگوی ریز سرویس ها وجود دارد. معماری [یکپارچه جایگزین معماری](https://microservices.io/patterns/monolithic.html) میکرو سرویس است. الگوهای دیگر به مسائلی می پرداخت که هنگام اعمال معماری میکرو سرویس با آن مواجه خواهید شد.



* الگوهای تجزیه
  + [تجزیه توسط قابلیت کسب و کار](https://microservices.io/patterns/decomposition/decompose-by-business-capability.html)
  + [تجزیه توسط زیر دوم](https://microservices.io/patterns/decomposition/decompose-by-subdomain.html)
* پایگاه داده [در هر الگوی خدمات توصیف](https://microservices.io/patterns/data/database-per-service.html) می کند که چگونه هر سرویس پایگاه داده خود را به منظور اطمینان از جفت شدن شل است.
* الگوی [API Gateway نحوه دسترسی](https://microservices.io/patterns/apigateway.html) مشتریان به خدمات در یک معماری میکرو سرویس را تعریف می کند.
* [دیسکاوری سمت مشتری](https://microservices.io/patterns/client-side-discovery.html) و الگوهای کشف سمت سرور برای مسیر درخواست برای یک مشتری به یک نمونه خدمات در دسترس در معماری میکروخوشایند استفاده می شود.
* الگوهای تحریک پیام رسانی و روش از راه دور دو راه متفاوت است که خدمات می توانند ارتباط برقرار کنند.
* سرویس واحد [به ازای هر میزبان](https://microservices.io/patterns/deployment/single-service-per-host.html) و [چندین سرویس در هر الگوی](https://microservices.io/patterns/deployment/multiple-services-per-host.html) میزبان دو استراتژی استقرار متفاوت هستند.
* الگوهای نگرانی برش متقابل: الگوی [شاسی میکرو](https://microservices.io/patterns/microservice-chassis.html) سرویس و [پیکربندی خارجی](https://microservices.io/patterns/externalized-configuration.html)
* الگوهای تست: آزمون [جزء خدمات و](https://microservices.io/patterns/testing/service-component-test.html) خدمات [ادغام قرارداد تست](https://microservices.io/patterns/testing/service-integration-contract-test.html)
* [مدارشکن](https://microservices.io/patterns/reliability/circuit-breaker.html)
* [نشانه دسترسی](https://microservices.io/patterns/security/access-token.html)
* الگوهای مشاهده پذیری:
  + [لاگ جماعت](https://microservices.io/patterns/observability/application-logging.html)
  + [متریک های کاربرد](https://microservices.io/patterns/observability/application-metrics.html)
  + [ورود حسابرسی](https://microservices.io/patterns/observability/audit-logging.html)
  + [ردیابی توزیع شده](https://microservices.io/patterns/observability/distributed-tracing.html)
  + [ردیابی استثنا](https://microservices.io/patterns/observability/exception-tracking.html)
  + [بررسی سلامت API](https://microservices.io/patterns/observability/health-check-api.html)
  + [استقرارها و تغییرات ورود به سیستم](https://microservices.io/patterns/observability/log-deployments-and-changes.html)
* الگوهای UI:
  + [ترکیب قطعه صفحه سمت سرور](https://microservices.io/patterns/ui/server-side-page-fragment-composition.html)
  + [ترکیب رابط کاربر سمت مشتری](https://microservices.io/patterns/ui/client-side-ui-composition.html)

استفاده های شناخته شده

بیشتر وب سایت های در مقیاس بزرگ از جمله نت فلیوکس، آمازون و [eBay](http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf) از یک معماری یکپارچه به یک معماری میکروخوشه تکامل یافته اند.

نت فلیرز که یک سرویس جریان ویدئویی بسیار محبوب است که تا ۳۰٪ ترافیک اینترنت را بر خود دارد، دارای مقیاس بزرگ، معماری سرویس گرا است. آنها بیش از یک میلیارد تماس در روز به ویدئو خود را جریان API از بیش از 800 انواع مختلف دستگاه رسیدگی. هر API طرفداران تماس به طور متوسط از شش تماس برای backend خدمات.

Amazon.com در اصل معماری دو لایه ای داشت. به منظور مقیاس آنها به یک معماری سرویس گرا متشکل از صدها سرویس backend مهاجرت کردند. چندین برنامه کاربردی این سرویس ها را فراخوانی می کنند از جمله برنامه هایی که Amazon.com وب سایت و سرویس وب API را پیاده سازی می کنند. برنامه Amazon.com وب سایت تماس 100-150 خدمات برای دریافت داده هایی که برای ساخت یک صفحه وب استفاده می شود.

سایت حراج ebay.com معماری یکپارچه به یک معماری سرویس گرا تکامل یافته است. ردیف برنامه از چندین برنامه مستقل تشکیل شده است. هر برنامه کاربردی منطق کسب و کار را برای یک حوزه عملکرد خاص مانند خرید یا فروش پیاده سازی می کند. هر برنامه از انشعاب های محور ایکس استفاده می کند و برخی از برنامه های کاربردی مانند جستجو از تقسیمات محور Z استفاده می کنند. Ebay.com ترکیبی از پوسته پوسته شدن به سبک ایکس، Y و Z را به ردیف پایگاه داده اعمال می کند.

نمونه های متعدد [دیگری از شرکت ها با](https://microservices.io/articles/whoisusingmicroservices.html) استفاده از معماری میکرو سرویس وجود دارد.

معماری یکپارچه

چهار چوب

شما در حال توسعه یک برنامه سازمانی سمت سرور. باید از مشتریان مختلف از جمله مرورگرهای رومیزی، مرورگرهای تلفن همراه و برنامه های کاربردی بومی تلفن همراه پشتیبانی کند. این برنامه همچنین ممکن است API برای احزاب 3 به مصرف افشا. همچنین ممکن است با برنامه های کاربردی دیگر از طریق خدمات وب یا یک کارگزار پیام ادغام شود. برنامه رسیدگی به درخواست (درخواست HTTP و پیام) با اجرای منطق کسب و کار؛ دسترسی به یک پایگاه داده؛ تبادل پیام با سیستم های دیگر؛ و بازگشت پاسخ HTML / طولاني / XML. اجزای منطقی متناظر با حوزه های عملکردی مختلف کاربرد وجود دارد.

مشکل

معماری استقرار برنامه چیست

* یک تیم از توسعه دهندگان کار بر روی برنامه وجود دارد
* اعضای تیم جدید باید به سرعت مولد شوند
* برنامه باید آسان به درک و تغییر
* شما می خواهید به تمرین استقرار مداوم از برنامه
* شما باید نمونه های متعددی از برنامه را بر روی ماشین های متعدد اجرا به منظور برآورده کردن مقیاس پذیری و الزامات در دسترس بودن
* شما می خواهید برای استفاده از فن آوری های در حال ظهور (چارچوب ها، زبان های برنامه نویسی، و غیره)

راه حل

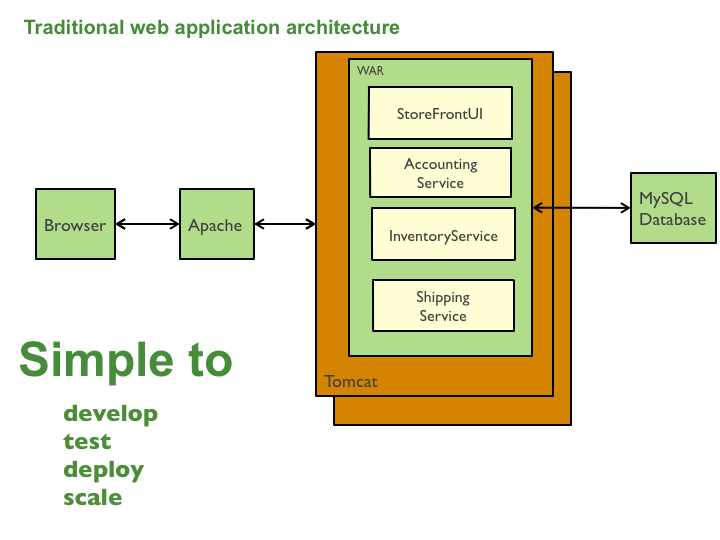
ساخت یک برنامه کاربردی با معماری یکپارچه. به عنوان مثال:

* یک فایل جنگ جاوا تک.
* سلسله مراتب دایرکتوری واحد از ریل یا کد NodeJS

مثال

بیایید تصور کنید که شما در حال ساخت یک برنامه تجارت الکترونیکی است که طول می کشد سفارشات از مشتریان، تایید موجودی و اعتبار در دسترس، و کشتی آنها را. این برنامه شامل اجزای متعددی از جمله StoreFrontUI، که پیاده سازی رابط کاربری، همراه با برخی از خدمات backend برای چک کردن اعتبار، حفظ موجودی و سفارشات حمل و نقل است.

این برنامه به عنوان یک کاربرد یکپارچه واحد مستقر می شود. به عنوان مثال، یک برنامه وب جاوا شامل یک فایل WAR واحد است که بر روی یک ظرف وب مانند Tomcat اجرا می شود. برنامه ریل شامل سلسله مراتب دایرکتوری واحد مستقر با استفاده از هر دو، به عنوان مثال، مسافر Phusion در آپاچی / Nginx یا JRuby در Tomcat. شما می توانید نمونه های متعددی از برنامه پشت ترازن بار به منظور مقیاس و بهبود در دسترس بودن اجرا کنید.



این راه حل دارای تعدادی مزایا است:

* ساده برای توسعه -- هدف از ابزار توسعه فعلی و IDEs این است که حمایت از توسعه برنامه های کاربردی یکپارچه
* ساده برای استقرار -- شما به سادگی نیاز به استقرار فایل جنگ (یا سلسله مراتب دایرکتوری) در زمان اجرا مناسب
* ساده به مقیاس -- شما می توانید برنامه را با اجرای کپی های متعدد از برنامه پشت ترازن بار مقیاس

با این حال، هنگامی که برنامه بزرگ می شود و تیم در اندازه رشد می کند، این رویکرد دارای تعدادی اشکالات است که به طور فزاینده ای قابل توجه می شود:

* پایه کد یکپارچه بزرگ توسعه دهندگان را مرعوب می کند، به ویژه پایه هایی که برای تیم جدید هستند. برنامه می تواند دشوار باشد به درک و تغییر. در نتیجه توسعه به طور معمول کند می شود. همچنین به دلیل اینکه مرزهای ماژول سخت وجود ندارد، مدولاریته در طول زمان از هم می پاشد. همچنین به این دلیل که درک چگونگی اجرای صحیح یک تغییر کیفیت کد در طول زمان کاهش می یابد، می تواند دشوار باشد. این یک مارپیچ رو به پایین است.
* بیش از حد IDE - بزرگتر پایه کد کندتر IDE و توسعه دهندگان کمتر مولد هستند.
* بیش از حد ظرف وب -- بزرگتر برنامه طولانی تر طول می کشد تا راه اندازی. این امر به دلیل هدر رفتن زمان در انتظار شروع ظرف تأثیر زیادی بر بهره وری توسعه دهنده داشت. همچنین تاثیر استقرار بیش از حد.
* استقرار مداوم دشوار است - یک برنامه یکپارچه بزرگ نیز مانعی برای استقرار های مکرر است. به منظور به روز رسانی یک جزء شما باید به توزیع مجدد کل برنامه. این کار وظایف پس زمینه (به عنوان مانند مشاغل کوارتز در یک برنامه جاوا) را قطع خواهد کرد، صرف نظر از اینکه آیا آنها تحت تاثیر تغییر قرار می گیرند، و احتمالا ً مشکلاتی ایجاد می کنند. همچنین این شانس وجود دارد که اجزایی که به روز نشده اند موفق به شروع درست نشند. در نتیجه خطر مرتبط با توزیع مجدد افزایش می یابد که به روز رسانی های مکرر را دلسرد می کند. این است که به خصوص یک مشکل برای توسعه دهندگان رابط کاربر, از آنجایی که آنها معمولا نیاز به تکرار به سرعت و دوباره به طور مکرر.
* پوسته پوسته شدن کاربرد می تواند دشوار باشد - یک معماری یکپارچه این است که تنها می تواند در یک بعد مقیاس کند. از یک سو می تواند با اجرای نسخه های بیشتری از برنامه، حجم تراکنش فزاینده ای را مقیاس کند. برخی ابرها حتی می توانند تعداد نمونه ها را به صورت پویا بر اساس بار تنظیم کنند. اما از سوی دیگر، این معماری نمی تواند با افزایش حجم داده مقیاس کند. هر کپی از نمونه برنامه به تمام داده ها دسترسی خواهد داشت، که باعث می شود caching کمتر موثر باشد و مصرف حافظه و ترافیک I/O را افزایش می دهد. همچنین، اجزای مختلف برنامه مورد نیاز منابع مختلف - یکی ممکن است پردازنده فشرده در حالی که دیگری ممکن است حافظه فشرده است. با یک معماری یکپارچه نمی توانیم هر جزء را به طور مستقل مقیاس کنیم
* مانع در برابر توسعه پوسته پوسته شدن - یک کاربرد یکپارچه نیز مانعی برای توسعه پوسته پوسته شدن است. هنگامی که برنامه می شود به اندازه خاصی مفید خود را به تقسیم سازمان مهندسی به تیم هایی که بر روی مناطق عملکردی خاص تمرکز. به عنوان مثال، ما ممکن است بخواهید به تیم UI، تیم حسابداری، تیم موجودی، و غیره. مشکل یک کاربرد یکپارچه این است که مانع از کار مستقل تیم ها می شود. تیم ها باید تلاش های توسعه ای و جا به جاهای مجدد خود را هماهنگ کنند. ایجاد تغییر و به روز رسانی تولید برای یک تیم بسیار دشوارتر است.
* نیاز به تعهد بلند مدت به پشته فن آوری -- معماری یکپارچه شما را مجبور به ازدواج با پشته فن آوری (و در برخی موارد ، به نسخه خاصی از آن فن آوری) شما در آغاز توسعه را انتخاب کرد. با یک برنامه یکپارچه، می تواند دشوار باشد به طور فزاینده اتخاذ یک فن آوری جدیدتر است. به عنوان مثال، بیایید تصور کنیم که شما JVM را انتخاب کرده اید. شما باید برخی از انتخاب زبان و همچنین جاوا شما می توانید زبان های JVM دیگر که در اینتر عمل سادگی با جاوا مانند Groovy و Scala استفاده کنید. اما اجزای نوشته شده به زبان های غیر JVM جایی در درون معماری یکپارچه شما ندارد. همچنین، اگر برنامه خود را با استفاده از چارچوب پلت فرم است که پس از آن منسوخ می شود و سپس آن را می تواند به چالش کشیدن به طور فزاینده مهاجرت برنامه به یک چارچوب جدیدتر و بهتر است. این امکان وجود دارد که به منظور اتخاذ یک چارچوب پلت فرم جدیدتر شما باید به بازنویسی کل برنامه، که یک تعهد مخاطره آمیز است.

**قدم اول**

استخراج entityوobject

**روش اول**

**استخراج خدمات میکرو از یک یکپارچه**

هنگامی که ما برای اولین بار شروع به dabbling در سیستم های توزیع شده، ما متوجه شدم که برخی از چیزهایی که ما باید به درک تنها از نظر تئوری. ما فهمیدم که چگونه مرزهای بین سرویس های خرد و اصول اساسی پشت توسعه، استقرار و مقیاس پذیری را طرح ریزی کنیم .

ما می توانم در مورد قوام و در دسترس بودن بالا برای روز ها بخوانید، رفتن را از طریق مطالعات موردی مختلف و نمونه. اما به عنوان کسی که تا به حال عمدتا ً روی برنامه های کاربردی یکپارچه کار می کرد، سوال ما باقی ماند.

چگونه می توانم یک سرویس را از یک برنامه یکپارچه استخراج کنیم .

میکرو سرویس ها یک الگوی معماری سطح بالا هستند، اما ویژگی ها در سطح کد کاربرد پژواک دارند. یکپارچه محکم همراه که در آن ما طق در هم تنیده است و مرزها وجود ندارد آسان نخواهد بود به refactor به microservices.

معماری نرم افزار بیش از استقرار و الگوهای ساختاری است. معماری شما شیوه فکر کردن شما در مورد طراحی را دیکته خواهد کرد. بازنویسی مجدد به میکرو سرویس ها بدون بازنویسی کامل غیرممکن خواهد بود مگر اینکه اصول اساسی را در پیاده سازی پیاده سازی کنیم.

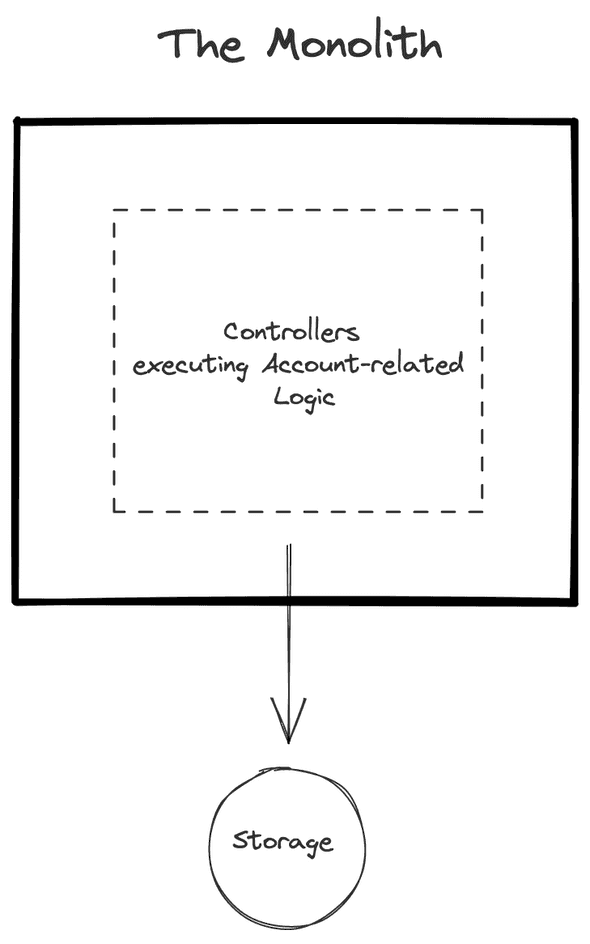
سفر به یک سیستم توزیع شده با تغییرات کد کوچک آغاز می شود. به طور خاص تر، قرار دادن مرزها، یکی از مفاهیم اصلی در میکرو سرویس ها است. هر سرویس باید بخشی از داما ه را «خود» کند. باید ذخیره سازی خاص خود را داشته باشد و از طریق نقاط پایانی (یا شکل دیگری از قرارداد) که ارائه می دهند با خدمات دیگر ارتباط برقرار کند.

**استخراج ما طق محکم زوج به میکروخوشایند**

استخراج یک میکرو سرویس از یک یکپارچه تنها در صورتی امکان پذیر است که مرزهایی را به شدت تعریف کرده باشد. مثال معمولی در این سناریو ما طق برای سفارشات و تحویل در یک برنامه تجارت الکترونیکی است. اما بیایید یک سناریو که در آن ما می خواهیم به استخراج ما طق عمیقا در مکان های متعدد در برنامه مانند ترجیحات کاربر و تجزیه و تحلیل کشف.

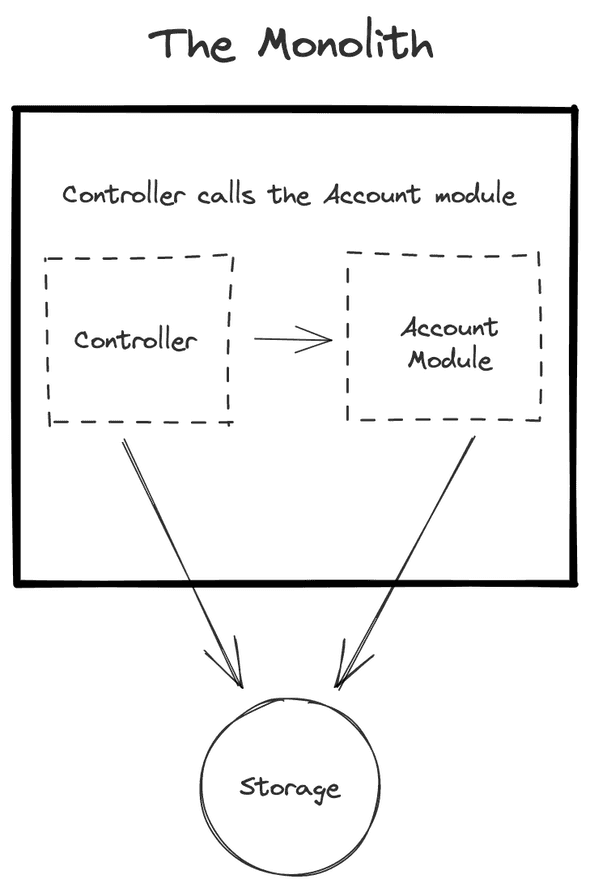
اغلب هنگامی که یک کاربر با یک محصول تعامل می کند، شما باید داده های مربوط به آنها را ذخیره و تغییر دهید. شما نیاز به ذخیره اطلاعات در مورد آخرین موارد مشاهده شده در یک فروشگاه آنلاین. شما آدرس تحویل خود را به روز رسانی زمانی که آنها را یک سفارش جدید و یا پیگیری از آخرین مکان های آنها را در یک وب سایت سفر جستجو کرده اند. لیست طولانی است.

ایده این است که شما بسیاری از مکان ها در برنامه خود را که در آن شما می خواهید برای لمس شی کاربر داشته باشد. این قابلیت اغلب در تمام طول برنامه پاشیده شده است، و شما می توانید تماس به پایگاه داده در کنترل کننده ها و ماژول های متعدد پیدا کنیم .

[](https://alexkondov.com/static/a3ea4f54c1047b512473bcf63b73c996/b2982/2.1.png)

ما شناسایی کرده ایم که این ما طق مربوط به حساب کاندیدای خوبی برای اولین میکرو سرویس ماست. اما قبل از اینکه حتی به زیرساخت ها فکر کنیم باید آن را در یکپارچه دو برابر کنیم. ما می خواهیم تمام این قابلیت ها را پشت یک انتزاع قرار بدهیم.

با گروه بندی آن را در یک ماژول ما پنهان کردن تمام جزئیات در اطراف چگونه حساب ها به روز شده (نوع ذخیره سازی، اطلاعات پایگاه داده، و غیره). ما یک قرارداد ایجاد می کنیم و کنترلی بر API داریم که ماژول حساب افشا می کند. این صرفا یک

تغییر در سطح کد در آماده سازی ما است. [](https://alexkondov.com/static/a009362a5462efd08a81deb30592e4f4/b2982/2.2.png)

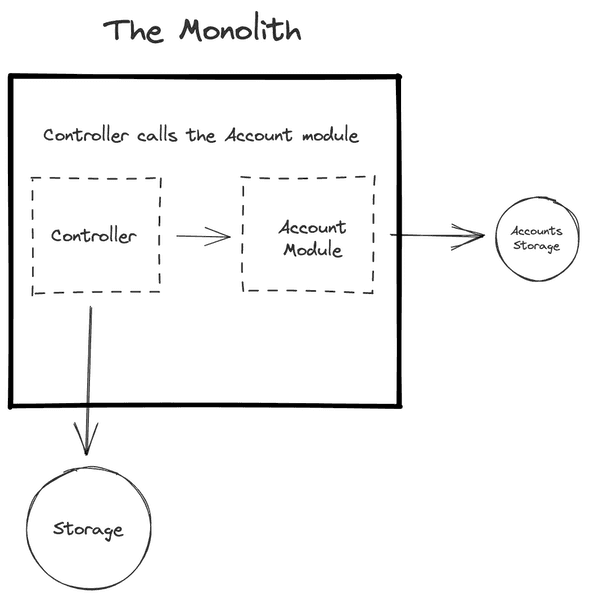
در سطح کد، این به معنای پیچیدن قابلیت در یک شیء و فراخوانی توابع آن است.}

یکی از سوالات فشار آور ترین ما زمانی که ما برای اولین بار شروع به برخورد با microservices بود آنچه اتفاق می افتد با کلید های خارجی و روابط هنگامی که ما تقسیم داده ها در سراسر پایگاه داده های مختلف. گاهی اوقات هیچ روابطی برای در نظر گرفتن وجود نخواهد داشت، اما اگر مجبور بودیم، دو گزینه داریم - تکراری یا به شناسه های ما حصر به فرد جهانی تکیه می کنیم.

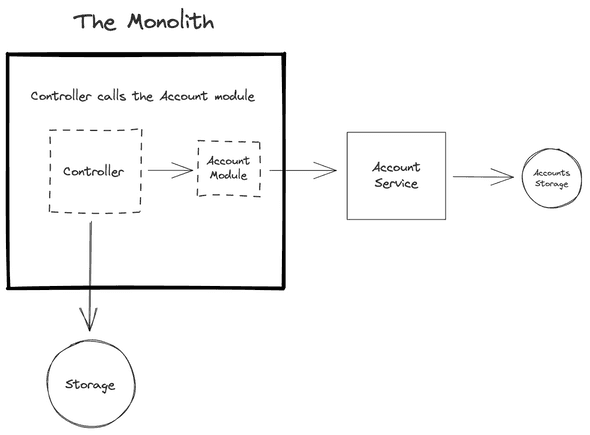
نحوه رسیدگی به تکثیر بستگی به ذخیره سازی دارد که از آن استفاده می کنیم. پایگاه داده به این معنی است که ما نیاز به ذخیره اشیاء تودرتو و به روز رسانی آنها را هر زمان که داده ها تغییر می کند. در یک پایگاه داده SQL، ما می توانیم کل جداول را تکراری کنیم تا روابط را در چک نگه داریم و خودمان همگام سازی را مدیریت کنیم.

اما لخت در ذهن است که هر چه تکثیر بیشتری برای مدیریت داشته باشید، پیچیدگی بیشتری برای مقابله با آن خواهید داشت. تکثیر اطلاعات بیش از حد ممکن است به این معنی باشد که فروشگاه یک سرویس فراتر از مسئولیت های خود رشد کرده است.

متناوبا، ما می توانیم برخی از شناسه های ما حصر به فرد جهانی مانند UUIDs استفاده کردیم و تکیه بر آنها برای جمع آوری داده های مرتبط بین خدمات. این رویکرد مورد علاقه ما است، و شانس برای گرفتن یک شناسه تکراری به اندازه کافی باریک است برای ما به مراقبت نیست.

[](https://alexkondov.com/static/b3f01c774a835abbad06035cd7b7c5e2/b6529/2.3.png)

شما ممکن است تعجب اگر این پیچیدگی سیستم ما را بیش از حد بالا نمی برد. تا یک لحظه پیش، ما فقط یک ما طق محکم همراه داشتیم. يه کم کثيف، شايد ولي هيچي ارزش خوندن سفيدپرها رو نداره. حالا ناگهان باید با قوام داده ها و تمام مشکلات اطراف آن مقابله کنیم. یک شکست را تصور کردیم که فروشگاه های داده های مختلف را از همگام سازی خارج می کند. چالش های توزیع بی اهمیت نیست، بنابراین مطمئن شوید که برخورد با آن ها پربارتر از مقابله با مشکلات یک یکپارچه است. برای کسب استقلال در کار، استقرار و مقیاس باید چیزی را قربانی کنیم. همش يه معامله ست، همونطور که گفته ميشه. به همین دلیل به مردم می گویم به جای مزایایی که می خواهند داشته باشند، مشکلاتی را که می توانند با آن کنار بییند انتخاب کنند. در حال حاضر ما یک مرز جامد خوب است که هیچ جزئیات نشت نیست. گام بعدی اجرای این ما طق در یک سرویس جداگانه و استقرار آن است. همراه با ما طق، ما باید یک لایه حمل و نقل اضافه کنیم - چه HTTP باشد و چه gRPC و چه چیز دیگری. ما باید چک های بهداشتی و ورود به سیستم را اضافه کنیم. این سرویس به ذخیره ای که در مرحله قبل ایجاد کردیم متصل خواهد شد. مهم این است که نوعی نظارت را فراموش نکنیم . تا به حال، ما یک یکپارچه واحد داشتیم و احتمالا راه هایی برای پیدا کردن اینکه آیا Fail کرده است یا نه، داشتیم. حالا که سرویس دیگری داریم نباید اجازه دهیم در سکوت Fail کند

[](https://alexkondov.com/static/e92f181b100a672032ec5a03c6c89bce/869d7/2.4.png).

هنگامی که میکرو سرویس جدید در حال اجرا است، ما می توانیم ما طق در ماژول به روز رسانی به تماس های HTTP به سرویس جدید ما به جای اجرای ما طق خود را. چرا تماس های HTTP را به طور مستقیم در کنترل کننده و خلاص شدن از شر این ماژول، شما ممکن است بپرسید؟ از آنجا که هنوز هم ممکن است ویژگی هایی وجود دارد که ما می خواهیم به انتزاعی از بقیه برنامه ما ممکن است نیاز به ساخت هدر، راه اندازی احراز هویت، و باز کردن پاسخ بازگشته در یک قالب دوستانه. درخواست های شبکه همیشه نیاز به دست زدن به خطا دارند که لازم نیست در جریان برنامه اصلی مدیریت شود، به خصوص اگر تماس از نوع " و فراموش کردن" باشد. این باز می شود درب برای بسیاری از تصمیم گیری های طراحی جالب است. ما فقط می توانیم خطا را وارد سیستم کنیم و آن را فراموش کنیم. ما می توانیم انتخاب کنیم که یک شی پاسخ خالی را به جای تهی برگردانیم. ما همچنین می توانیم چیزی وارد سیستم و پرتاب یک خطای برنامه سفارشی به توسط تماس گیرنده رسیدگی می شود. اما در همه موارد مقداری پیچیدگی را در بر خواهیم گرفت. سپس اگر سرویس حساب ها تغییر شکستنی به یکی از نقاط پایانی آن ایجاد کند، باید آن را در یک مکان به روز کنیم.

**استخراج ماژول های در حال حاضر تاسیس**

در برخی موارد دیگر، شما ممکن است مسئولیت های داما ه متعدد در همان یکپارچه داشته باشد. آنها حتی ممکن است متعلق به توسعه دهندگان یا تیم های مختلف باشند. با این حال، همه آنها در یک پایگاه کد زندگی می کنند و مردم همچنان پا بر روی پا یکدیگر.

اصل استخراج چنین ماژولی یکسان است - با مرزها شروع کنیم . مطمئن شوید که ماژول شما می خواهید برای استخراج شامل همه چیز آن نیاز به کار -- توابع ابزار ، ما طق کسب و کار و دیگر نهادهای مشترک است. تکراری کسانی که بین ماژول ها به اشتراک گذاشته شده است. سپس ذخیره سازی آن را جدا کرده و آن را به عنوان یک سرویس جداگانه مستقر کنیم .

دروازه API نیز به ما می دهد متمرکز تعادل بار، محدود کردن نرخ، مجوز، و مزایای دیگر. همانطور که شما ممکن است حس کرده اند، ما یک بار دیگر به شعاع دید پیچیدگی بالا رفت.

**پیچیدگی عملیاتی و سازمانی**

شما باید پیچیدگی اجرای یک معماری میکروخوشه را در تولید در نظر ب گیرید. برای به دست آوردن استقرار مستقل، پوسته پوسته شدن، و توسعه، شما تمام مشکلات سیستم های توزیع شده را به دست خواهید آورد بررسی غیر فنی این است که آیا ساختار سازمان شما می تواند چنین معماری را اداره کند یا نه. در پایان قانون کانوی همیشه برنده می شود. بنابراین اگر شما یک تیم از چهار توسعه دهنده، شانس در حال اجرا یکپارچه آسان تر خواهد بود. اگر شما یک زن و شوهر از تیم در تلاش برای کار بر روی همان codebase، سپس تجزیه آن احتمالا نتایج بهتری عملکرد. به عنوان یک قاعده کلی، جریان ارتباطی در تیم های شما باید در جریان ارتباطی سیستم های شما ما عکس شود.

**کد در فایل ضمیمه موجود استExt.py**

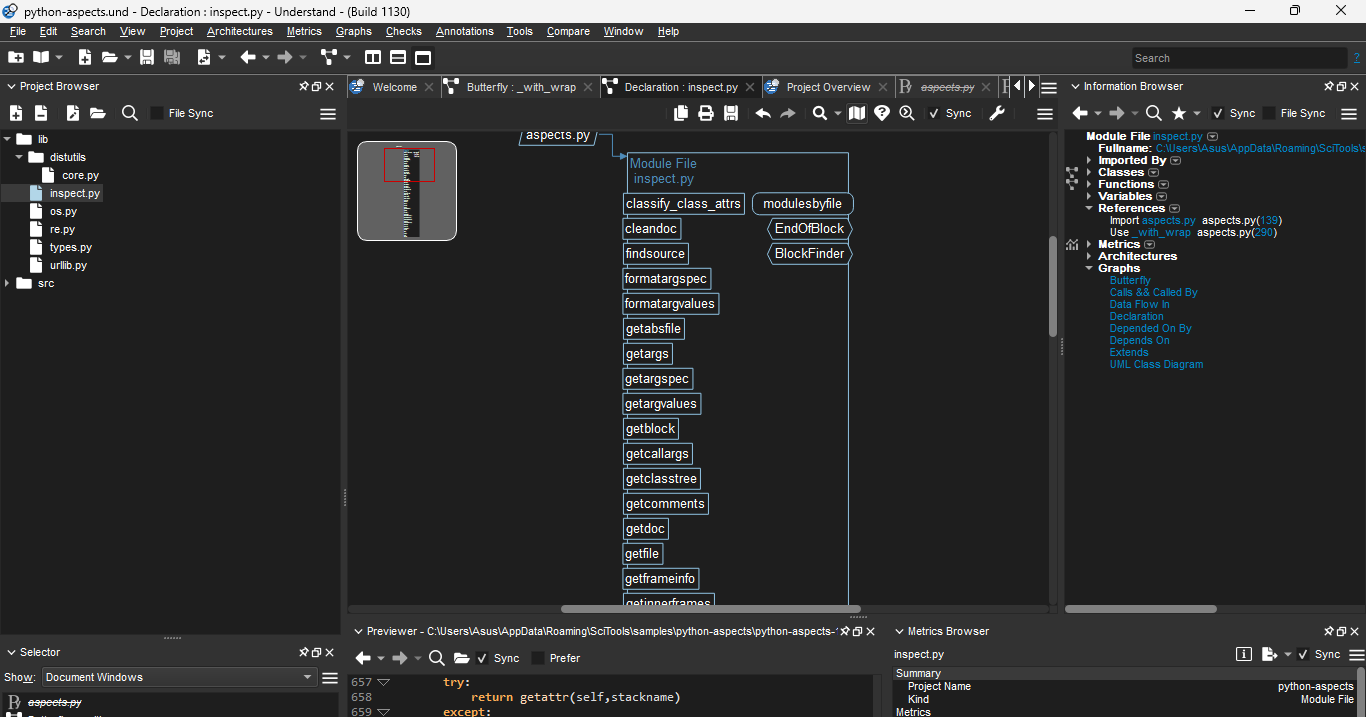
**مرحله دوم روش اول**

**Insideالگوریتم هایی که در folderالگوریتم هست را به فایل mainفراخوانی کردیم**

**روش دوم**

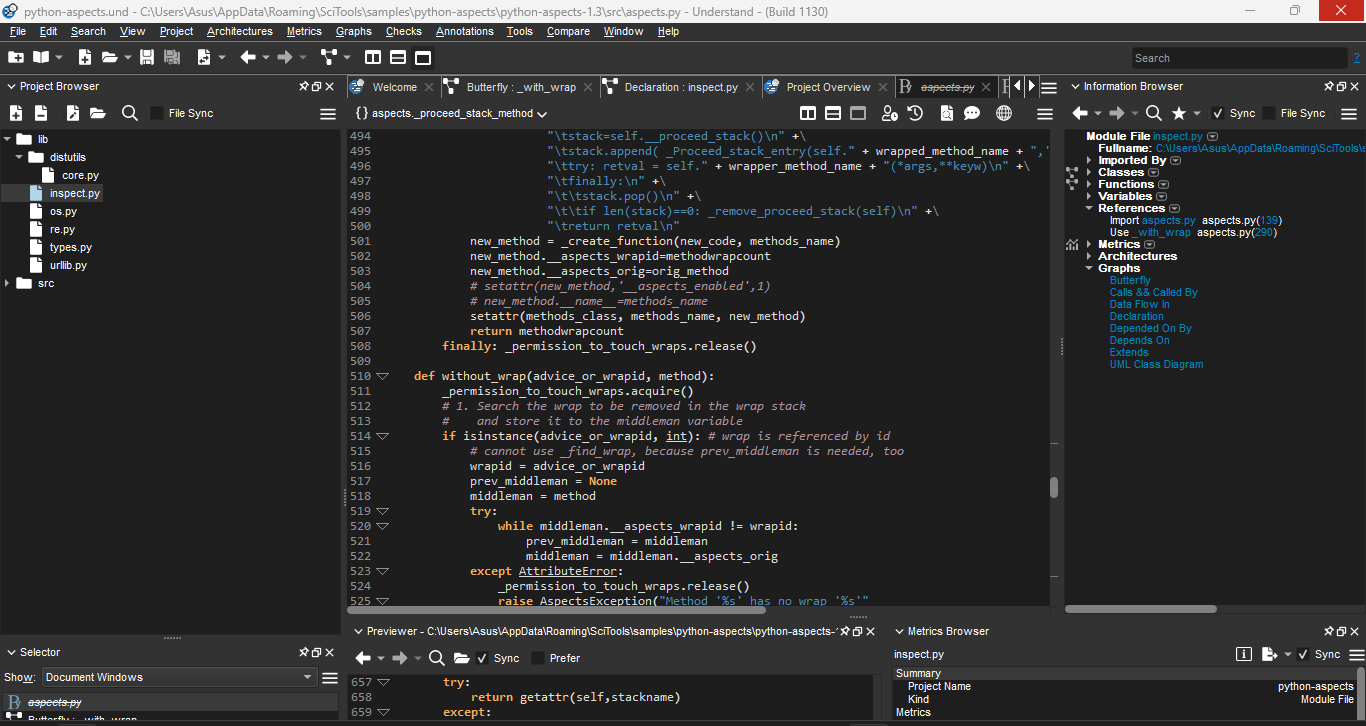
**استفاده از نرم افزار understanding scitools**

به وسیله این نرم افزار موجودیت ها روابط و منابع را استخراج می کنیم

این نرم افزار قادر است توسط یک سری از پترنها به ما این امکان را بدهد گراف دانشی از روی سیستم منولوتیک خود داشته باشیم 

در مرحله بعد قسمتی از گراف با موجودیت ها را extracting

میکنیم این نرم افزار قابلیت دسترسی به سیستم های پروگرام و سمپل کد را هم داراست

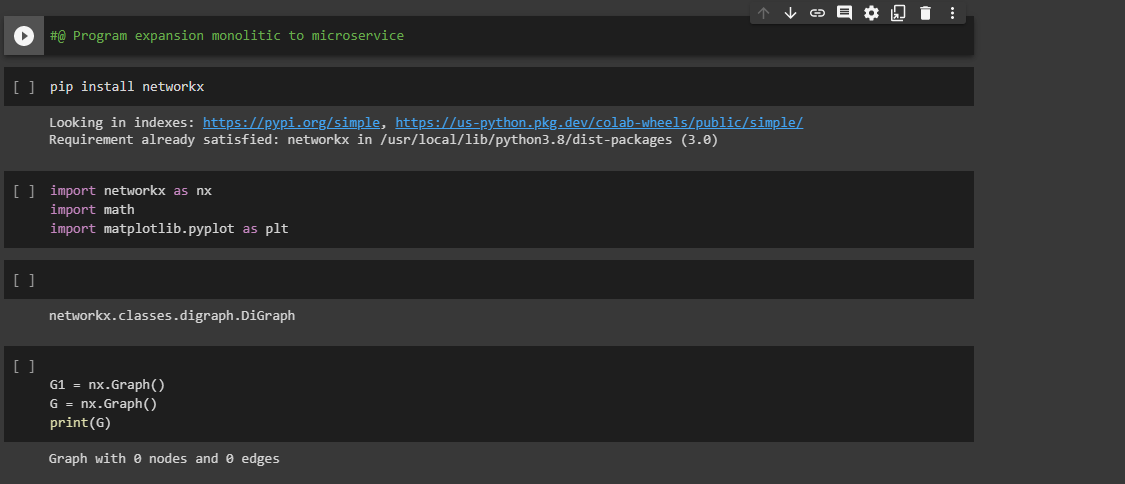


در مرحله بعد از کتابخانه networkxبرای استفاده از الگوریتم ها برای عملیات بر روی گراف کمک میگیریم

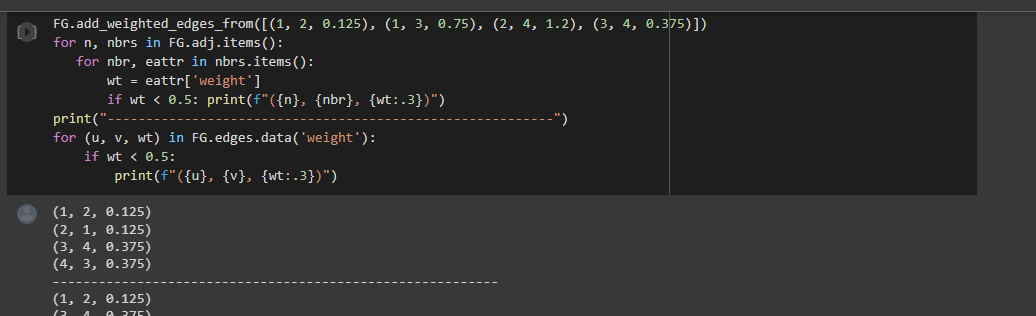
* ساختارهای داده ای برای گراف ها، دی گراف ها، و چند گراف ها
* بسیاری از الگوریتم های گراف استاندارد
* ساختار شبکه و اقدامات تجزیه و تحلیل
* ژنراتورها برای گراف های کلاسیک، گراف های تصادفی، و شبکه های مصنوعی
* گره ها می توانند "هر چیزی" (مانند، متن، تصاویر، سوابق XML)
* لبه ها می توانند داده های دلخواه را نگه دارند (به عنوان مانند، وزن ها، سری های زمانی)
* منبع باز [3 بند مجوز BSD](https://raw.githubusercontent.com/networkx/networkx/master/LICENSE.txt)
* خوب تست شده با بیش از 90٪ پوشش کد
* مزایای اضافی از پایتون شامل نمونه برداری سریع، آسان برای تدریس، و چند پلت فرم

استفاده میکنیم نرم افزار سابق به ما یک خروجی .dotمی دهد که ما ان را با colabبارگذاری و از از الگوریتم های داخل کتاب خانه network انها را ضریب وابستگی و ارتباطشان را محاسبه می کنیم

و از روش خروجی انقدر این کار را تکرار کرده تا به بهترین حالت برسیم



دراین مرحله تمام نود های مرتبط به یک دیگر میشوند



در ابتدا تمام نود های خوانده شده را در گراف مان اد کردیم



لینک کد

<https://colab.research.google.com/drive/1eF94Y0shFqvKx8RcX93jNBvNiCPz8RMs?usp=sharing>

لازم به ذکراست روش اول به دلیل الگوریتم نا پذیر بودن داکر به بن بست خورد

و روش دوم هم به دلیل نداشتن برچسب نود ها ناقص است