4th International Conference on Web Research
April 25-26, 2018; Tehran, IRAN
ICWR2018
کنفرانس بین المللی وب پژوهی
چهارمین
۵ و ۶ اردیبهشت ۱۳۹۷؛ تهران، ایران

Migration to the Microservice Architecture Style Comparing to the Monolithic Architecture Style with Forwarding Challenges Analysis

Mohammad Heydari[1], Fereidoon Shams Aliee[2]

[1] MSc Student in IT Engineering, Department of Computer Science and Engineering,

Shahid Beheshti University, Tehran, Iran

Moh.heydari@mail.sbu.ac.ir

[2] Associate Professor, Department of Computer Science and Engineering,

Shahid Beheshti University, Tehran, Iran

F_Shams@Sbu.ac.ir

Abstract

In this paper, first we get familiar with key terms on Software Architecture area. Then we mention principal rules on microservices architecture. After that, we show general interest in microservices architecture comparing to the Service Oriented Architecture by fetch the Google Trend Service Statistics. After that, we discuss about Monolithic Architecture main issues and Comparing microservices architecture with Service Oriented Architecture. In the most important section of the article, we hit Technical and experimental result that extracted from 21 notable developers and expert in software industry in migration causes to the microservice architecture which most important one named the Enhanced maintenance in large scale. We will introduce Jolie Programming language which natively support Microservice Architecture. Microservice forward challenges will be describe and Container Technology on the Shoulder of Docker Open Source Platform will be introduce. Container is as a competitor for Hypervisor and we will hear of them more in future. most important design pattern for microservices will be describe with solution.

Keywords: Software Architecture, Microservice, Monolithic, Docker, Container

4th International Conference on Web Research
April 25-26, 2018; Tehran, IRAN
ICWR2018
کنفرانس بین المللی وب پژوهی
چهارمین
۵ و ۶ اردیبهشت ۱۳۹۷؛ تهران، ایران

مهاجرت به سبک معماری میکرو سرویس ها در سیستم های نرم افزاری در قیاس با سبک معماری یکپارچه به همراه تحلیل چالش های پیش رو

محمد حیدری [۱] فریدون شمس علیئی [۲]

[۱] دانشجوی مقطع کارشناسی ارشد مهندسی فناوری اطلاعات، دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی تهران

Moh.heydari@mail.sbu.ac.ir

[۲] دانشیار دانشکده دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی تهران

F_Shams@Sbu.ac.ir

چکیده

در نوشتاری پیش رو ابتدا تعاریف اصطلاحات کلیدی درس حوزه معماری نرم افزار را بررسی می کنیم و سپس به اصول کلیدی معماری میکروسرویس ها اشاره خواهیم کرد. سپس با واکشی آمار جست و جوی عبارت "Microservice" در سطح موتور جست و جوی Google نرخ صعودی افزایش میل و رغبت به کنکاش در این سبک معماری را نسبت به معماری سرویس گرایی بررسی می کنیم. در ادامه مشکلات معماری یکپارچه را نام می بریم و تفاوت معماری میکروسرویس ها با معماری سرویس گرا را تحلیل می کنیم. در ادامه در مهمترین بخش علل مهاجرت به معماری میکروسرویس ها را با بررسی نتایج تجربی ۲۱ متخصص حوزه صنعت نرم افزار را بررسی کرده که مهمترین عامل را می توان در بهبود نگهداری در مقیاس وسیع دانست. زبان برنامه نویسی Jolie را معرفی خواهیم کرد، زبانی که بصورت کاملا Native از این نوع معماری پشتیبانی می کند. در اثنای کار چالش های میکروسرویس ها بر خواهیم شمرد و فناوری Container را بر بستر متن باز Docker معرفی خواهیم کرد که بعنوان رقیب اصلی Hypervisor ها مطرح شده است و آینده روشنی را با توجه به سبُکی و کاهش پیچیدگی که دارد می توان برایشان تصور کرد.. به ۹ الگوی های طراحی میکروسرویس ها به صورت موردی اشاره خواهیم کرد و سپس در پایان در بخش نتیجه گیری به درسهایی که معماری میکروسرویس ها می بایست از نقاط ضعف معماری سرویس گرا بعنوان یک عبرت هیچ گاه فراموش نکند اشاره خواهیم داشت.

کلیدواژه ها : معماری نرم افزار ، میکروسرویس، سبک معماری، یکپارچه، کانتینر، داکر

۱ – مقدمه

معماری میکروسرویس ها سبک و سیاقی از معماری است که از شیوه تفکر مبتنی بر سرویس گرایی الهام گرفته شده است. اصطلاح میکروسرویس در جست و جوهای اخیر در رابطه با اصطلاح "میکروسرویس ها" در سطح اینترنت نشان می دهد که بیشترین جست و جوها مبتنی بر فناوری هستند. این مطلب عنوان می کند که دوران جست و جوهای عامه نظیر عبارت "میکروسرویس چیست ؟ " به سرآمده است. نه فقط فروشندگان نرم افزاری نظیر مایکروسافت یا آی بی ام که تولیدکنندگان محتوی نظیر Netflix و Amazon نیز از معماری میکروسرویس ها در محصولات خود استفاده می کنند.[۱]

۱-۱ تعاریف اصطلاحات کلیدی در مقاله

معماری : یک چارچوب منسجم و ساختاری است که ارتباط اجزای یک سیستم را بصورت رسمی و دقیق را برقرار می کند.

نرم افزار : به مجموعه ای از دستورات و داده هایی که گفته می شود که به منظور رسیدن به هدفی خاص برای اجرا در سیستم های رایانه ای طراحی شده اند.

معماری نرم افزار : به فرآیند تعریف یک راهکار ساختاری که تمامی نیازمندی های عملیاتی و فنی نرم افزار را برطرف می سازد می گویند که به بهبود ویژگی های کیفی نرم افزار از جمله امنیت و عملکرد می پردازد.[Microsoft]

در واقع معماری نرم افزار توصیفی از زیرسیستم ها و مولفه های یک سیستم نرم افزاری و ارتباطات بین آنهاست.

مولفه : یک بخش کپسوله شده از یک سیستم نرم افزاری است که دارای یک رابط [i] است و به نگهداری سازه بلوک [ii] های ساختار یک سیستم می پردازد. در سطح برنامه نویسی یک مولفه می تواند در قالب یک ماژول، کلاس، شی یا مجموعه ای توابع مرتبط نمایش یابد.[۲]

سبک معماری : توصیفی از انواع مولفه ها و توپولوژی آنهاست. همچنین به تشریح الگوی داده و کنترل تعامل بین مولفه ها می پردازد و توصیفی غیررسمی از مزایا و معایب استفاده از یک سبک بخصوص در معماری است.

تعریفی هم از را با هم بررسی می کنیم : "سبک معماری مجموعه ای منظم و هماهنگ از قیدهایی است که نقش ها و ویژگی های عناصر معمارگونه را محدود می کند و ارتباط بین این عناصر را با هر معماری که مطابق با آن سبک باشد را ممکن می سازد".[۳]

در همین اواخر میکروسرویس ها به منظور شکستن معماری برنامه های کاربردی به سرویس های قابل استقرار به صورت مستقل که می توانند به سرعت در هر گونه منبع زیرساختی که نیاز داشته باشند استقرار پیدا کنند مطرح شدند.[۴] اصطلاح میکروسرویس به طور گسترده از مارچ ۲۰۱۲ مطرح شد و به مجموعه ای سرویس های نسبتا کوچک ، پایدار ، مجرد و خودمختار اشاره داشت که بطور مستقل با یک هدف مشخص و از پیش تعیین شده استقرار می یابند. میکروسرویس ها وابسته به هیچ زبان برنامه نویسی نیستند و در اصطلاح معروف Language-agnostic هستند و می توانند توسط تیم های توسعه گوناگون با سبک و سیاق خاص خودشان مورد استفاده قرار گیرند. [۵] مثلا تیمی که سمت سرور را توسط تکنولوژی Node Js کار می کند می تواند با تیمی که سمت سرور را توسط PHP کد میزند در راستای پیاده سازی معماری میکروسرویس ها همکاری کند و هیچ مشکلی پیش نخواهد آمد. از آنجایی که میکروسرویس ها نقطه مقابل معماری یکپارچه هستند ابتدا بایستی تعریفی از میکروسرویس ها و یکپارچگی [iii] ارائه کنیم .

تعریف ۱ (میکروسرویس ها) : یک پردازش منسجم و مجزی است که از طریق پیام ها با هم ارتباط برقرار می کنند.[۶]

تعریف ۱-۱ : میکروسرویس برنامه ای کوچک است که می تواند به صورت مستقل استقرار یابد ، مقیاس پذیر باشد ، تست شود و تنها یک وظیفه منفرد خواهد داشت. میکروسرویس تنها و تنها یک کار انجام می دهد که به سادگی قابل درک خواهد بود. این وظیفه منفرد می تواند یک نیاز وظیفه مندی باشد یا یک نیاز غیر وظیفه مندی یا حتی یک نیازمندی چندمحوری [iv] . یک مثال ساده می تواند پردازشگر صف باشد ، چیزی که وظیفه خواندن یک پیام از صف را دارد. [۷]

تعریف ۲-۱ : میکروسرویس یک سرویس سبک وزن و مستقل است یک کار را انجام می دهد و با دیگر سرویس های مشابه از طریق یک رابط خوش تعریف ارتباط برقرار می کند.[۸]

تعریف ۲ (یکپارچگی) : برنامه ای نرم افزاری تشکیل شده از ماژول هایی که نمی توانند به طور مجزی و مستقل اجرا شوند. این عدم توانایی،بکارگیری معماری یکپارچه را در محیط های توزیع شده را بدون وجود چارچوب با مشکل روبرو می کند.[۶]

تعریف ۳ – معماری میکروسرویس ها : معماری میکروسرویس، برنامه ای توزیع شده است که تمامی ماژول هایش میکروسرویس هستند.

تعریف ۴ – مفهوم اتصال سست : یک کلاس مبتنی بر اتصال سست می تواند به صورت مجزا از دیگر کلاس های (پیوسته) آزمایش شود. این مفهوم در مقابل اتصال کامل [v] قرار دارد و بدین معناست که شما می توانید یک سرویس را به صورت مجزی از سایر سرویس های مربوطه آزمایش کنید بصورتی که تغییری در سایر سرویس ها ایجاد نشود. اگر شما مجموعه ای از سرویس های کوچکی دارید که باید با هم به صورت همزمان بروزرسانی شوند باید بگویم آنها میکروسرویس نیستند، چون که مبتنی بر اتصال سست نیستند.[۹]

هر میکروسرویس می تواند با دیگر میکروسرویس ها هماهنگ شده ، تجمیع شود و مورد استفاده مجدد قرار گیرد. [۱۰] این رویکرد چند مزیت را با خود به همراه دارد: مهمترین مساله اینکه مدیریت مولفه های سیستم به سادگی صورت می گیرد ، هزینه توسعه ، نگهداری و استقرار توزیع نرم افزار کاهش می یابد.

این روزها مهاجرت به سبک و سیاق میکروسرویس ها امری حساس و خطیر تلقی می شود. بسیاری از شرکت ها و سازمان های عظیم مشغول ریفکتورینگ سیستم های سمت سرور خود هستند تا با آسانی های پارادایم جدید میکروسرویس ها خود را تطبیق دهند در حالی که سازمان هایی هم هستند که از همان ابتدا توسعه نرم افزاری خود را مطابق با معماری میکروسرویس ها بنا نهادند. [۱۱]

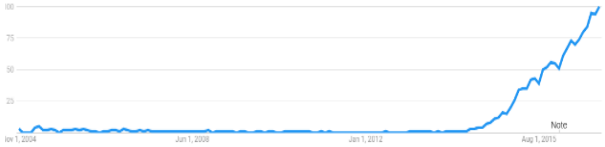
۲ – سه اصل کلیدی معماری میکروسرویس ها

معماری میکروسرویس ها[۶] بر سه اصل ساده استوار است:

- **محدوده مرزی** [vi] **:** این اصطلاح برای بار نخست توسط Eric Evan در اثر معروفش با نام طراحی مدل رانه بکار گرفته شد.[۱۲] و به یکی از ویژگی های کلیدی معماری میکروسرویس ها اشاره دارد. : تمرکز بر روی قابلیت های کسب و کار . ویژگی های مرتبط درون یک قابلیت کسب و کار ترکیب و سپس نهادینه می شوند و در انتها **به** عنوان یک سرویس پیاده سازی می شوند.
- **اندازه :** اندازه یک مفهوم حیاتی برای میکروسرویس هاست و مزایای عمده ای را در ارتباط با قابلیت نگهداری و توسعه سرویس ها برای ما به ارمغان می آورد. استفاده ایده آل از معماری میکروسرویس ها چنین است که اگر یک سرویس خیلی بزرگ است باید به دو یا چند سرویس کوچکتر پالایش شود. بدین ترتیب حفظ دانه بندی [vii] و نگهداری بر ارائه تنها یک قابلیت کسب و کار واحد متمرکز است.
- **استقلال :** این مفهوم حاکی اتصال سست [viii] و انسجام بالا در میکروسرویس ها هست بدین صورت که هر سرویس در معماری میکروسرویس ها در عمل مستقل از دیگری است و تنها راه ارتباط سرویس ها از طریق اینترفیس های منتشر شده است. [۱۱]

۲-۱ اما چرا تمایل به استفاده از میکروسرویس ها و مهاجرت به این سبک معماری سرویس گرا تا این حد رواج یافته است ؟

با نگاهی به آمار جست و جوهای در سطح گوگل رشد گرایش به سمت این سبک معماری را بررسی می کنیم :

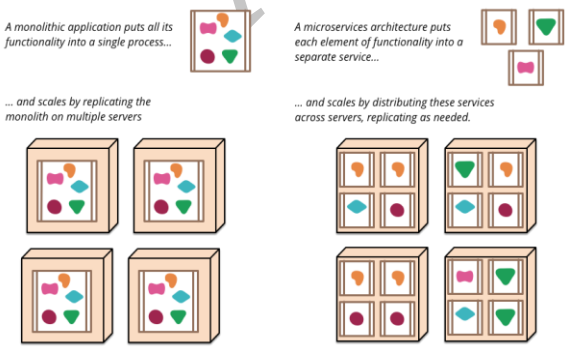


شکل (۱) : بررسی جستجوی میکروسرویس ها در سطح گوگل [1]

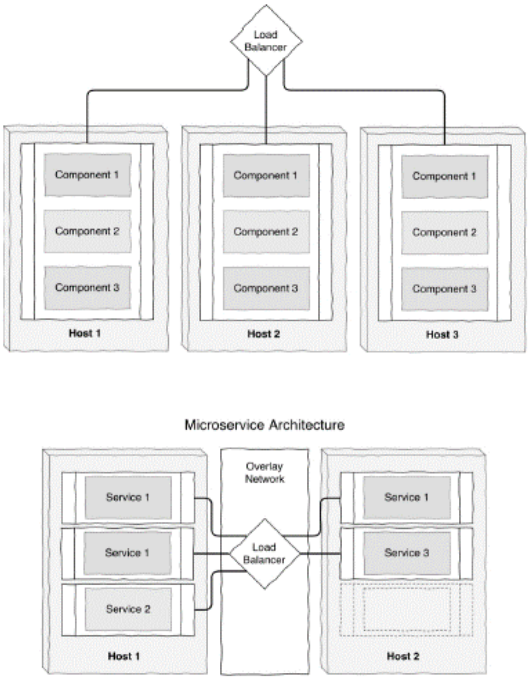
- جست و جو ها در رابطه با خاصه کلمه Microservices از مارچ ۲۰۱۴ رشد صعودی به خود گرفته است و در نوامبر ۲۰۱۶ روند صعودی خود را با شیب بسیار زیاد ادامه داده است.
- یکی از دلایل تمایل به مهاجرت به سبک و سیاق معماری میکروسرویس ها برقراری ارتباط سرویس ها توسط مکانیزم های سبک وزن که غالبا HTTP هست می باشد. [۱۳] مزیت دیگر اینکه مدیریت مرکزی هر یک از این سرویس ها به نوبه ی خود یک سرویس مجزی است و تداخلی رخ نخواهد داد و می تواند توسط زبان های برنامه نویسی مختلف نوشته شود و مدل داده خاص خودش را داشته باشد. اما در معماری یکپارچه مساله به شکلی دیگر است : در رابطه با یک برنامه تحت وب که به زبان جاوا نوشته شده است می توانید یک فایل WAR را تصور کنید. اگر بصورت ظاهری به ساختار برنامه نگاه کنیم از سرویس های و مولفه های مختلفی تشکیل شده است اما به عنوان یک راهکار واحد[ix] استقرار خواهد یافت. درست است که برای مقیاس پذیری می توانیم کپی های مختلفی از برنامه را اجرا کنیم اما به هر حال آنها یکسانند و شبیه به هم. پس چه مزیتی دارند ؟ جدای از اینکه برنامه بسیار حجیم و سنگین خواهد شد اما توسعه آن آسان تر است. بدون شک استقرار برنامه آسان تر خواهد بود و این بزرگترین مزیت راهکار یکپارچگی است. مسیر مقیاس پذیری هم بسیار روشن خواهد بود ؛ چرا که ما می توانیم کپی های مختلفی از برنامه را در پشت یک Load Balancer اجرا کنیم. اما در این بین مشکلاتی [۸] هم گریبان گیر ما خواهد شد.

۳ – مشکلات معماری یکپارچه

- **اول اینکه فهم و تغییر برنامه سخت خواهد شد** ، این مشکل وقتی بیشتر به چشم میخورد که برنامه بزرگتر شود. با رشد برنامه ،افزودن توسعه دهندگان جدید و جابجایی اعضای تیم مشکل خواهد بود.
- **دوم اینکه حجم زیاد کد برنامه ، بهره وری را کاهش می دهد** ، در نتیجه کیفیت فنی کدینگ کاهش می یابد و خاصیت اصلی پیمانه ای بودن مورد فرسایش قرار می گیرد. چالش اصلی برنامه های مبتنی بر یکپارچگی اینست که اعضای تیم را از کار کردن بصورت مستقل منع می کند و تمام تیم می بایست امر توسعه و استقرار مجدد را بصورت هماهنگ با هم انجام دهند و همین مساله باعث افت سرعت خواهد شد.
- **سوم اینکه توسعه مداوم مشکل خواهد شد.** برنامه های یکپارچه مانعی برای به روزسانی های متنوع و پی در پی خواهند بود. بدین صورت که اگر بخواهیم چند مولفه کوچک را بروزرسانی کنیم باید کل برنامه را بصورت یکجا مجددا استقرار دهیم.
- **چهارم اینکه مقیاس پذیری[x] برنامه با سختی روبرو خواهد شد.** چرا که برنامه یکپارچه تنها در یک بعد مقیاس داده می شود. بدین صورت که ما می توانیم حجم تبادلات[xi] میان برنامه را از طریق اجرای کپی های متعدد افزایش دهیم اما از طرفی دیگر این نوع معماری نمی تواند توسط افزایش حجم داده ها مقیاس پذیر باشد. علت این است که هر نمونه از کپی های برنامه به تمام داده ها دسترسی دارد و همین مساله سبب افت تاثیرگذاری عملیات ذخیره سازی می شود. از سویی دیگر افزایش مصرف حافظه[xi i] و به دنبالش افزایش ترافیک ورودی و خروجی برنامه را به همراه خواهد داشت. از این حیث در معماری یکپارچه مقیاس پذیری مولفه ها بصورت مجزی غیر ممکن خواهد بود.
- اما از آنجایی که میکروسرویس ها به صورت مجزی از یکدیگر توسعه داده شده و استقرار می یابند توسط پردازش های مجزی اجرا می شوند در نتیجه دارای مانیتورینگ و مقایس پذیری بصورتی کاملا مستقل خواهند بود.
- **پنجم اینکه تغییر پشته فناوری برنامه در معماری یکپارچه بشدت مشکل و تقریبا غیر ممکن است.** بعبارتی دیگر تقریبا هیچ راهی برای تغییر چارچوب برنامه وجود ندارد و انطباق برنامه با یک فناوری جدید بسیار سخت خواهد بود چرا که تمام مولفه های وابسته به برنامه با همان تکنولوژی که از ابتدا برنامه بر مبنای آن استارت خورده است عجین[xi i i] شده اند.



شکل(۲) : سمت چپ معماری یکپارچه را نشان می دهد که تمام عملکردش را درون یک فرآیند قرار می دهد بر خلاف معماری میکروسرویس که هر یک از عناصر یک عملکرد بخصوص درون سرویس های مجزی قرار داده می شوند. [24]



شکل(۳) : شماتیک مقیاس پذیری میکروسرویس ها در قیاس با یکپارچگی [۱۴]

٤ – پیاده سازی میکروسرویس با اتکا بر زبان برنامه نویسی

اما در ارتباط با بحث پیاده سازی معماری میکروسرویس ها توسط یک زبان خاص می توان به Jolie اشاره کرد ، زبانی که بصورت کاملا Native از این نوع معماری پشتیبانی می کند. در این زبان هر برنامه یک میکروسرویس است که تشریح و توصیف آن توسط یک رفتار خاص و یک سری اطلاعات استقراریافته ساخته و ارائه می شود که در تلاشند تا چگونگی برقراری ارتباطشان را با دیگر میکروسرویس ها ممکن سازند. در این مضمون ، مفهوم توزیع و اشاعه امری ذاتی و طبیعی در برنامه می باشد از آنجایی که هر برنامه قادر است ویژگی ها و عملکردش را در قالب یک URL[xiv] بخصوص داشته باشد که این آدرس خاص می تواند توسط دیگر میکروسرویس ها فراخوانی[xv] شود. مقیاس پذیری غیر یکنواخت[xvi] هم به خوبی قابل مشاهده است. به هر حال میکروسرویس های Jolie به آسانی قابل پیاده سازی و استقرار در Docker هستند. [۱۴, ۱۶] همچنین Jolie مکانیزم های پیشرفته ای را در ارتباط با آگاه سازی خطاهایی که فی ما بین میکروسرویس ها رخ می دهد برایمان به ارمغان می آورد.[۱۴, ۱۷]

٤-۱ نمایش سینتکس زبان برنامه نویسی جولی[۱۸]

سینتکس جولی را در قالب یک مثال ساده یک سرویس که "هر چیزی را دریافت کند چاپ میکند" نشان می دهیم. اول از همه باید یک اینترفیس تعریف کنیم که دیگر سرویس ها تمام عملکردهای کناری را لیست کرده و استفاده کنند.

```
interface PrintInterface {
      OneWay: print ( string )
}
```

شکل (٤) : قطعه کد اینترفیس [18]

این اینترفیس یک تابع یک طرفه PrintInterface تعریف می کند بدین معنی که هر سرویسی که از این اینترفیس استفاده می کند قادر است از این تابع استفاده کرده یا آنرا فراخوانی کند بدون دریافت یا تولید پاسخ. سپس در ادامه ما خود سرویس Print را با نام PrintService تعریف می کنیم که نام ورودی، مکان و پروتکل و اینترفیسی که استفاده می کند را بر می گرداند. رفتار تابع نیز در بخش Main مربوط به Service نشان داده شده است. رفتار سیستم هم نمایش دهنده تابع Print است که تمامی خطوطی که دریافت می کند را چاپ می کند.

```
include ''console.iol''

include ''printInterface.iol''

outputPort PrintService {
      Location: ''socket://localhost:8000''
      Protocol: json
      Interfaces: printInterface
}

main {
      print( line ){
         print@Console( line )()
}
```

شکل (٥): قطعه کد سمت سرور [18]

در پایان ما سرویس سمت کلاینت را تعریف می کنیم که شامل اطلاعات مورد نیاز به منظور فراخوانی سرویس چاپ و فراخوانی تابع چاپ هست. (Print@PrintService)

```
include ''printInterface.iol''

outputPort PrintService {
      Location: ''socket://localhost:8000''
      Protocol: json
      Interfaces: printInterface
}

main {
      print@PrintService(''Hello, world!'')
}
```

شکل (٦): قطعه کد سمت کلاینت [18]

٥ – علل مهاجرت به معماری میکروسرویس ها با بررسی نتایج تجربی ۲۱ متخصص حوزه صنعت نرم افزار

در این بررسی عوامل زیر از جمله عوامل اصلی مهاجرت به میکروسرویس ها برشمرده شده است :

- **قابلیت نگهداری بالا :** معماری ماژولار میکروسرویس ها باعث شده است که نرخ پیچیدگی سیستم های یکنواخت کاهش پیدا کند. شکستن یک سیستم به سرویس های مجزی و خودمختار[xvi i] باعث

شده است که توسعه دهندگان ماژول های موردنیازشان را بدون نیاز به دیگر توسعه دهندگان تست کرده و به اعمال تغییرات در آن بپردازند. همین عامل سبب سادگی در امر توسعه توزیع پذیری می شود.

- **مقیاس پذیری** : در مقایسه با سیستم های یکنواخت ، معماری میکروسرویس ها مقیاس پذیری بهتری دارد چرا که این امر در سیستم های یکنواخت نیازمند سرمایه گذاری عظیم در بحث سخت افزاری و کدزنی است. اگر یک Bottleneck در مولفه ای یافت شود یک قطعه سخت افزاری قدرتمند می تواند استفاده شود یا نمونه های مختلف از همان برنامه یکنواخت می تواند در سراسر سرویس های سیستم اجرا شده و توسط یک Load Balancer مدیریت شود.
- **مهاجرت سازمانهای عظیم به این سبک معماری:** میکروسرویس ها بسیار جذاب هستند و بسیاری از کمپانی ها بزرگ در حال تطبیق این نوع معماری بر روی سیستم های خود هستند. در نتیجه بسیاری از فعالان نرم افزاری سعی در کنکاش پتانسیل این معماری ارائه می کنند و رفته رفته این علاقه در میان متخصصان نرم افزار بیشتر رواج پیدا می کند.
- **تحمل خطا** [xvi i i] **:** خرابی یک میکروسرویس غالبا تمام سیستم را تحت تاثیر قرار نخواهد داد. در مقابل در معماری یکنواخت خرابی یک مولفه تمام برنامه را دچار مشکل می کرد. جالب اینجاست که در صورت خرابی میکروسرویس می توان نسخه قبلی آن را جایگزین نسخه فعلی کرد بدون آنکه نیاز به راه اندازی مجدد کل سیستم باشد و این معرکه است ، همچنین میکروسرویسی که دچار خرابی شده است می تواند به سرعت راه اندازی مجدد شود.
- **آزمایش آسان فناوری :** میکروسرویس ها طبق تعریف کوچک هستند و مولفه های کوچک ساده تر و سریع تر توسعه پیدا می کنند ، بنابراین برای آزمایش میکروسرویس با فناوری های جدید و اضافه کردن ویژگی های جدید به آنها کار ساده ای پیش رو خواهیم داشت ، چرا که میکروسرویس ها از طبیعت چند زبانی [xi x] بهره می برند ویژگی که معماری یکنواخت فاقد آنست چرا که اجازه توسعه با چندین زبان را نخواهد داشت.
- **مهاجرت سازمانهای عظیم به این سبک معماری:** میکروسرویس ها بسیار جذاب هستند و بسیاری از کمپانی ها بزرگ در حال تطبیق این نوع معماری بر روی سیستم های خود هستند. در نتیجه بسیاری از فعالان نرم افزاری سعی در کنکاش پتانسیل این معماری ارائه می کنند و رفته رفته این علاقه در میان متخصصان نرم افزار بیشتر رواج پیدا می کند.
- **مسئولیت های نرم افزاری مجزی :** میکروسرویس ها تنها مسئول یک وظیفه خواهند بود که در راستای مرزهای خوش تعریف [xx] می باشد و دارای ویژگی خودشمولی [xxi] می باشد، در نتیجه توسعه بسیار آسان خواهد بود.

۵-۱ چالش های پیش روی میکروسرویس ها

- **توسعه سیستم های توزیع شده دارای پیچیدگی است.** به دلیل اینکه هر چیز یک سرویس مستقل است و حجم رسیدگی به درخواست هایی که بین ماژول ها در جریان است امری است فزاینده و صعودی پس کنترل جریان انتقال، امری حیاتی تلقی می شود.
- **مدیریت پایگاه های داده متعدد و تراکنش های سنگین امری چالش برانگیز تلقی خواهد شد.**
- **فرآیند تست بسیار پرزحمت خواهد بود.** بر خلاف معماری یکنواخت که ما تنها نیاز به راه اندازی WAR بر روی سرور برنامه داشتیم و اطمینان حاصل کنیم اتصال با پایگاه داده مبنایی برقرار خواهد شد. اما در میکروسرویس ها ، هر سرویس مجزی پیش از انجام فرآیند تست می بایست تایید شده باشد
- **پیاده سازی میکروسرویس ها دارای پیچیدگی است.** نیاز به هماهنگی میکروسرویس ها در میان حجم کثیری از سرویس ها امری متحمل است، در حالی که شاید به مانند پیاده سازی WAR درون یک Container ساده و بی دردسر نخواهد بود.

اما راه حل چیست ؛ چطور می توانیم این معایب را برطرف کنیم ؟ البته که با یک نوع خودکارسازی و ابزارهای مطلوب می توانند معایب فوق را رفع کرد.

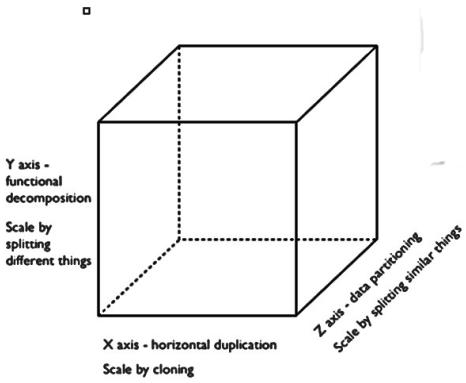
شاید بتوان گفت بزرگترین مزیت میکروسرویس ها این هست که درون پکیج های یکپارچه WAR-like مستقر نخواهند شد. پس استقرار میکروسرویس ها به چه صورت خواهد بود ؟

اما در هنگام مهاجرت از سیستم های یکنواخت به معماری میکروسرویس ها دچار چالش هایی می شویم که عمده ترین آنها طبق نظر متخصصان نرم افزار پیچیدگی جداسازی [xxi i] از معماری یکنواخت ، تقسیم داده ها در پایگاه های داده موروثی و ارتباطات بین سرویس ها می باشد. از سوی دیگر ذهنیت مردم هم نسبت به مهاجرت مزید بر علت است. همچنین میکروسرویس ها مبتنی بر سرویس گرایی می باشند نیاز مُبرم به لایه هم نواسازی [xxi i i] دارند که همین امر باعث پیچیدگی سیستم شده و نیاز به توسعه مطمئن را مطرح می کند.

۵-۲ علت اصلی مهاجرت به میکروسرویس ها

طبق نظر فعالان نرم افزاری مهمترین مزیت بهبود قابلیت نگهداری برشمرده است. اما چرا ؟ جواب واضح است ، وقتی پیچیدگی سیستم پایین باشد در نتیجه قابلیت نگهداری هم بهبود پیدا میکند. مورد بعدی افزایش عملکرد سیستم است چرا که طبیعت بیشتر سیستم های نوظهور مبتنی بر ابر [xxi v] می باشد و بدین جهت مقیاس پذیری هم افزایش پیدا میکند. همچنین سیستم های مبتنی بر میکروسرویس ها در اجرای بلند مدت به نسبت سیستم های یکنواخت هزینه کمتری خواهند داشت. اما در کوتاه مدت نتیجه برعکس خواهد بود!

یکی از موارد اصلی که پشت هر گونه راه حل های معمارگونه مطرح می شود قابلیت مقیاس پذیری [xxv] است. شماتیک ذیل ابعاد مقیاس پذیری را نشان می دهد. [xxvi]



شکل (۷) : ابعاد مقیاس پذیری [25]

۵-۳ استقرار معماری میکروسرویس ها

بهترین راه استقرار برنامه های مبتنی بر میکروسرویس ، Container ها می باشند. Container ها راه حل هایی هستند به منظور کسب اطمینان اینکه نرم افزار هنگام جابجایی از یک محیط اولیه به محیط ثانویه بصورت مطمئن [xxvi i] اجرا شود و بر بستر Docker قرار می گیرند. Container به توسعه دهنده این اجازه را می دهد تا تمام بخش های موردنیاز مانند کتابخانه ها و دیگر وابستگی ها را بسته بندی کرده و به عنوان یک پکیج مستقل روی سیستم های دیگر استفاده کند.

۵-۴ آشنایی با بستر Docker

Docker خود را چنین معرفی می کند:

"Build any app, Ship anywhere, run anywhere"

"هر برنامه ای را که تمایل داشتید بسازید ، هر جایی که خواستید ارسال کنید و هر جایی که خواستید آنرا اجرا کنید."

داکر در واقع یک سکوی متن باز است که فرآیند استقرار نرم افزار را با معرفی مفهوم Container سرعت می بخشد. این ابزار کمکی است به بزرگ به توسعه دهندگان و مدیران سیستم هاست که در ساخت و ارائه و اجرای برنامه های توزیع شده با استفاده از Container ها سعی در پیاده سازی دارند. در واقع Container ها ظرفی برای پیاده سازی Image هایی هستند که قصد اجرای آنها را در سطح سیستم عامل داریم.

از جمله ویژگی های Docker می توان به موارد ذیل اشاره کرد :

۱) چابکی ایجاد و حذف سریع محفظه ها
۲) انتقال پذیری سبک و آسان [xxviii]
۳) صرفه جویی در هزینه [xxi x]
۴) مدیریت قدرتمند مصرف منابع
۵) امنیت

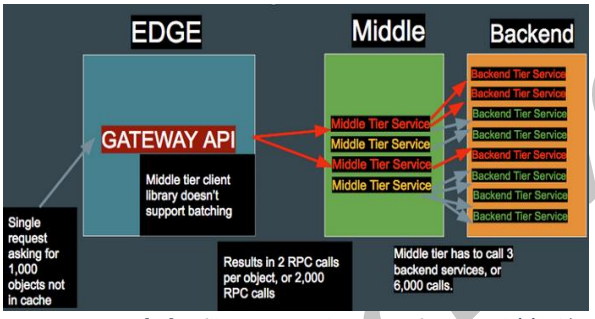
در ادامه اجزای اصلی Docker را با هم مرور می کنیم که ۳ دسته هستند :

۱) Docker Daemon: مدیر و مجری محفظه ها
۲) Docker CLI: رابط دستوری مرتبط با Daemon
۳) Docker Image: Image های برنامه مربوطه

تلاش هایی [۱۹]در جهت اتصال زبان Jolie به داکر در راستای هم نواسازی میکروسرویس ها[xxx] صورت گرفته است که راهکارهای معمارگونه ای هم ارائه شده است.

۶ – کشف یک رخنه امنیتی در واسط های برنامه نویسی موجود در سازمان های مبتنی بر معماری میکروسرویس ها و ارائه دو ابزار تحلیل نقاط ضعف توسط Netflix

بنا بر گزارش سه ماه اول اینترنت در **۲۰۱۷** حملات لایه کاربرد کمتر از ۱٪ حملات DDOS[xxxi] را شامل می شوند. طبق گفته مهندسان Netflix این نوع از حملات برای سازمان های مبتنی بر معماری میکروسرویس ها دردسرساز می باشند. مشکل موجود در این سبک این است که یک واسط برنامه نویسی [xxxi i] در دروازه سازمان [xxxi i i] وجود دارد که درخواست های متعددی را به سوی سرویس های پشتی و میانی می فرستد، اگر شخص مهاجم مهارت شناسایی واسط را داشته باشد کار تمام است، چرا که می تواند کنترل سرویس های میانی را در دست بگیرد و مقدمات حمله را فراهم کند، در این صورت کل سیستم از کار می افتد، علت هم اینست که یک ارسال درخواست به سرویس های میانی خود می تواند چندین هزار درخواست و فراخوانی پشتی و میانی را ایجاد کند. [۲۰]



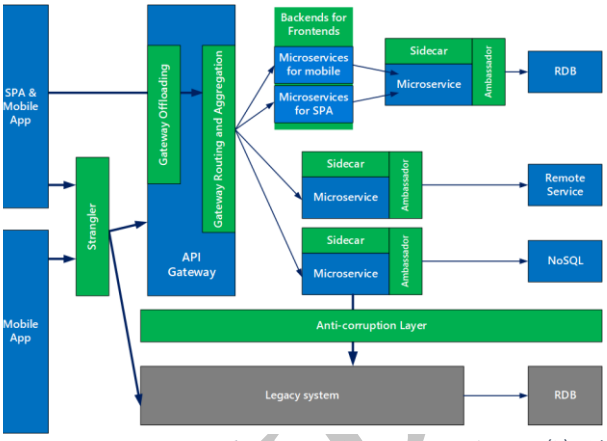
شکل (8): لایه های سه گانه معماری میکروسرویس ها با الگوی Getway API [20]

۶-۱ الگوهای جدید طراحی میکروسرویس ها [۲۱]

۹ الگو به طور خاص به منظور طراحی میکروسرویس ها سودمند هستند که توسط تیم AzureCAT در پایگاه معماری Azure منتشر شده اند. علت اصلی مستندسازی این الگوها، افزایش میل و رغبت به سمت این سبک معماری در صنعت بوده است. هدف اصلی معماری میکروسرویس ها افزایش سرعت انتشار نرم افزار عنوان شده است و این سرعت رو با تجزیه سازی یک برنامه به سرویس های کوچک خودمختار انجام می دهد که می توانند به تنهایی و بدون هیچگونه وابستگی استقرار یابند. یک معماری میکروسرویس همانند بسیاری نوآوری های دیگر **چالش هایی** را با خود به همراه دارد که الگوهای ۹ گانه فوق سبب کاهش این چالش ها شوند.

۱) **Anti-Corruption Layer:** یک جبهه بین برنامه جدید و برنامه موروثی ایجاد می کند تا مطمئن شود طراحی یک برنامه جدید محدود به وابستگی هایش به سیستم های موروثی نیست.

۲) **Backends for frontends**: سرویس های سمت سرور مجزی برای انواع مختلف کلایت می سازد مانند دسکتاپ و موبایل. در این صورت یک سرویس سمت سرور منفرد نیاز ندارد تا تمامی نیازمندی های پیچیده انواع مختلف کلاینت را پوشش دهد. این الگو همچنین با جداسازی دغدغه های اختصاصی کلاینت[xxxi v] باعث می شود هر میکروسرویس را به طور ساده و به دور از پیچیدگی نگهداری



شکل(۹) : نمودار فوق نشان می دهد که چطور این الگوها در معماری میکروسرویس ها بکارگرفته می شوند. [22]

کرد.

۳) **Bulkhead**: منابع حیاتی سیستم مانند مخزن اتصالات، حافظه، واحد پردازش مرکزی را برای هر سرویس یا بارکاری به صورت جدا از هم ایزوله می کند. با استفاده از این الگو، یک بارِ کاری تمامی منابع سیستم را به تنهایی مصرف نمی کند تا دیگران دچار قطعی و گرسنگی در دسترسی به منابع شوند و با پیشگیری از خطاهای دنباله داری که توسط یک سرویس رخ می دهد، باعث انعطاف پذیری سیستم می شود.

۴) **Gateway Aggregation**: درخواست های ارسالی به سوی میکروسرویس های چندگانه مستقل را درون یک درخواست واحد تجمیع می کند که همین امر باعث کاهش تبادل بین مصرف کننده و سرویس های مربوطه می شود.

۵) **Gateway Offloading**: میکروسرویس ها را قادر می سازد تا عملکرد اشتراکی سرویس را Offload کنند، مانند استفاده از گواهینامه امنیتی SSL به جهت ارائه به دروازه رابط برنامه نویسی API Gateway

۶) **Gateway Routing**: درخواست مسیرها به میکروسریس های مختلف از طریق یک نقطه انتهایی منفرد انجام می شود. به همین جهت مصرف کننده نیاز ندارد تا تعداد زیادی از نقاط انتهایی پایانی را مدیریت کند.

۷) **Ambassador**: به منظور Offload وظایفی مانند نظارت، مسیریابی و امنیت مانند TLS (امنیت لایه انتقال) استفاده می شود.

۸) **Sidecar**: استقرار مولفه های کمکی یک برنامه را به عنوان کانتینرها یا پردازش های جداگانه به منظور ایزوله سازی و کپسوله کردن را بر عهده دارد.

۹) **Strangler**: از مهاجرت افزایشی بوسیله جداسازی قسمت های خاص یک خدمت به همراه سرویس های جدید پشتیبانی می کند. به تازگی الگوی جدیدی هم تحت عنوان Database-Is-the-Service Pattern ارائه شده است که در آن سیستم مدیریت پایگاه داده غیر رابطه ای درون یک خوشه به همراه منطق کاری ارائه شده است.[۲۲]

۶-۲ جایگزین Endpoints و Dumb Pipes هایی برای گذرگاه سرویس سازمانی Enterprise Service Bus (ESB)[xxxv]

محصولات گذرگاه سرویس سازمانی معمولا شامل امکانات پیچیده به منظور مسیریابی پیام، کاریگرافی، دگرگونی و اعمال قوانین کسب و کار می باشند. جامعه میکروسرویس از دو رویکرد جایگزین بهره می برد : Endpoints و Dumb Pipes. برنامه هایی که بر مبنای معماری میکروسرویس ها طراحی شده اند می بایست حدالامکان مجزی و منسجم باشند. آنها منطق دامنه خاص خودشان را دارند به طوری که یک درخواست را دریافت می کنند، منطق مناسب را اعمال می کنند و یک پاسخ مقتضی را تولید می کنند. آنها از طریق پروتکل های ساده RESTish بجای پروتکل های WS-Choreography یا BPEL یا هم نواسازی بوسیله یک ابزار مرکزی راهنمایی و هدایت می شوند.[۲۳]

۷ – نتیجه گیری[xxxvi]

معماری میکروسرویس ها یک رویکرد بسیار عالی در معماری سیستم های نرم افزاری است اما نمی توان گفت به معنی یک راه حل جادویی بر حل تمامی مشکلات در حوزه نرم افزار می باشند. تا به امروز تصورات غلطی در رابطه با میکروسرویس ها وجود داشته است که ناشی از عدم فهم صحیح رسالت این سبک معماری است. از جمله اینکه با ریفکتورینگ یک مانولیت به چند میکروسرویس دیگر مانولیتی در کار نیست.

بالعکس با اینکار ما بجای داشتن یک مانولیت چندین مانولیت (به تعداد سرویس ها) ولی در ابعاد کوچکتر و دغدغه های جدیدتری خواهیم داشت. برداشت اشتباه دیگر کاهش پیچیدگی سیستم می باشد، بالعکس پیچیدگی در سیستم موجود خواهد بود اما در جاهای دیگر خودش را نشان می دهد که در اینصورت دغدغه های مرتبط با پیچیدگی های جدیدتری نیز معرفی خواهند شد که از جمله آنها می توان به Inter-Process Communication و Distributed Data اشاره کرد. [۲۴]

اما باید گفت با تجربه ها / درسهایی که معماری میکروسرویس ها از معماری سرویس گرا گرفته است می تواند در آینده به موفقیت برسد که می توان از آن تجارب / درسها به موارد کلیدی ذیل اشاره کرد :

۱) **بر خطی مشی اصلی میکروسرویس ها باید پایبند بود** : معماری سرویس گرا زمانی که از رسالت اصلی خود که چابکی و کاهش هزینه ها بود فاصله گرفت دچار نیاز شدید به فناوری شد و همین مساله میل و رغبت به معماری سرویس گرا را کاهش داد. در معماری میکروسرویس ها هم هرچند کسب و کار اصل قرار دارد و سرویس ها بر اساس کسب و کارمان طراحی و پیاده سازی می شوند اما در این نقطه هم فناوری هایی هستند که ادعا دارند با کانتینر کردن برنامه های خود به معماری میکروسرویس دست یافته اند.

۲) **با نمونه های موفق باید شروع کرد :** هنگام معرفی معماری سرویس گرا پیاده سازی های کمی از این سبک وجود داشت و بیشتر بعنوان یک الگوی تفکر مطرح شده بود اما میکروسرویس ها از پیاده سازی سازمانهای موفق نشات گرفته است. بعنوان مثال Amazon , Netflix دو نمونه موفق پیاده سازی معماری میکروسرویس ها هستند و یا مهاجرت اپلیکیشن بازار و دیوار.

۳) **چشم انداز بسیار مهم است** : تغییر تعریف و ماهیت Enterprise Service Bus ها در معماری سرویس گرا عاملی بود بر تغییر فرهنگ سازمانی منطبق با سرویس گرایی که لازمه تفکر معماری سرویس گرا بود. پس معماری میکرسرویس ها باید علاوه بر پایبندی به چشم اندازش در نگهداریش کوشش کند.

٤) به دنبال هماهنگی در اهداف باید باشیم و نه قطعیت تام : معماری سرویس گرا در راستای نیل به دو هدف ارائه شد. اول افزایش زمان پاسخ به نیازهای بازار و دوم کاهش هزینه های یکپارچه سازی سیستم، اما یک مشکل وجود داشت، آن هم اینکه میل شدیدی به مرکزیت محوری در معماری سرویس گرا رشد کرد، همین عامل باعث شد تمرکز سازمانها بر کنترل و استاندارد سازی یکپارچگی برنامه هایشان نیل پیدا کند. اما مشکل اصلی در عدم هماهنگی و تعادل دو هدف فوق بود. اما معماری سرویس گرا با اهداف افزایش سرعت تحویل نرم افزار و افزایش صحت سیستم در هنگام مقیاس وسیع شدنش ارائه شد. این دو هدف دارای هماهنگی و تعادل هستند.

٥) اصل کاری ها قواعد و الگوها هستند نه فناوری : امکان دارد فناوری Docker که در بالا معرفی کردیم به زودی قدیمی شود و جای خود را به یک فناوری جدیدتر بدهد، همانطور که یک روز Hypervisor ها در Virtual Machineها یکه تازی میکردند و بی رقیب پیش می رفتند، البته درصد فعلی Hypervisor ها به نسبت Docker مقدار ۹۰٪ به ۱۰٪ می باشد که پیش بینی میشود در ۵ سال آینده به ۵۰٪ برسد. پس نتیجه میگیریم نمی توان به فناوری وابسته بود و الگوها مقدم هستند.

سپاسگزاری

در پایان از استاد عزیز و گرانقدرم دکتر فریدون شمس علیئی به جهت راهنمایی های ارزنده ایشان تشکر و قدردانی می کنم.

مراجع

[۱] Balalaie, A., A. Heydarnoori, and P. Jamshidi, *Microservices architecture enables DevOps: migration to a cloud-native architecture.* IEEE Software, 2016. 33(3): p. 42-52.

[۲] Franchitti, J.-C., *Application Servers Presentation in Book Scheme*, in *Application Servers*. New York University.

[۳] Roy Fielding, PhD Thesis, University of California 200, at https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[۴] Pahl, C. and P. Jamshidi. *Microservices: A Systematic Mapping Study*. in *CLOSER (1)*. 2016.

[۵] Taibi, D., V. Lenarduzzi, and C. Pahl, *Processes, Motivations and Issues for Migrating to Microservices Architectures: An Empirical Investigation.* IEEE Cloud Computing, 2017.

[۶] Dragoni, N., et al., *Microservices: yesterday, today, and tomorrow*, in *Present and Ulterior Software Engineering*. 2017, Springer. p. 195-216.

[۷] Thönes, J., *Microservices.* IEEE Software, 2015. 32(1):. 116-116.

[۸] Namiot, D. and M. Sneps-Sneppe, *On micro-services architecture.* International Journal of Open Information Technologies, 2014. 2(9) . 24-27.

[۹] Mauro, T., *Adopting Microservices at Netflix: Lessons for Architectural Design*, in *nginx*. 2015 at https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

[۱۰] Montesi, F., *Jolie: a service-oriented programming language*. 2010 at http://www.fabriziomontesi.com/files/m10.pdf

[۱۱] Mazzara, M., et al., *Towards Microservices and Beyond.* 2017 at https://arxiv.org/abs/1610.01778

[۱۲] Evans, E., *Domain-driven design: tackling complexity in the heart of software*. 2004: Addison-Wesley Professional.

[۱۳] Uckelmann, D., M. Harrison, and F. Michahelles, *An architectural approach towards the future internet of things*, in *Architecting the internet of things*. 2011, Springer. p. 1-24.

[۱۴] Dragoni, N., et al., *Microservices: How to make your application scale.* arXiv preprint arXiv:1702.07149, 2017.

[۱۵] in *Enterprise Architect Portal* Webpage. 2017.

[۱۶] Merkel, D., *Docker: lightweight linux containers for consistent development and deployment.* Linux Journal, 2014. 2014(239): 2.

[۱۷] Guidi, C., et al., *Dynamic error handling in service oriented applications.* Fundamenta Informaticae, 2009. 95(1): p. 73-102.

[۱۸] Mazzara, M., et al., *Microservices Science and Engineering.* SEDA 2016, 2018: p. 11.

[۱۹] Giaretta, A., N. Dragoni, and M. Mazzara, *Joining jolie to docker-orchestration of microservices on a containers-as-a-service layer.* arXiv preprint arXiv:1709.05635, 2017.

[۲۰] Kovacs, E., *Netflix Helps Identify APIs at Risk of Application DDoS Attacks*, in *SecurityWeek* at https://www.securityweek.com/netflix-

ITBAM 2016, Porto, Portugal, September 5-8, 2016 .Springer.

[۲۴] *Microservices*, in http://martinfowler.com/articles/microservices.html. 2014.

[۲۵] Abbott, Martin L., and Michael T. Fisher. *The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise*. Pearson Education, 2009.

helps-identify-apis-risk-application-ddos-attacks 2017.

[۲۱] .

[۲۲] Wasson, M., *Design patterns for microservices*, in *Microsoft*, https://azure.microsoft.com/en-us/blog/design-patterns-for-microservices/ 2017.

[۲۳] Urso, A. *The Database-is-the-Service Pattern for Microservice Architectures*. in *Information Technology in Bio-and Medical Informatics: 7th International Conference,*

Interface [i]
Building-blocks [ii]
Monolith [iii]
Cross Functional Requirements (XFR) [iv]
Tightly Coupled [v]
Bounded-Context [vi]
Granularity [vii]
Loosely Coupling [viii]
Unit Solution [ix]
Scalability [x]
Transaction Volumes [xi]
Memory Consumption [xii]
Sticking [xiii]
Uniform Resource Locator [xiv]
Invoke [xv]
Non-Uniform Scalability [xvi]
Self-deployable [xvii]
Fault Tolerance [xviii]
Polyglot Nature [xix]
Well-Defined [xx]
Self-Contained [xxi]
Decouple Complexity [xxii]
Orchestration [xxiii]
Cloud Base [xxiv]
Scalability [xxv]
شکل برگرفته از کتاب **The Art of Scalability** می باشد. [xxvi]
Reliable [xxvii]
Portability [xxviii]
Cost Saving [xxix]
Microservices Orchestration [xxx]
Distributed Denial of Service [xxxi]
API [xxxii]
Enterprise Gate [xxxiii]
Separating client-specific concerns. [xxxiv]
Refactor.ir [xxxv]
Conclusion at Refactor.ir [xxxvi]