معماری مایکروسرویس

(Microservice Architecture)

امیر مهجوریان – ۱۳۹۸

۱) تعاریف و مفاهیم معماری مایکروسرویس

۱-۱) تاریخچه و تعاریف

معماری مایکروسرویس (Microservice Architecture) یا به‌اختصار مایکروسرویس‌ها که در منابع فارسی گاهی به نام میکروسرویس‌ها نیز ترجمه شده است، در چندسال اخیر به‌شدت در حال گسترش و فراگیری در میان توسعه‌دهندگان و معماران است. اگرچه اولین اشاره مستقیم به واژه "مایکروسرویس‌ها" به سال ۲۰۱۱ و در یک کارگاه معماری نرم‌افزار برمی‌گردد، اما داغ شدن این موضوع در طی سال‌های ۲۰۱۴ و ۲۰۱۵ بود؛ هم‌اکنون مایکروسرویس‌ها یکی از موضوعات جذاب در دنیای نرم‌افزار و معماری محسوب می‌شود و هرماه مقالات، کتاب‌ها و ارایه‌های جدیدی از آن منتشر می‌شود و در کنفرانس‌ها یا سمینارهای تجاری-علمی نیز علاقمندان زیادی را به خود جذب می‌کند؛ حتی بر اساس گزارش‌های گوگل از میزان رشد جستجوی عبارات مرتبط با مایکروسرویس‌ها می‌توان به نقش محوری آن در معماری و توسعه سیستم‌ها پی برد.

در دنیای تجاری نیز شرکت‌های مطرحی پیشگام پیاده‌سازی و استقرار آن بوده‌اند که از جمله می‌توان به شرکت‌های Uber ، Netflix ، Amazon ، Ebay و Sound Cloud اشاره نمود. بر اساس تحقیقاتی متفاوتی که توسط Forrester[۱] ، Redhat[۲] و Dimensional Research[۳] انجام شده است، بیش از ۷۰ درصد پرسش‌شوندگان اعلام کرده‌اند که برنامه‌ای برای توسعه و پیاده‌سازی معماری مایکروسرویس دارند.

در این مطلب در خصوص اصول و مفاهیم معماری مایکروسرویس، ابزارها و تکنیک‌های آن و چالش‌های پیش‌روی پیاده‌سازی در مقیاس سازمانی، مطالبی ارایه خواهد شد. در ابتدا تعاریف مختلفی که توسط مراجع معتبر ارایه شده، مرور می‌گردد:

---

۱ Forrester Data Global Business Technographics® Developer Survey, 2017
۲ Redhat 2017 Microservices Survey
۳ Global Microservices Trends: A Survey of Development Professionals

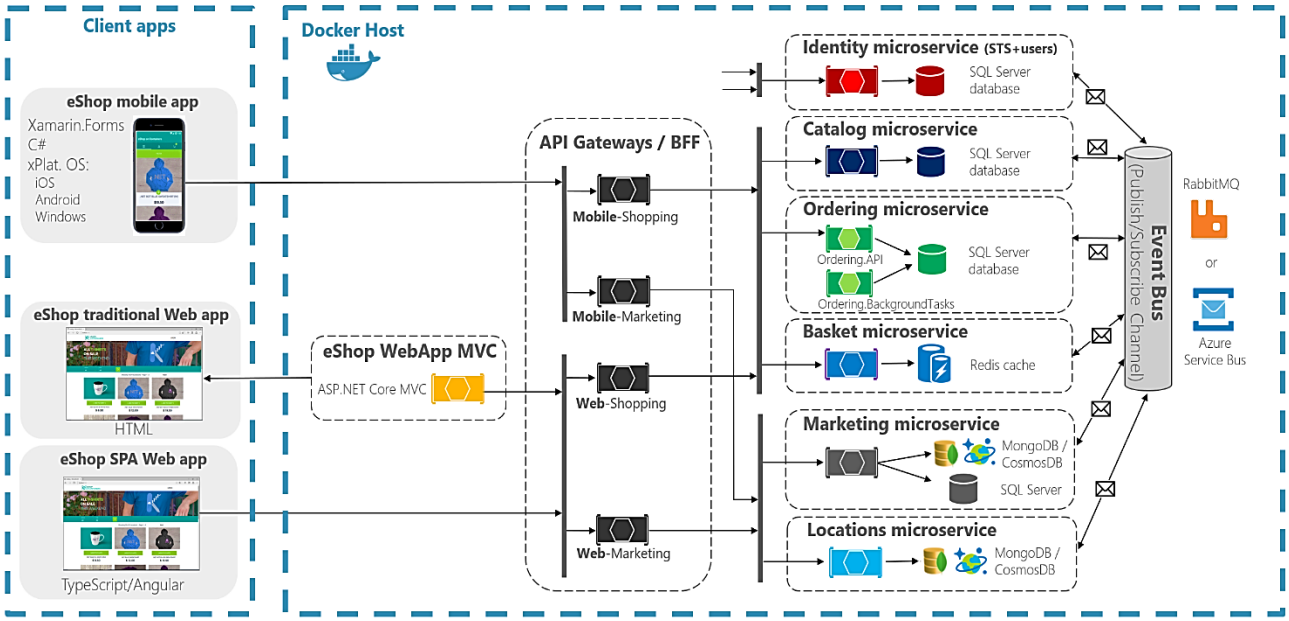
این مقاله در سال ۱۳۹۸ منتشر شده است.

- مایکروسرویس‌ها یک تکنیک توسعه نرم‌افزار مشتق شده از سبک معماری سرویس‌گرا است که از مجموعه‌ای از سرویس‌های خوش‌تعریف تشکیل شده است. در معماری مایکروسرویس پروتکل‌های ارتباطی سبک و مستقل از پلتفرم هستند و سرویس‌ها دامنه و مسئولیت معیین و مشخصی دارند، مزایای این معماری بهبود ماژولاریتی سیستم و تسهیل توسعه، استقرار و تست سیستم است؛ همچنین سیستم توسعه‌یافته دارای مقیاس‌پذیری بالا و سرعت بالاتر اعمال تغییر است. این معماری با رویکرد DevOps در توسعه و پشتیبانی نرم‌افزارها هماهنگی دارد. (Wikipedia)
- سبک معماری مایکروسرویس رویکردی برای توسعه یک نرم‌افزار متشکل از تعدادی سرویس کوچک و مستقل است که هر سرویس به‌اتکاء منابع و زیرساخت خودش اجرا شده و از طریق پروتکل‌های سبک مبتنی بر HTTP با دیگران ارتباط دارد. این سرویس‌ها براساس قابلیت-های کسب‌وکار طراحی و ساخته می‌شوند و بر بسترهای فناوری با زبان‌های برنامه‌نویسی مختلفی قابل استقرار هستند. این سرویس‌ها حداقل نیاز به مدیریت متمرکز را دارند و هر سرویس پایگاه داده مخصوص به خود را مدیریت می‌کند. (Martin Fowler)
- مایکروسرویس‌ها به‌صورت خلاصه سرویس‌های دانه‌ریز و خودمختاری هستند که با یکدیگر همکاری می‌کنند. هر سرویس باید بتواند مستقلا تغییر کند بدون اینکه منجر به تغییر دیگر سرویس‌های مرتبط یا استفاده‌کنندگان از سرویس شود. (Sam Newman)
- معماری مایکروسرویس یک رویکرد مهندسی مبتنی بر شکست یک نرم‌افزار به ماژول‌های تک-کارکردی است که مستقلا تولید و مستقر می‌شوند و با واسط‌های خوش‌تعریف با دیگر سرویس‌ها ارتباط دارند. این سرویس‌ها توسط تیم‌های کوچکی تولید و پشتیبانی می‌شوند که از تمام چرخه حیات سرویس پشتیبانی می‌کند (IBM)
- معماری مایکروسرویس از مجموعه‌ای از سرویس‌های خودمختار و کوچک تشکیل شده است که هر سرویس مستقل بوده و یک قابلیت کسب‌وکار را پیاده‌سازی می‌نماید (Microsoft)
- معماری مایکروسرویس یک رویکرد چابک و ماژولار به توسعه نرم‌افزار است که برخلاف نرم‌افزارهای یک‌تکه - که همه مولفه‌ها و قابلیت‌های سیستم بایکدیگر آمیخته شده‌اند- مبتنی بر مجموعه‌ای از سرویس‌های کوچک‌تر و مستقل از هم با ارتباط سست است. هر سرویس مسوول انجام وظایف و پردازش خود است و یک کارکرد مشخص از کل سیستم را پشتیبانی می‌کند و با دیگر سرویس‌ها از طریق API ارتباط دارد (Oracle)

این مقاله در سال ۱۳۹۸ منتشر شده است.

در جمع‌بندی تعاریف شده می‌توان گفت: "معماری مایکروسرویس، سبک خاصی از معماری نرم‌افزار و مشتق-شده از معماری سرویس‌گرا است که هدف آن خودمختاری بالای سرویس‌ها از نظر منطق کارکردی-داده‌ای و نیز پلتفرم پیاده‌سازی و اجرا است. این سبک معماری علاوه بر معماری سرویس‌گرا از مفاهیم معماری رخداد محور و سیستم‌های توزیع‌شده نیز بهره‌برده است."

نمونه‌ای از عناصر یک سیستم ساده فروش اینترنتی مبتنی بر معماری مایکروسرویس در شکل زیر نشان داده شده است.



شکل ۱-۱: نمونه سیستم ساده فروش اینترنتی مبتنی بر معماری مایکروسرویس

۱-۲) اصول و ویژگی‌های معماری مایکروسرویس

مهمترین اصول و ویژگی‌های معماری مایکروسرویس به قرار زیر است:

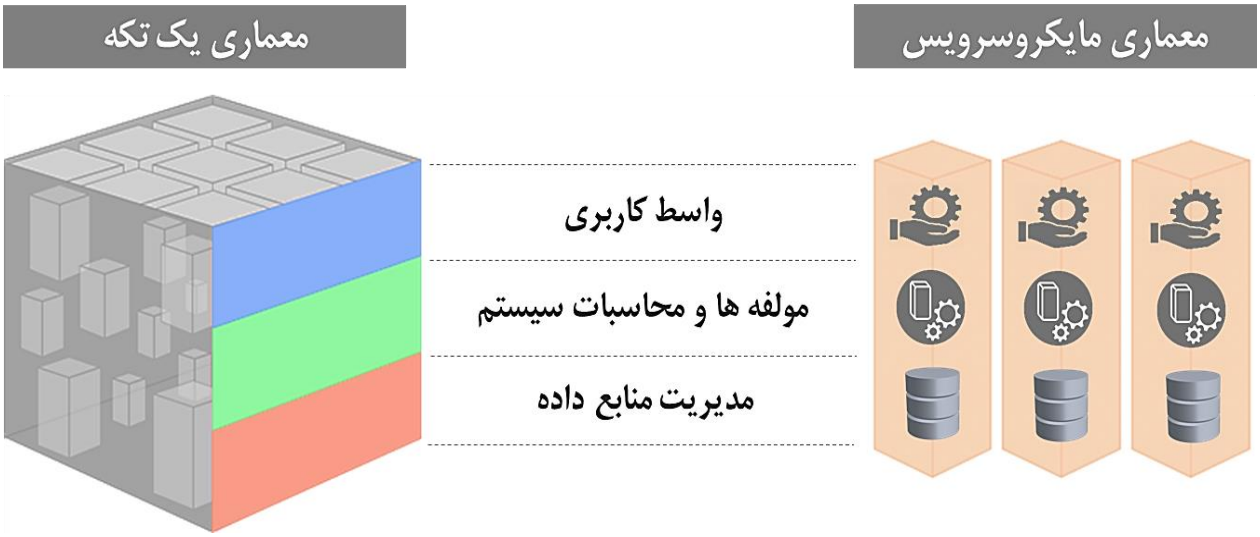
- هر سرویس مسوول یک دامنه مشخص و به‌خوبی تعریف‌شده از سیستم(صورت‌مساله) است که مستقلا تولید(Build) و استقرار(Deploy) می‌یابد.
- هر سرویس از فناوری‌ها و ابزارهای مناسب خود بهره می‌برد و لزومی ندارد همه سرویس‌های یک سیستم از یک فناوری، زبان‌برنامه‌نویسی یا پلتفرم استفاده کنند.
- سرویس‌ها با واسط‌های خوش تعریف و سبک با یکدیگر تعامل دارند، خروجی هر سرویس باید بتواند ورودی سرویس‌های دیگری قرار گیرد.
- هر سرویس مسوول مدیریت داده‌های خود است و می‌تواند از انواع ابزارهای DBMS استفاده نماید.

این مقاله در سال ۱۳۹۸ منتشر شده است.

- اصول کلی معماری سرویس‌گرا در این معماری نیز صادق است.
- ترجیح استفاده از روش پیام‌رسانی غیرهمزمان(Asynchronous) نسبت به همزمان
- ترجیح استفاده از روش همکاری کاریگرافی(Choreography) نسبت به ارکستریشن

۱-۳) معماری مایکروسرویس دربرابر معماری یک‌تکه(Monolithic)

بسیاری از کتاب‌ها و مقالاتی که درباره معماری مایکروسرویس وجود دارد، برای توضیح ویژگی و تمایز معماری مایکروسرویس از مقایسه با معماری یک‌تکه استفاده کرده‌اند. بر مبنای نظر این نویسندگان در سیستم‌های یک‌تکه، مجموعه مولفه‌ها-سرویس‌ها-داده‌ها چنان در هم آمیخته است که نمی‌توان بلوک‌های سازنده این سیستم‌ها را مستقلا از هم جدا کرده و یا تغییر(جایجا) نمود؛ اما در معماری مایکروسرویس هدف این است که یک سیستم به مجموعه‌ای از ماژول(سرویس) کاملا مستقل(خودمختار) تقسیم شود که هر سرویس همه محاسبات، داده‌ها و قوانین مورد نیاز را در خود داشته باشد و برای اجرا به سایر سرویس‌ها نیاز نداشته باشد یا حداقل وابستگی وجود داشته باشد.(شکل زیر)



شکل ۱-۲: مقایسه مفهومی معماری یک‌تکه در برابر مایکروسرویس

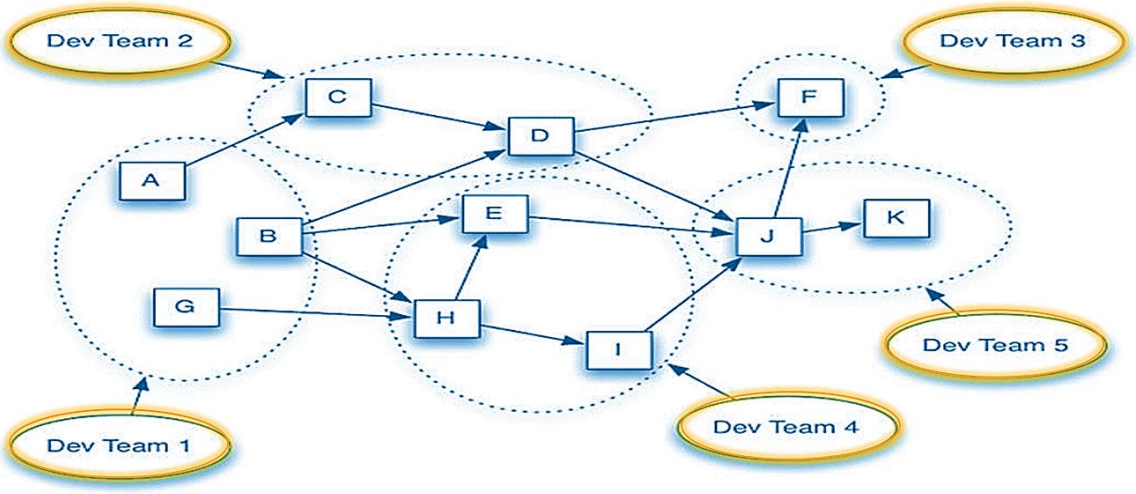
۱-۴) طراحی و استخراج مایکروسرویس‌ها

مهمترین موضوع در معماری مایکروسرویس، نحوه استخراج و دانه‌بندی سرویس‌ها است بگونه‌ای که این سرویس‌ها کاملا خودمختار بوده و حداقل نیاز به مدیریت مرکزی(متمرکز) در سیستم وجود داشته باشد، برای استخراج سرویس فضای مساله به مجموعه‌ای از دامنه‌ها شکسته شده و هر تیم مسوول طراحی-پیاده‌سازی-پشتیبانی یکی از دامنه‌ها خواهد بود که با مفاهیم DevOps نیز همخوانی دارد؛ برای استخراج سرویس و

این مقاله در سال ۱۳۹۸ منتشر شده است.

انتخاب دامنه‌ها در معماری مایکروسرویس معمولا از چندین روش و تکنیک استفاده می‌شود که معروف‌ترین آن‌ها طراحی مبتنی‌بر دامنه (Domain Driven Design) است، کتاب‌های مختلفی در این حوزه وجود دارد که معروف‌ترین آن‌ها نوشته Eric Evans[۴] می‌باشد. علاوه بر متد گفته شده مجموعه‌ای از تکنیک‌های شناسایی سرویس مبتنی بر معماری سرویس‌گرا و شی‌گرایی نیز مورد استفاده قرار می‌گیرد که در بخش‌های بعدی معرفی خواهند شد.

صرف‌نظر از تکنیک و روش شناسایی سرویس در معماری مایکروسرویس، این اصل پذیرفته شده است که برای چابکی در توسعه و تغییرات سرویس، برای هر مجموعه از سرویس‌های مرتبط یک تیم تخصصی با رویکرد DevOps اختصاص یابد(شکل زیر).



شکل ۱-۳: نحوه تقسیم مسوولیت تیم‌های طراحی-پیاده‌سازی سرویس

[۴] Domain-Driven Design: Tackling Complexity in the Heart of Software

۱-۵) ارتباط معماری سرویس‌گرا با معماری مایکروسرویس

یکی از موضوعات مورد بحث دراین حوزه، نسبت معماری مایکروسرویس با معماری سرویس‌گرا است؛ دلیل این امر شباهت‌ها و اشتراکات فناوری-متدها دراین دو معماری است. در این خصوص دو دیدگاه کلی وجود دارد؛ دیدگاه اول معماری مایکروسرویس را رویکردی در تقابل با معماری سرویس‌گرا می‌داند و به نوعی معماری سرویس‌گرا را نوعی معماری یک‌تکه قلمداد می‌کند که معماری مایکروسرویس برای تغییر آن آمده است. دیدگاه دوم که مورد تایید مراجع معتبرتر است معتقد است معماری سرویس گرا یک پارادایم فکری است و فناوری‌هایی مانند وب-سرویس(نسل اول تحقق معماری سرویس‌گرا) یا مایکروسرویس‌ها نمونه‌هایی از تحقق آن هستند که هرکدام نقاط قوت و ضعفی دارند.

برای بررسی موضوع نگاهی می‌کنیم به تعریف معماری سرویس‌گرا توسط IBM که در سال ۲۰۰۲ (هفده سال قبل) ارایه شده بود و نویسنده نیز در پایان‌نامه کارشناسی ارشد[۵] و کتابی[۶] که بعدا در سال ۱۳۸۹ منتشر نمود، به آن ارجاع داد:

> *"معماری سرویس‌گرا رهیافتی برای ساخت سیستم های توزیع شده است که کارکردهای نرم افزاری را در قالب سرویس ارائه می کند. این سرویس ها هم توسط دیگر نرم افزارها قابل فراخوانی هستند و هم برای ساخت سرویس های جدید مورد استفاده قرار می گیرند، این رهیافت برای یکپارچه سازی فناوری ها در محیطی که انواع مختلفی از سکوهای نرم افزاری و سخت افزاری وجود دارد ایده آل است."*

با مقایسه تعریف فوق از معماری سرویس‌گرا با اصول و تعاریفی که درباره معماری مایکروسرویس ارایه شد، به روشنی مشخص می‌شود که تقابل(تفاوت) عمده‌ای بین این دو نیست، و شاید بتوان معماری مایکروسرویس را گونه کامل‌تر و اصولی‌تر از تحقق معماری سرویس‌گرا نسبت به فناوری وب-سرویس و استانداردهای آن دانست که طی دو دهه گذشته معرفی شدند.

---

[۵] پایان نامه کارشناسی ارشد با عنوان " تدوین متدولوژی معماری سازمانی سرویس گرا در جهت پوشش به چارچوب زکمن"

[۶] کتاب " اصول، مبانی و روش های معماری سازمانی سرویس گرا " انتشارات دانشگاه شهید بهشتی: نویسندگان دکتر فریدون شمس و امیررضا مهجوریان

۱-۶) نتایج و مزایای معماری مایکروسرویس

مهمترین مزایا و نتایج حاصل از معماری مایکروسرویس به قرار زیر است:

حق‌انتخاب فناوری/ابزار:

- در معماری مایکروسرویس حق انتخاب سبد متنوعی از فناوری‌ها-ابزارها برای تیم‌های طراحی و پیاده‌سازی مهیا است به‌صورتی‌که می‌توان در فرایند تولید یک سیستم، برای مایکروسرویس‌های مختلف از فناوری‌ها و ابزارهای مختلفی استفاده کرد بدون اینکه نگرانی از مشکلات بعدی یکپارچگی وجود داشته باشد.

پایداری سیستم:

- یکی از ویژگی‌های اصلی سیستم‌های پایدار امکان ادامه فعالیت سایر سرویس‌ها(مایکروسرویس‌ها) در صورت ازکار افتادن یک سرویس است(و در صورت نیاز جایگزینی سرویس ازکار افتاده با نمونه مشابه یا پشتیبان)، این موضوع در معماری مایکروسرویس به دلیل خودمختاری و عدم وابستگی سرویس‌ها محقق می‌شود.

مقیاس‌پذیری بالا و هدفمند:

- امکان مقیاس‌پذیری سیستم در سیستم‌های یک‌تکه به‌صورت همه یا هیچ است، اما در معماری مایکروسرویس امکان مقیاس‌پذیری موثر به ازای هر سرویس دلخواه میسر است. بنابراین هر بخش از سیستم که بار(load) کاری بیشتری داشته باشد، متناسبا می‌تواند منابع پردازشی بیشتر نیز دراختیار گیرد و نیازی نیست برای همه مولفه‌های سیستم مقیاس‌پذیری یکنواخت انجام شود.

توسعه و تغییرات:

- امکان تغییر در منطق هر سرویس بدون نگرانی از تاثیرات منفی در سایر سرویس‌ها به دلیل خودمختاری سرویس‌ها ساده‌تر است، این موضوع البته برای تغییر منطق داخلی است و در صورتیکه واسط سرویس تغییر کند باید به سایر سرویس‌ها(استفاده‌کنندگان) اطلاع داده شود.

استفاده مجدد و ترکیب‌پذیری:

- به‌مانند معماری سرویس‌گرا یکی از اهداف اصلی از توسعه سرویس‌ها، امکان استفاده مجدد و ترکیب سرویس‌های موجود برای ایجاد سرویس‌های جدید است.

تطابق با معماری سازمان سرویس‌گرا:

- سازمان سرویس‌گرا صرفا در مشتری-محوری و ارایه سرویس باکیفیت به مشتریان خلاصه نمی‌شود بلکه موضوع مهم‌تر، معماری داخلی سازمان و نحوه چیدمان عناصر و منابع برای پویایی بیشتر و

این مقاله در سال ۱۳۹۸ منتشر شده است.

استقلال هر واحد از سایر واحدها است، معماری مایکروسرویس منطبق با معماری سازمانی سرویس‌گرا و تسهیل‌کننده آن است.

سهولت جابجایی و جایگزینی سرویس‌ها:

- با توجه به خودمختاری کارکردی و فناوری سرویس‌ها، به سادگی می‌توان یک سرویس را که عملکرد آن مناسب نبوده با نمونه بهتر جایگزین نمود یا یک سرویس را به‌تنهایی به یک محیط/سیستم دیگر منتقل نمود و از آن استفاده نمود

۲. متدها و ابزارهای مرتبط با معماری مایکروسرویس

۲-۱) Containers

کانتینرها(Containers)، یک روش استاندارد برای بسته‌بندی(Package) نرم‌افزار-سرویس و همه متعلقات و اجزاء آن است به‌صورتیکه بتوان آن را به محیط-پلتفرم‌های دیگری منتقل و بدون مشکل اجرا نمود. کانتینرها باعث می‌شوند فناوری داخل هر سرویس از محیط بیرون مخفی مانده و بدین صورت سرویس‌ها در عین استفاده از فناوری‌ها-سکوهای مختلف، از نظر بیرونی به روش استاندارد قابل استفاده و تعامل باشند.

از جمله ابزارها و فناوری‌های مدیریت کانتینرها مبتنی بر ابر(Cloud) :

- Kubernetes
- Docker Swarm
- Amazon ECS
- Azure Service Fabric
- Cloud Foundry
- Google Cloud Functions
- IBM Bluemix OpenWhisk
- Oracle Application Container

۲-۲) API Gateway

اگرچه هر سرویس با حداکثر خودمختاری مسوول انجام یک کارکرد مشخص در دامنه سیستم است اما مدیریت کل سیستم و انجام اموری نظیر احرازهویت، تقسیم‌بار، مجوزدهی، لاگ‌گیری، مانیتورینگ، و .. نیاز به یک مدیریت/ابزار متمرکز غیر از سرویس‌ها دارد که می‌تواند توسط API Gateway پیاده‌سازی شود،

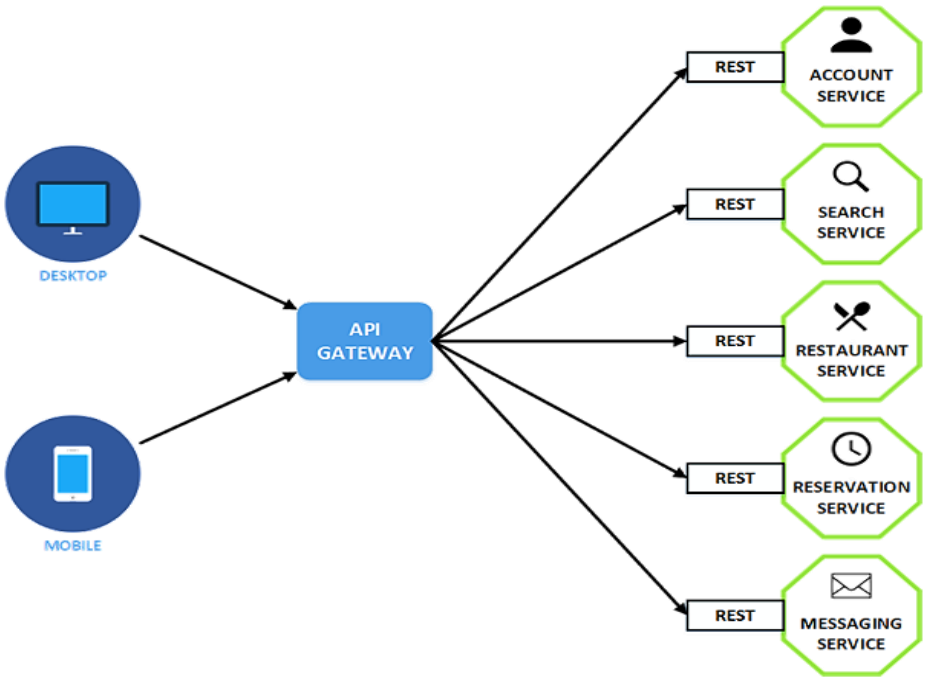
این مقاله در سال ۱۳۹۸ منتشر شده است.

درصورتیکه از این الگو(روش) استفاده نشود، کلیه موارد مدیریتی ذکرشده به صورت کاملا توزیع‌شده توسط سرویس‌ها مدیریت خواهد شد.

از جمله ابزارهای مطرح برای API Gateway :

- Apigee API Management
- Amazon API Gateway
- IBM API Connect
- WSO2 API Management
- MuleSoft Anypoint API Management
- Oracle API Manager
- Kong API Gateway
- Azure API Gateway

شکل زیر مثال ساده‌ای از یک سیستم مبتنی بر معماری مایکروسرویس متشکل از تعدادی سرویس خودمختار و یک دروازه(Gateway) دسترسی است.



شکل ۲-۱: کاربرد و جایگاه API Gateway

۲-۳) روش‌های احرازهویت و مجازشماری

از جمله روش‌های احراز هویت(Authentication) و مجازشماری(Authorization) :

- Token based solution
- 3rd party Authentication Federation
- API Gateway A&A services
- Cognito

۲-۴) روش‌های شناسایی و کشف سرویس

از جمله روش‌های شناسایی و کشف(Discovery) سرویس برای سرویس‌گیرندگان:

- Client Side-Service Discovery
- Application Load Balancer-based Service Discovery
- DNS-Based Service Discovery
- Service Discovery using ECS Event Stream

۲-۴) متدلوژی‌های تحلیل و طراحی

در حوزه مایکروسرویس‌ها متاسفانه متدلوژی‌های شناخته شده و مدونی برای تحلیل و طراحی سرویس وجود ندارد(متدلوژی حاصل سال‌ها تجربه آموزی است و به دلیل جدید بودن این معماری، در سال‌های آینده باید منتظر انتشار متدلوژی‌های مایکروسرویس باشیم)، اما مجموعه‌ای از منابع و رهنمودهای قابل استفاده را می‌توان به صورت زیر معرفی نمود:

- روش‌های طراحی مبتنی بر دامنه(DDD) خصوصا کتاب‌های آقای Eric Evans
- مانیفست واکنشگر (Reactive Manifesto)
- مفاهیم و تکنیک‌های DevOps از جمله Twelve-Factor App
- الگوهای کارکردی سرویس (Functional Decomposition Pattern)
- تئوری CAP

۲-۵) پلتفرم‌های طراحی و تولید

از جمله پلتفرم‌های طراحی و تولید مایکروسرویس‌ها:

- Ballerina
- Netflix OSS
- MSF4J

- Akka
- Vert.X
- Fabric8
- Cocaine
- Deis
- Lightbend
- OpenWhisk
- VAMP

علاوه بر موارد گفته شده، برای طراحی موفق در معماری مایکروسرویس نیاز به دانستن تکنیک‌ها، استانداردها و مفاهیم مهم دیگری نیز هست که تشریح آن‌ها از حوصله این نوشته خارج است و تنها به صورت تیتر-وار به آنها اشاره می‌شود: Bounded Context ، Conway's Law ، Platform Services ، SOEA ، MDM ، BCM ، Command Query Responsibility Segregation ، Data-Access pattern ، EDA ، NoSQL Data Design و ...

۳) چالش‌ها و راهکارهای پیشنهادی

معماری مایکروسرویس مبتنی بر اصول و مبانی کلان‌تری است که در پارادایم سرویس‌گرایی قرار دارد، ازآنجا که سرویس‌گرایی و اصول آن منطبق با نیازهای واقعی کسب‌وکار و فناوری است (و روز به روز نیز این نیازها مهم‌تر و گسترده‌تر می‌شود)، بکارگیری سرویس‌گرایی در معماری سازمان، سیستم‌های اطلاعاتی و به‌صورت کلان در حوزه فناوری اطلاعات نیز، نه یک موج گذرا که یک روند ادامه‌دار و آینده‌دار است؛ معماری مایکروسرویس نیز از این قاعده مستثنی نیست و باید انتظار داشت که این معماری و یا مشتقات بعدی آن همچنان در محور توجهات قرار گیرد، لذا مدیران و تصمیم‌گیران با نگاهی راهبردی به پارادایم سرویس‌گرایی و انواع فناوری‌ها-الگوهای پیاده‌سازی آن که در گذر زمان تکامل می‌یابند باید برنامه‌ای بلندمدت و هدفمند برای مدرن‌سازی سیستم‌ها و سرویس‌های فاوا متناسب با سرعت تغییرات کسب‌وکار داشته باشند.

از آنجاکه صرف تمایل مدیران و تصمیم‌گیران به استفاده از این معماری تضمین‌کننده موفقیت و نتایج واقعی نیست، به همین منظور ابتدا چالش‌های معماری مایکروسرویس مرور می‌شود و سپس راهکارهایی برای بکارگیری هدفمند این معماری جهت افزایش میزان موفقیت در طراحی و پیاده‌سازی سیستم‌های مبتنی بر مایکروسرویس‌ها ارایه می‌گردد.

۳-۱) چالش‌های معماری مایکروسرویس

چالش‌های پیاده‌سازی معماری مایکروسرویس را نمی‌توان نادیده گرفت، این چالش‌ها را می‌توان در سه دسته طبقه‌بندی نمود:

چالش‌های معماری:

- مانیتورینگ، مدیریت نسخه‌ها، تضمین امنیت و کنترل مایکروسرویس‌ها به مراتب سخت‌تر از سایر معماری‌ها و سیستم‌های متمرکز یا لایه‌ای است.
- پیاده‌سازی معماری‌های توزیع‌شده (با سرویس‌های خودمختار) نیاز به ابزارها و متدهای پیچیده‌تری برای پیاده‌سازی و نگهداشت دارد.
- به دلیل اینکه هر مایکروسرویس خودمختاری بالایی دارد، حجم زیادی از دوباره‌کاری در برنامه‌نویسی هر سرویس برای اعمال مکانیزم‌های کنترلی-امنیتی-مشترک باید انجام شود.

چالش‌های سازمانی:

- پیاده‌سازی معماری مایکروسرویس نیازمند نیروی انسانی بسیار ماهر و توانمند است که تامین آن برای هر سازمانی ممکن نیست.

این مقاله در سال ۱۳۹۸ منتشر شده است.

- ساختار سازمانی بیشتر واحدهای فناوری اطلاعات و تیم‌های توسعه نرم‌افزار مبتنی بر تقسیم افقی فناوری‌ها-ابزارها (UI-Application-Database) شکل گرفته است و تغییر این ساختار به تیم‌های مبتنی بر DevOps با یک دامنه محدود کسب‌وکار، سخت و زمان‌بر است.
- انتخاب سبک مناسب معماری بنا بر نیاز هر سازمان یکی از چالش‌های قدیمی است، هر صورت-مساله‌ای نیاز به راهکار مناسب خود دارد و قرار نیست هر سیستم کوچک یا بزرگی مبتنی بر مایکروسرویس باشد. انتخاب بین معماری‌ها-راهکارهای مختلف و نحوه یکپارچگی بین آن‌ها از جمله دغدغه‌های مدیران و تصمیم‌گیران است.

چالش‌های فناوری/ابزار:

- تنوع بیش‌از حد ابزارها و فناوری‌های جدید اگرچه یک مزیت است اما از نگاه دیگر تهدیدی برای تیم‌های برنامه‌نویسی است که در جنگلی از ابزارها سردرگم شده و تا مهارت کافی برای استفاده از یک ابزار را بدست می‌آورند، ابزارها و فناوری‌های جدیدی ظهور کرده و فرصت مسلط‌شدن کافی روی یک ابزار را از دست می‌دهند.
- ابزارهای جدید به‌سرعت ارایه می‌شوند و توسط برنامه‌نویسان در پروژه‌های واقعی مورد استفاده قرار می‌گیرند در حالیکه مسایل مهمی مانند امنیت و پایداری این ابزارها به‌صورت کامل ارزیابی نشده‌است.
- معماری مایکروسرویس مجموعه جدیدی از ابزارها-فناوری‌ها را برای پشتیبانی از نیازمندی‌های خود معرفی کرده است که خود باعث افزایش پیچیدگی و افزایش زمان و هزینه تولید سیستم می‌شود.

۳-۲) راهکارها و الزامات بکارگیری معماری مایکروسرویس

همانطور که در ابتدای این بخش گفته‌شد، پارادایم سرویس‌گرایی(به صورت عام) و معماری مایکروسرویس (به‌صورت خاص) روندهای آینده‌دار صنعت فاوا هستند که در دنیای رقابتی و سرویس-محور کسب‌وکار نیز تضمین‌کننده همراستایی بین کسب‌وکار با فاوا هستند، اما برای بکارگیری هدفمند و موفق این معماری، الزامات(قوانین) زیر پیشنهاد می‌شود:

- قانون اول اینکه همه‌جا و هر سیستمی نباید مایکروسرویسی باشد، برای هر صورت-مساله‌ای باید نیازمندی‌های معماری(وظیفه‌مندی-غیروظیفه‌مندی)، محدودیت‌ها-اجبارها و مجموعه شرایط محیطی-زمینه‌ای توسط معمار ارشد تحلیل شود و سبک(راهکار) مناسب معماری انتخاب شود.
- قانون دوم اینکه در صورت انتخاب معماری مایکروسرویس باید بدانیم که ضمن تبعیت از اصول اصلی این معماری، انتخاب‌های مهم و سختی وجود دارد که نحوه(مدل) معماری منطقی و فیزیکی سیستم

این مقاله در سال ۱۳۹۸ منتشر شده است.

را مشخص خواهد کرد، لذا معماری مایکروسرویس به اشکال و چیدمان‌های متنوعی قابل پیاده‌سازی است که انتخاب نامناسب در این مرحله نتایج جبران‌ناپذیری دارد.

- قانون سوم اینکه نباید فریب عناوین ابزار-فناوری-پروتکل را خورد؛ هر شرکت نرم‌افزاری که ادعا می‌کند محصولات مبتنی بر معماری مایکروسرویس تولید کرده باید راست آزمایی شود و صرف استفاده از چند ابزار و پلتفرم مرتبط با معماری مایکروسرویس به معنای تولید سیستم سرویس‌محور منطبق با اصول مایکروسرویس‌ها نیست(همچنان که هنوز بعد از گذشت بیش از یک دهه از ظهور معماری سرویس‌گرا، درصد کمی از سیستم‌هایی که با این عنوان طراحی و فروخته شده‌اند، واقعا طبق اصول سرویس‌گرایی طراحی و پیاده‌سازی شده‌اند)
- قانون چهارم اینکه "استقرار و به‌نتیجه‌رسیدن" یک سبک معماری جدید در سازمان نیازمند تغییر فرهنگ و معماری کسب‌وکار در آن سازمان است، تحقق ایده‌ال معماری مایکروسرویس نیازمند بازنگری معماری سازمانی با رویکرد سرویس‌گرا است. تغییرات می‌توان از هر دو طرف شروع شود و به طرف دیگر سرایت نماید. یکی از دلایل موفقیت شرکت‌های Netflix، Amazon در پیاده‌سازی معماری مایکروسرویس، ماهیت سرویس‌گرا کسب‌وکار این شرکت‌ها ارزیابی می‌شود.
- قانون پنجم اینکه انتخاب و دانه‌بندی مناسب سرویس‌ها از مهمترین عوامل موفقیت در طراحی و استقرار معماری مایکروسرویس است که متاسفانه متدلوژی‌های مدون و استانداردی هنوز برای آن ارائه نشده است، بنابراین سازمان‌ها(تیم‌ها) باید قبل از شروع پیاده‌سازی، نسبت به تدوین یک متدلوژی داخلی(بومی) از مجموعه مراجع-کتاب‌ها-تجارب داخلی یا بین‌المللی اقدام نمایند. رابطه معناداری بین عدم بکارگیری یک متدلوژی مناسب با شکست پروژه‌های سرویس‌گرا وجود دارد.

این مقاله در سال ۱۳۹۸ منتشر شده است.

۴) منابع و مراجع

Books:

1. Building Microservices, Sam Newman (2015)
2. Microservices for the Enterprise, Kasun Indrasiri & Prabath Siriwardena (2018)
3. Vertically Integrated Architectures, Jos Jong (2019)
4. Building Microservices with ASP.NET Core, Kevin Hoffman (2017)
5. Building Evolutionary Architectures: Support Constant Change, Neal Ford, Rebecca Parsons, and Patrick Kua (2017)

Papers and Presentations:

6. Essential Capabilities behind Microservices, Kim Kao (2019)
7. Microservices: Powered by Containers-as-a-Service, Chris Rosen (2017)
8. Implementing Microservices on Oracle Cloud, Lucas Jellema (2018)
9. Microservices, containers and event-driven architecture, Simon Green (2018)
10. Microservices: The Journey So Far and Challenges Ahead, Pooyan Jamshidi, C. Pahl, N. Mendonca, J. Lewis, S. Tilkov (2018)
11. MicroServices Architecture, Araf Karsh Hamid (2018)
12. Pragmatic approach to MicroServices, David Hymers (2019)
13. Microservices, Containers, Databases and Persistence Models, Kuassi Mensah and Paul Parkinson (2018)
14. Comparison of different architecture styles, Attila Balogh-Biró (2016)

Websites:

15. https://microservices.io
16. https://developer.ibm.com/technologies/microservices
17. https://www.mulesoft.com/resources/api/what-are-microservices
18. https://en.wikipedia.org/wiki/Microservices
19. https://www.redhat.com/en/topics/microservices
20. https://www.docker.com/solutions/microservices
21. https://aws.amazon.com/microservices/
22. https://martinfowler.com/articles/microservices.html
23. https://www.nginx.com/learn/microservices
24. https://docs.oracle.com/en/solutions/learn-architect-microservice/index.html
25. https://wso2.com
26. https://camunda.com/learn/whitepapers/microservices-and-bpm
27. https://dzone.com
28. https://dotnet.microsoft.com/learn/web/microservices-architecture

این مقاله در سال ۱۳۹۸ منتشر شده است.

29. https://www.tibco.com/solutions/microservices
30. https://www.infoq.com/microservices
31. https://www.microservices.com
32. https://www.cloudfoundry.org/microservices
33. https://samnewman.io

این مقاله در سال ۱۳۹۸ منتشر شده است.