**معماری میکروسرویس چیست؟**

یک معماری برای سیستم هایی است که قصد دارند به صورت توزیع شده کارکنند.به بیان ساده در یک پروژه میکروسرویسی مهم نیست شما با چه زبانی کار میکنید چون در نهایت باید سرویسی ارائه دهید تا با فراخوانی آن توسط قسمت های دیگر پروژه همکاری صورت گیرد.در مقابل پروژه هایی را داریم که به صورت Monolith و متمرکز پیاده شده اند.

**مزایا،معایب و چالش های معماری میکروسرویس**

1. قسمت عمده ای از پروژه مستقل می کنند و به صورت Loosly couple پیاده شده اند
2. از زبان ها و تکنولوژی های مختلف می توان برای پیاده سازی سرویس ها استفاده کرد.
3. برخلاف سیستم های متمرکز به دلیل توزیع پذیری بالای سیستم های میکروسرویسی ویژگی های زیر را ندارند:
4. رفتار قابل پیش بینی سرویس ها
5. موفقیت قابل پیش بینی سرویس ها
6. کارایی قابل پیش بینی سرویس ها

4. یکپارچی در تست و تست مختلط در این نوع سیستم ها دچار مشکل می شود.به بیان ساده تر در صورتی که یکی از سرویس ها دچار تغییرات ورژنی شود، ممکن است دیگر سرویس ها دچار خطا بشوند. در این صورت تنها راه حل مشکل،تعامل اعضای تیم برای رفع مشکل است.

5. مشکل بودن عیب یابی در این معماری

**اصول در معماری میکروسرویس**

1. Cohsion
2. Single Concern
3. Loose couple
4. Discrete بودن: به این معنا که سرویس ها به خوبی کپسوله سازی شده باشند. به عبارت دیگر سرویس ها باید مرز بندی مشخصی داشته باشند و آن را از سرویس های دیگر که هر کدام در Environment مخصوص خود که اجرا می شوند جدا کند.اصل Discrete بودن به این معنا نیز هست که هر سرویس باید در repository و source control و همچنین CICD مخصوص خود را نیز داشته باشد تا کاملا ایزوله شود. با رعایت این اصل برنامه قابلیت انتقال بهتری خواهد داشت.
5. Transportable بودن: هر سرویس باید بتواند از یک Runtime Environment به یک Environment دیگر انتقال یابد. در حال حاضر ساده ترین راه برای این کار استفاده از Container ها و داکر است و این موضوع با وجود اصل Discrete امکان پذیر است و راهکاری که برنامه نویس برای انتقال میکروسرویس باید ارائه دهد تغییر در Configuration و deployment است.
6. Autonomous یا خودمختاری سرویس ها : سرویس ها باید به صورت جداگانه deploy بشوند و یا روی PAAS قرار بگیرند تا به عنوان یک سرویس مورد استفاده قرار گیرند. در هر صورت در این معماری سعی می شود تا همه سرویس ها با هم روی یک ماشین قرار نگیرند و با هم Pack نشوند. در این صورت ما [رثق اثشی نسبتا بالایی خواهیم داشت ولی با این کار مدیریت سیستم های توزیع شده آسان تر خواهد بود.پ
7. Resilience یا تاب آوری: در طراحی یک میکروسرویس باید از خود بپرسیم سرویسی که طراحی کردیم آیا فاصله مناسبی با سایر سرویس ها دارد که در صورت بروز مشکل در سایر سرویس ها این سرویس به کار خود ادامه دهد یا خیر.
8. Observability یا قابل دیده شدن

**تجزیه پروژه به سرویس های مختلف**

بر اساس قابلیت های بیزینسی:

روش اول: توضیح سیستم به یک فرد غیر فنی

روش دوم: اثبات اینکه سیستم چه کاری را انجام می دهد و یا اثبات ارزش هر یک از قابلیت های سیستم

بر اساس دامنه و زیر دامنه (Domain and SubDomain)

3 نوع زیر دامنه داریم: Core, Supporting, Generic

: Core کلید اصلی طراحی دامنه پروژه است که در اکثر پروژه هایی که در یک دسته قرار دارند مشابه است مثل دامنه Product در فروشگاه اینترنتی

: Supporting

: General به بیزینس ارتباطی ندارد.

مقایسه دو روش بالا بر اساس استاندارد های معماری میکروسرویس:

|  |  |  |
| --- | --- | --- |
| بر اساس زیر دامنه | بر اساس بیزینس | مقایسه |
|  | Winner | Cohesion and loose couple |
| Winner |  | Size of microservice |
|  | Winner | Stablity of the design |
| Winner |  | Intuitive for engineer |

**نکته تکمیلی:** برای جداسازی سرویس ها بهترین راه وجود ندارد.

از پوشه 2 ویدیو سوم شروع کن ادامه بده این قسمت رو

**معماری رویدار محور:**

این معماری مدل Request-Response منسوخ دانسته شده ارسال درخواست ها با استفاده از Event و یک تکنولوژی توزیع Event انجام می شود.

**ویژگی های Event Driven**

در مدل Request-Response فرایند به صورت Synchronous و در EventDriven به صورت Asynchronous انجام می شود. از لحاظ Inversion of Controllدر مدل Request-Response فرستنده باید از گیرنده خبر داشته باشد، که در این صورت به آن وابسته است. اما در Event Driven به دلیل وجود یک لایه به نام Orchestration و توزیع کننده Event ها این وابستگی بین فرستنده و گیرنده وجود ندارد.