میکروسرویس ها :

تقسیم کردن ماژول های یک پروژه یا سرویس بزرگ به سرویس های کوچکتری که همواره باهم به شیوه های مختلف در ارتباط هستند را میکروسرویس میگوییم و این معماری در مقابل معماری یکپارچه قرار میگیرد

معماری میکروسرویس مزایای زیادی در رابطه با معماری یکپارچه یا monolithic دارد برای مثال برای هر قسمت میتوان از فریمورک و زبان جدا برای نوشتن سرویس مورد نظر استفاده کرد و هر قسمت دیتابیس مربوط به خود را داشته باشد ، اگر بخشی از پروژه به مشکل بخورد برخلاف معماری یکپارچه همه ی برنامه دچار مشکل نمیشود

البته این نوع معماری معایبی هم دارد که برای مثال در یک پروژه ی بزرگ با سرویس های زیاد مانیتورینگ تک تک سرویس ها وقت گیر و هزینه بر خواهد بود

تراکنش (Transaction) :

یک تراکنش در دیتابیس باید از چهار خاصیت استفاده کند :

Atomic: همه ی دستورات تراکنش یا اجرا شود و یا هیچکدام اجرا نشود و در صورت خطا به حالت اول باز گردد

Consistency: بعد از اجرای هر تراکنش یک دیتابیس باید از یک وضعیت درست و معتبر به وضعیت درست و معتبر دیگری برود

Isolation: هر دستور بدون توجه به دستور های دیگر اجرا شود و اجرای همزمان دستور ها روی هم تاثیر نگذارد

Durability: بعد از انجام مرحله commit و اتمام تراکنش داده های بصورت پایدار ماندگار شوند

تراکنش ها در محیط های توزیع شده :

در سیستم یکپارچه چون تراکنش ها دارای چهارویژگی بالا می باشند نگرانی در مورد سلامت داده ها وجود ندارد ، اما در معماری های میکروسرویس که هرکدام از دیتابیس ها جدا از هم در حال کار هستند و هرکدام تراکنش های خود را دارند و حتی ممکن است هرکدام باهم تفاوت داشته باشند نیازمند وجود الگو هایی هستیم که بتوانیم این مشکل را رفع کنیم و بتوانیم تمام تراکنش های سیستم میکروسرویس را مانند یک تراکنش واحد با همان خصوصیات بالا مدیریت کنیم .

برای این کار الگو های متفاوتی وجود دارد که دو مورد از آن ها 2pc و saga می باشند

الگوی 2PC (Two phase commit) :

همانطور که از نام این روش بر می آید commit کردن تراکنش ها در دو فاز انجام می شود . در این روش بخشی به نام coordinate وظیفه ی مدیریت تراکنش ها را خواهد داشت . در واقع شبیه به یک مرکز کنترل عمل میکند و طبق پاسخ هایی که سرور ها برای این مرکز کنترل میفرستند ، این مرکز دستوراتی را برای آن ها میفرستد ، بنابراین مدیریت commit یا rollback تراکنش ها بر عهده مرکز فرماندهی یا coordinator قرار میگیرد و این کار در دو فاز انجام می شود

در فاز اول مرکز کنترل پیامی مبنی بر شروع عملیات به سرور ها میفرستد و پس از قفل کردن داده های مورد نیاز عملیات شروع می شود و اگر به هردلیلی هر کدام از سرور ها به خطا بر بخورند مرکز کنترل پیام abort و rollback را به سرور ها می فرستد و اگر عملیات تمام سرور ها موفق باشد توسط هر کدام از سرور ها یک پیام مبنی بر موفقیت آمیز بودن عملیات برای مرکز کنترل می فرستند و وارد فاز دوم میشویم

در فاز دوم اگر تمام سرور ها در مرحله قبل موفق به انجام عملیات خود شدند از مرکز کنترل پیغامی برای commit کردن هرکدام از سرور ها فرستاده میشود و اگر همه سرور ها موفق به commit شدند عملیات به پایان خواهد رسید و اگر هرکدام از سرور ها در این میان موفق به commit نشوند پیغام rollback برای سرور ها فرستاده میشود و عملیات به صورت اول باز خواهد گشت

این روش نیز مشکلاتی دارد که برای مثال می توان به رد و بدل شدن زیاد پیام ، وابستگی کل سیستم به یک مرکز کنترل و پشتیبانی نشدن همه دیتابیس ها از الگوی 2pc اشاره کرد

الگوی Saga :

Choreography : در این روش بر خلاف روش 2pc که یک مرکز فرماندهی وظیفه مدیریت transaction ها را بر عهده داشت در این روش به جای اینکه مرکز فرماندهی تصمیم بگیرد و مشخص کند چه باید کرد هر میکروسرویس شرکت کننده خود به تنهایی تصمیم میگیرد که بر مبنای وضعیت موجود خود که آیا موفق به commit و یا دچار مشکل شد چه event را publish کند و یا در صورت subscribe شدن چه عملیاتی را بر روی دادها خود انجام دهد یعنی منطق عملیات به جای آنکه در یکجا متمرکز

شـــده باشـــد ، ایـــن وظیفـــه بـــین تمـــامی میکروســـرویس هـــا پخـــش شـــده اســـت دیگـــر بصورت متمرکز و یک جا مشخص نخواهد بود.

مزایا و این روش شامل :

سادگی : این روش بر عکس روش دیگر ساده است هر زمان که تغییر business مانند ایجاد حذف و ویرایش و غیره رخ دارد کافی است که یک event را publish کنید

عدم وابستگی :در این روش چون بر مبنای message broker پیش رفتیم تمامی میکرو سرویس ها کامل از یکدیگر بی اطلاع می باشند

معایب این روش شامل :

درک سخت فرآیند : به دلیل اینکه مرکز فرماندهی وجود ندارد و منطق فرآیند در تمامی سرویس ها پخش شده است درک مطلب دشوار خواهد بود برا برنامه نویسان و حتما عیب یابی و توسعه آن نیز سخت تر خواهد بود

به وجود آمدن حلقه: اگر دقت نکنیم ممکن است که حلقه به وجود بیاوریم که هیچگاه پایان نپذیرد . سرویس ها تا بی نهایت در یک حلقه گیر کنند.

Orchestration-based: در این روش برخلاف روش قبلی یک مرکزفرماندهی orchestration وظیفه مدیریت فرآینید saga بر مبنای پیام هایی که بصورت async به این مرکز می رسد را بر عهده دارد و بر مبنای آن تصمیم میگیرد که microservice شرکت کننده در saga چه عملیات را انجام دهد. در این روش هر شرکت کننده پس از انجام کار خود یک پیام به مرکز فرماندهی صادر میکند و منتظر پیام بعد خواهد شد و در این مرکز فرماندهی تصمیم خواهد گرفت که در گام بعد چه میکروسرویسی شرکت کننده در این فرآیند چه عملیاتی را باید انجام دهد.