مطالعات علوم کاربردی در مهندسی

دوره ۱۰، شماره ۳، پاییز ۱۴۰۳

صفحات ۸۲-۷۲

Online ISSN: ۲۴۷۶-۴۵۰۷

Print ISSN: ۲۵۳۸-۲۰۵۵

www.irijournals.com

توسعه نرم‌افزارهای مقیاس بزرگ با استفاده از معماری‌های میکروسرویس

پرنیا اعتبارزاده

دانشجوی کارشناسی مهندسی کامپیوتر گرایش نرم افزار، دانشگاه ازاد اصفهان واحد خوراسگان، اصفهان ایران

چکیده

توسعه نرم‌افزارهای مقیاس بزرگ با استفاده از معماری‌های میکروسرویس، انقلابی در نحوه طراحی و پیاده‌سازی سیستم‌ها به شمار می‌آید. این رویکرد به تیم‌ها این امکان را می‌دهد که به جای ایجاد یک سیستم واحد و بزرگ، اجزای مجزای کوچکی را طراحی کنند که هر یک وظایف خاص خود را بر عهده دارند. با این تفکیک، تغییرات و به‌روزرسانی‌ها به سادگی و با کمترین تأثیر بر سایر قسمت‌ها انجام می‌شود. این انعطاف‌پذیری نه تنها به بهبود کارایی کمک می‌کند، بلکه امکان مقیاس‌پذیری را نیز به طرز چشمگیری افزایش می‌دهد. به این ترتیب، هر میکروسرویس می‌تواند به طور مستقل توسعه، آزمایش و استقرار یابد و این امر موجب تسریع در فرآیند توسعه می‌شود. علاوه بر این، معماری میکروسرویس به تیم‌ها این امکان را می‌دهد که از فناوری‌های مختلف در هر میکروسرویس استفاده کنند، به طوری که می‌توانند با توجه به نیازهای خاص هر بخش، بهترین ابزارها و زبان‌ها را انتخاب کنند. این رویکرد به کاهش وابستگی‌ها و افزایش توانایی نوآوری کمک می‌کند. همچنین، با استقرار میکروسرویس‌ها بر روی زیرساخت‌های ابری، دسترسی و مقیاس‌پذیری به طرز چشمگیری بهبود می‌یابد. در نهایت، این مدل توسعه، نه تنها چالش‌های مقیاس‌پذیری را برطرف می‌کند، بلکه به سازمان‌ها اجازه می‌دهد تا به سرعت به تغییرات بازار پاسخ دهند و تجربه بهتری برای کاربران خود فراهم آورند.

واژه‌های کلیدی: توسعه نرم‌افزار، مقیاس بزرگ، معماری میکروسرویس، طراحی سیستم‌ها

دوره ۱۰، شماره ۳، پاییز ۱۴۰۳، صفحات ۸۲-۷۲

مقدمه

در دنیای فناوری اطلاعات، توسعه نرم‌افزارهای مقیاس بزرگ به یکی از چالش‌های اصلی برای سازمان‌ها و شرکت‌ها تبدیل شده است. با پیشرفت سریع تکنولوژی و افزایش نیاز به پاسخگویی به تقاضاهای کاربران، نیاز به رویکردهای نوین در طراحی و پیاده‌سازی نرم‌افزارها بیش از پیش احساس می‌شود. یکی از این رویکردها، استفاده از معماری میکروسرویس است که به دلیل قابلیت مقیاس‌پذیری، انعطاف‌پذیری و توانایی در مدیریت پیچیدگی سیستم‌ها، به محبوبیت زیادی دست یافته است. این مقاله به بررسی جوانب مختلف توسعه نرم‌افزارهای مقیاس بزرگ با استفاده از معماری میکروسرویس می‌پردازد. در این راستا، مواردی چون تعریف میکروسرویس‌ها، مزایای آن‌ها، چالش‌ها و روش‌های پیاده‌سازی، ابزارها و تکنولوژی‌های مرتبط و بهترین شیوه‌ها مورد بررسی قرار خواهد گرفت. در نهایت، نتیجه‌گیری‌هایی درباره آینده این معماری و تأثیر آن بر توسعه نرم‌افزارها ارائه می‌شود.

میکروسرویس‌ها، به عنوان واحدهای مستقل و کوچک نرم‌افزاری، به توسعه‌دهندگان این امکان را می‌دهند که هر سرویس را به‌طور جداگانه طراحی، توسعه و مستقر کنند. این ویژگی، نه‌تنها زمان توسعه را کاهش می‌دهد، بلکه به تیم‌ها این اجازه را می‌دهد تا با تمرکز بیشتری بر روی نیازهای خاص هر بخش از سیستم کار کنند. به عبارتی، میکروسرویس‌ها به سازمان‌ها کمک می‌کنند تا بتوانند به‌سرعت به تغییرات بازار و نیازهای کاربران پاسخ دهند. یکی از مزایای کلیدی معماری میکروسرویس، امکان استفاده از فناوری‌های مختلف برای هر سرویس است. به عنوان مثال، یک میکروسرویس می‌تواند با زبان برنامه‌نویسی خاصی توسعه یابد، در حالی که میکروسرویس دیگری ممکن است بر اساس فناوری‌های کاملاً متفاوتی بنا شود. این تنوع، به تیم‌ها این امکان را می‌دهد که از بهترین ابزارها و تکنیک‌ها برای حل چالش‌های خاص خود بهره‌برداری کنند.[1]

با این حال، پیاده‌سازی میکروسرویس‌ها بدون چالش نیست. مدیریت ارتباطات بین سرویس‌ها، اطمینان از امنیت داده‌ها و حفظ یکپارچگی در کل سیستم از جمله مسائلی هستند که توسعه‌دهندگان با آن مواجه می‌شوند. همچنین، فرایند نظارت و عیب‌یابی در یک محیط میکروسرویسی پیچیده‌تر از سیستم‌های متمرکز است و نیاز به ابزارهای پیشرفته‌تری دارد. در راستای غلبه بر این چالش‌ها، استفاده از ابزارهای مدیریت میکروسرویس و پلتفرم‌های اورکستراسیون مانند Kubernetes و Docker به شدت توصیه می‌شود. این تکنولوژی‌ها نه‌تنها به مدیریت بهتر سرویس‌ها کمک می‌کنند، بلکه امکان مقیاس‌پذیری و استقرار خودکار را نیز فراهم می‌آورند.[2]

بهترین شیوه‌ها در توسعه میکروسرویس‌ها شامل طراحی APIهای قوی، مستند کردن دقیق سرویس‌ها و پیاده‌سازی استراتژی‌های تست خودکار است. پیروی از این شیوه‌ها می‌تواند به کاهش خطاها و افزایش کیفیت نرم‌افزار کمک کند. در پایان، می‌توان گفت که معماری میکروسرویس به عنوان یک رویکرد نوین، نه‌تنها به سازمان‌ها اجازه می‌دهد تا در برابر تغییرات سریع بازار انعطاف‌پذیری بیشتری داشته باشند، بلکه به آن‌ها این امکان را می‌دهد که با ایجاد سیستم‌های پیچیده‌تر، به ارائه خدمات بهتر و سریع‌تر به کاربران خود بپردازند. با توجه به روند فعلی فناوری، به نظر می‌رسد که آینده درخشان و پر از فرصت‌هایی برای توسعه میکروسرویس‌ها در انتظار ماست.[3]

تعریف میکروسرویس‌ها

میکروسرویس‌ها، رویکردی در طراحی نرم‌افزار هستند که در آن، یک برنامه به مجموعه‌ای از سرویس‌های کوچک و مستقل تقسیم می‌شود. هر کدام از این سرویس‌ها وظیفه خاصی را انجام می‌دهند و از طریق API (رابط برنامه‌نویسی کاربردی) با

یکدیگر ارتباط برقرار می‌کنند. این رویکرد به برنامه‌نویسان این امکان را می‌دهد که به‌راحتی هر سرویس را به‌صورت مستقل توسعه، آزمایش و استقرار دهند. این تقسیم‌بندی به تیم‌های توسعه اجازه می‌دهد تا به‌طور موازی روی بخش‌های مختلف نرم‌افزار کار کنند، که این امر به تسریع در روند توسعه و کاهش زمان تحویل محصول نهایی منجر می‌شود. به‌علاوه، میکروسرویس‌ها به راحتی مقیاس‌پذیرند؛ به این معنا که می‌توانند به‌صورت جداگانه افزایش یا کاهش یابند، بدون آنکه بر عملکرد کل سیستم تأثیر بگذارند.

از دیگر مزایای این معماری، قابلیت جایگزینی آسان سرویس‌هاست. اگر یکی از میکروسرویس‌ها نیاز به به‌روزرسانی یا تغییر داشته باشد، می‌توان آن را به‌راحتی با نسخه جدیدی جایگزین کرد، بدون آنکه نیاز به توقف سایر بخش‌های سیستم باشد. این ویژگی باعث افزایش پایداری و انعطاف‌پذیری نرم‌افزار می‌شود. علاوه بر این، میکروسرویس‌ها به توسعه‌دهندگان اجازه می‌دهند از فناوری‌ها و زبان‌های برنامه‌نویسی مختلف برای هر خدمت استفاده کنند. این تنوع موجب می‌شود که تیم‌ها بتوانند بهترین ابزارها را برای حل مسائل خاص انتخاب کنند و به این ترتیب، کارایی و عملکرد بهتری را به ارمغان آورند.[4]

با این حال، چالش‌هایی نیز در پی اجرای این رویکرد وجود دارد. مدیریت ارتباطات بین میکروسرویس‌ها، پیاده‌سازی استراتژی‌های امنیتی و نظارت بر عملکرد هر سرویس به دقت و برنامه‌ریزی نیاز دارد. همچنین، در صورت عدم هماهنگی مناسب، ممکن است مشکلاتی در هم‌زمانی داده‌ها و یکپارچگی سیستم ایجاد شود. در نهایت، میکروسرویس‌ها به عنوان یک الگوی نوین در توسعه نرم‌افزار، نه تنها به بهبود فرآیند توسعه و استقرار کمک می‌کنند، بلکه به شرکت‌ها این امکان را می‌دهند که به‌سرعت به تغییرات بازار و نیازهای کاربران پاسخ دهند. این رویکرد، با بهره‌مندی از قابلیت‌های مدرن، می‌تواند به یک مزیت رقابتی تبدیل شود که در دنیای دیجیتال امروز، حیاتی است.[5]

مزایای استفاده از معماری میکروسرویس

۱. مقیاس‌پذیری

یکی از بزرگ‌ترین مزایای میکروسرویس‌ها، قابلیت مقیاس‌پذیری آن‌هاست. سازمان‌ها می‌توانند به‌راحتی سرویس‌هایی را که نیاز به منابع بیشتری دارند، مقیاس‌گذاری کنند بدون اینکه بر روی سایر بخش‌ها تأثیر بگذارند. این ویژگی به ویژه در زمان‌هایی که ترافیک کاربران به‌طور ناگهانی افزایش می‌یابد، بسیار مهم است. یکی از بزرگ‌ترین مزایای میکروسرویس‌ها، قابلیت مقیاس‌پذیری بالای آن‌هاست که به سازمان‌ها این امکان را می‌دهد تا به‌راحتی سرویس‌هایی را که به منابع بیشتری نیاز دارند، افزایش دهند. این ویژگی به توسعه‌دهندگان اجازه می‌دهد تا با تمرکز بر روی بخش‌های خاص، به‌سرعت به نیازهای متغیر بازار پاسخ دهند. به عنوان مثال، در زمان‌هایی که تعداد کاربران به‌طور ناگهانی افزایش می‌یابد، سازمان می‌تواند بدون ایجاد اختلال در عملکرد دیگر بخش‌ها، منابع لازم را به سرویس‌های پرتقاضا اختصاص دهد. این امر نه تنها تجربه کاربری را بهبود می‌بخشد بلکه به صرفه‌جویی در هزینه‌ها و منابع نیز منجر می‌شود.[6]

علاوه بر این، مقیاس‌پذیری میکروسرویس‌ها به تیم‌ها این امکان را می‌دهد تا به‌طور مستقل بر روی هر سرویس کار کنند و در نتیجه توسعه و استقرار سریع‌تری را فراهم کنند. این رویکرد، سازمان‌ها را قادر می‌سازد تا به‌راحتی به تغییرات بازار و نیازهای مشتریان پاسخ دهند، بدون آنکه مجبور باشند کل سیستم را تغییر دهند یا تحت تأثیر مسائل دیگر قرار گیرند. بدین ترتیب، میکروسرویس‌ها به عنوان یک راهکار مؤثر برای مدیریت بار ترافیکی و افزایش کارایی در دنیای دیجیتال امروز شناخته می‌شوند.

۲. افزایش انعطاف‌پذیری

معماری میکروسرویس این امکان را فراهم می‌کند که تیم‌های توسعه بتوانند به‌طور مستقل بر روی بخش‌های مختلف برنامه کار کنند. این استقلال به تیم‌ها این اجازه را می‌دهد که از فناوری‌ها و زبان‌های مختلف برای توسعه هر سرویس استفاده کنند و در نتیجه، نوآوری و پیشرفت سریع‌تری به وجود آید. معماری میکروسرویس به تیم‌های توسعه این امکان را می‌دهد که به‌طور مستقل بر روی اجزای مختلف یک برنامه کار کنند. این استقلال به هر تیم اجازه می‌دهد تا با استفاده از فناوری‌ها و زبان‌های برنامه‌نویسی متفاوت، به طراحی و پیاده‌سازی سرویس‌های خاص خود بپردازد. در نتیجه، این رویکرد به کاهش وابستگی‌ها و افزایش انعطاف‌پذیری منجر می‌شود و نوآوری را در فرایند توسعه تسریع می‌کند.[7]

همچنین، این معماری موجب می‌شود که تیم‌ها بتوانند به سرعت به نیازهای متغیر بازار پاسخ دهند. هر سروری که به‌طور مستقل توسعه یافته، قابلیت ارتقاء و به‌روزرسانی بدون تأثیر بر دیگر اجزا را داراست. این ویژگی‌ها نه‌تنها به بهبود کارایی و عملکرد سیستم کمک می‌کند، بلکه منجر به افزایش رضایت مشتریان می‌شود، چرا که می‌توانند از به‌روزرسانی‌های مکرر و بهبودهای مستمر بهره‌مند شوند.[8]

۳. بهبود نگهداری و به‌روزرسانی

از آنجا که میکروسرویس‌ها به‌صورت مستقل عمل می‌کنند، به‌روزرسانی یا نگهداری یک سرویس خاص بدون تأثیر بر سایر بخش‌ها امکان‌پذیر است. این ویژگی باعث کاهش زمان Downtime و افزایش بهره‌وری می‌شود. میکروسرویس‌ها به عنوان اجزای مستقل در یک سیستم عمل می‌کنند و این امکان را فراهم می‌آورند که هر سرویس به‌طور جداگانه به‌روزرسانی یا نگهداری شود. این ویژگی به توسعه‌دهندگان اجازه می‌دهد تا بدون ایجاد اختلال در عملکرد سایر بخش‌ها، تغییرات لازم را اعمال کنند. در نتیجه، زمان Downtime به حداقل می‌رسد و کاربران تجربه بهتری از خدمات دریافت می‌کنند. با این رویکرد، سازمان‌ها می‌توانند به سرعت به نیازهای بازار واکنش نشان دهند و در عین حال، کیفیت خدمات را حفظ کنند.[9]

علاوه بر این، قابلیت مستقل بودن میکروسرویس‌ها باعث می‌شود که تیم‌های مختلف توسعه، به‌طور موازی روی چندین سرویس کار کنند. این موضوع نه‌تنها موجب افزایش سرعت توسعه و زمان تحویل می‌گردد، بلکه به بهبود بهره‌وری کلی سازمان نیز کمک می‌کند. با کاهش وابستگی‌ها و ارتقاء قابلیت‌های مقیاس‌پذیری، میکروسرویس‌ها به کسب‌وکارها این امکان را می‌دهند که به‌راحتی با تغییرات و چالش‌های جدید سازگار شوند و در نتیجه، موقعیت رقابتی خود را تقویت کنند.[10]

روش‌های پیاده‌سازی موفق میکروسرویس‌ها

- **انتخاب فناوری مناسب**

انتخاب فناوری مناسب برای پیاده‌سازی میکروسرویس‌ها می‌تواند تأثیر زیادی بر عملکرد و مقیاس‌پذیری آن‌ها داشته باشد. ابزارها و فریمورک‌هایی مانند Spring Boot، Docker و Kubernetes به توسعه‌دهندگان کمک می‌کند تا میکروسرویس‌ها را به‌راحتی پیاده‌سازی کنند. انتخاب فناوری مناسب برای پیاده‌سازی میکروسرویس‌ها، نقشی کلیدی در عملکرد و مقیاس‌پذیری سیستم‌ها ایفا می‌کند. ابزارها و فریمورک‌هایی که در این زمینه به کار می‌روند، به توسعه‌دهندگان این امکان را می‌دهند تا با سهولت بیشتری به طراحی و پیاده‌سازی میکروسرویس‌ها بپردازند. به‌عنوان مثال، Spring Boot با فراهم آوردن ساختارهای ساده و استاندارد، توسعه‌دهندگان را قادر می‌سازد تا به سرعت سرویس‌های قوی و مقیاس‌پذیر را

ایجاد کنند. همچنین، Docker با قابلیت بسته‌بندی و توزیع آسان، فرآیند استقرار میکروسرویس‌ها را به شدت تسهیل می‌کند و Kubernetes نیز به مدیریت و هماهنگی این کانتینرها در مقیاس بزرگ کمک می‌کند.[11]

با استفاده از این فناوری‌ها، می‌توان به سادگی چالش‌های مربوط به مقیاس‌پذیری و نگهداری را پشت سر گذاشت. به عنوان مثال، با قدرت Kubernetes در مدیریت اتوماسیون و مقیاس‌پذیری، توسعه‌دهندگان می‌توانند به سرعت به نیازهای متغیر بازار پاسخ دهند و منابع را بهینه کنند. در نتیجه، انتخاب صحیح این ابزارها نه تنها به بهبود عملکرد میکروسرویس‌ها کمک می‌کند، بلکه زمینه‌ساز رشد و توسعه پایدار در پروژه‌های نرم‌افزاری خواهد بود.[12]

- **استفاده از CI/CD**

استفاده از روش‌های CI/CD (ادغام مداوم و تحویل مداوم) باعث می‌شود که به‌روزرسانی‌های مکرر و تست‌های خودکار به‌راحتی انجام شود. این رویکرد به توسعه‌دهندگان این امکان را می‌دهد که به‌سرعت به بازخوردها پاسخ دهند و کیفیت نرم‌افزار را بهبود بخشند. روش‌های CI/CD (ادغام مداوم و تحویل مداوم) به‌عنوان یک استراتژی کلیدی در فرآیند توسعه نرم‌افزار، امکان به‌روزرسانی‌های مکرر و انجام تست‌های خودکار را به‌طور مؤثری فراهم می‌آورند. این رویکرد نوآورانه به توسعه‌دهندگان امکان می‌دهد تا به‌سرعت به تغییرات و نیازهای کاربران پاسخ دهند و در نتیجه، زمان لازم برای انتشار ویژگی‌های جدید به حداقل برسد. با بهره‌گیری از ابزارهای اتوماسیون، تیم‌ها می‌توانند بر کیفیت کد نظارت داشته باشند و اطمینان حاصل کنند که هر تغییر به‌راحتی ادغام می‌شود و بدون بروز مشکلات جدی به مرحله تولید می‌رسد.[13]

این فرآیند نه‌تنها به تسریع چرخه توسعه کمک می‌کند، بلکه به بهبود مداوم کیفیت نرم‌افزار نیز می‌انجامد. تست‌های خودکار که در طول مراحل مختلف توسعه انجام می‌شوند، به شناسایی زودهنگام مشکلات و اشکالات کمک می‌کنند. به این ترتیب، تیم‌ها می‌توانند با اطمینان بیشتری به عرضه محصولات جدید بپردازند و به این ترتیب، تجربه کاربری بهتری را برای مشتریان فراهم آورند. در نهایت، این رویکرد به توسعه‌دهندگان اجازه می‌دهد تا با تمرکز بر نوآوری و بهبود مستمر، به اهداف تجاری خود دست یابند.[14]

- **طراحی API مناسب**

طراحی API مناسب و مستندات دقیق آن بسیار مهم است. این کار به توسعه‌دهندگان اجازه می‌دهد تا به‌راحتی با یکدیگر ارتباط برقرار کنند و از قابلیت‌های مختلف سرویس‌ها بهره‌برداری کنند. طراحی یک API مناسب به‌عنوان بنیادی برای ارتباطات میان سیستم‌ها و برنامه‌ها عمل می‌کند. این طراحی باید به گونه‌ای باشد که علاوه بر سادگی و کاربرپسندی، قابلیت‌های متنوعی را نیز ارائه دهد. یک API کارآمد باید به توسعه‌دهندگان این امکان را بدهد که به راحتی با آن تعامل کنند و بتوانند به سرعت ویژگی‌های مختلف آن را شناسایی و استفاده کنند. انتخاب نام‌های معنادار برای متدها، تعیین دقیق پارامترها و خروجی‌ها، و ارائه روش‌های مناسب برای مدیریت خطاها، از جنبه‌های کلیدی در این فرآیند به شمار می‌روند.[15]

مستندات دقیق و جامع، مکمل ضروری برای طراحی API به حساب می‌آید. این مستندات باید شامل توضیحات واضحی درباره عملکرد هر بخش از API، نمونه‌های کاربردی و راهنماهای تصویری باشد تا توسعه‌دهندگان بتوانند به سرعت به اطلاعات مورد نیاز دسترسی پیدا کنند. همچنین، به‌روزرسانی مستمر مستندات به همراه تغییرات API، تضمین‌کننده انطباق با نیازهای روزافزون کاربران خواهد بود. در نهایت، یک API با طراحی صحیح و مستندات مناسب، می‌تواند به تسهیل همکاری میان تیم‌ها و افزایش بهره‌وری در پروژه‌ها کمک شایانی نماید.[16]

دوره ۱۰، شماره ۳، پاییز ۱۴۰۳، صفحات ۸۲-۷۲

ابزارها و تکنولوژی‌های مرتبط

برای پیاده‌سازی میکروسرویس‌ها، ابزارها و تکنولوژی‌های متعددی وجود دارند که می‌توانند به بهبود عملکرد و مدیریت این معماری کمک کنند. برخی از این ابزارها شامل:

- Docker: برای بسته‌بندی و استقرار میکروسرویس‌ها در کانتینرها.
- Kubernetes: برای مدیریت و اورکستراسیون کانتینرها.
- Istio: برای مدیریت ترافیک و امنیت بین میکروسرویس‌ها.
- Prometheus و Grafana: برای نظارت و تحلیل عملکرد سیستم.

علاوه بر این ابزارها، گزینه‌های دیگری نیز وجود دارند که می‌توانند به ارتقاء قابلیت‌های میکروسرویس‌ها کمک کنند.

- Spring Cloud: این مجموعه از ابزارها، توسعه‌دهندگان جاوا را قادر می‌سازد تا به سادگی میکروسرویس‌هایی با ویژگی‌های مقیاس‌پذیری و قابلیت تعامل بالا ایجاد کنند.
- Apache Kafka: یک پلتفرم توزیع‌شده برای مدیریت جریان داده‌ها است که امکان تبادل اطلاعات بین میکروسرویس‌ها را با سرعت و کارایی بالا فراهم می‌کند.
- Consul: این ابزار به عنوان یک سرویس کشف و مدیریت پیکربندی عمل می‌کند و به میکروسرویس‌ها اجازه می‌دهد تا به راحتی یکدیگر را شناسایی کنند و تعامل داشته باشند.
- ELK Stack (Elasticsearch, Logstash, Kibana): مجموعه‌ای از ابزارها برای جمع‌آوری، ذخیره‌سازی و تجزیه و تحلیل لاگ‌ها که می‌تواند به شفافیت و درک بهتر از رفتار سیستم کمک کند.

علاوه بر این، مفاهیمی همچون Serverless Architecture نیز به سرعت در حال گسترش هستند که به توسعه‌دهندگان اجازه می‌دهد تا بدون نیاز به مدیریت سرور، بر روی کد و توسعه ویژگی‌های جدید تمرکز کنند. در نهایت، استفاده از API Gateway نیز به عنوان یک نقطه ورودی مرکزی برای مدیریت ترافیک و درخواست‌ها بین میکروسرویس‌ها می‌تواند کارایی را افزایش دهد. این ابزارها و تکنیک‌ها، در کنار یکدیگر، زمینه را برای ایجاد سیستم‌های مقیاس‌پذیر، قابل اطمینان و کارآمد فراهم می‌کنند که می‌توانند به نیازهای متغیر کسب‌وکارها پاسخ دهند. توجه به این نکته ضروری است که انتخاب ابزار مناسب بستگی به نیازها و شرایط خاص هر پروژه دارد. هر کدام از این فناوری‌ها ویژگی‌ها و مزایای خاص خود را دارند و شناخت صحیح از آن‌ها می‌تواند به بهینه‌سازی فرآیند توسعه و کاهش هزینه‌ها منجر شود. به طور کلی، رویکردی تلفیقی و انعطاف‌پذیر در انتخاب ابزارها می‌تواند به تحقق اهداف تجاری و فنی کمک شایانی نماید.[17,18]

بهترین شیوه‌ها در توسعه میکروسرویس‌ها

- **انتخاب صحیح دامنه سرویس‌ها**

تقسیم‌بندی دقیق دامنه‌ها و مسئولیت‌ها برای هر میکروسرویس از اهمیت بالایی برخوردار است. این کار به جلوگیری از تداخل و پیچیدگی‌های اضافی کمک می‌کند. تقسیم‌بندی واضح دامنه‌ها و مسئولیت‌ها در هر میکروسرویس، کلید موفقیت در طراحی سیستم‌های مقیاس‌پذیر و قابل نگهداری است. این فرآیند به مهندسان نرم‌افزار این امکان را می‌دهد که به وضوح مشخص کنند که هر میکروسرویس چه وظایفی را بر عهده دارد و به چه نوع داده‌هایی دسترسی پیدا می‌کند. با این رویکرد، از بروز

تداخل عملکردی بین میکروسرویس‌ها جلوگیری می‌شود و هر تیم می‌تواند بر روی حوزه مشخصی متمرکز شود. این موضوع نه تنها موجب افزایش کارایی و کاهش احتمال خطا می‌شود، بلکه به بهبود روند توسعه نیز کمک می‌کند.[19]

علاوه بر این، تقسیم‌بندی دقیق دامنه‌ها، به تسهیل فرآیند نگهداری و به‌روزرسانی سیستم کمک می‌کند. وقتی وظایف هر میکروسرویس به‌طور واضح تعریف شده باشد، توسعه‌دهندگان می‌توانند به راحتی تغییرات لازم را اعمال کنند بدون آنکه نگران تأثیر منفی بر سایر بخش‌ها باشند. این نوع سازماندهی، به کاهش پیچیدگی‌های اضافی و افزایش قابلیت فهم کد کمک می‌کند و در نتیجه، زمان مورد نیاز برای توسعه و آزمایش را کاهش می‌دهد. در نهایت، این شیوه نه تنها به بهینه‌سازی عملکرد سیستم کمک می‌کند، بلکه به تسهیل همکاری بین تیم‌ها نیز می‌انجامد.[20]

- **استفاده از الگوهای طراحی**

استفاده از الگوهای طراحی مانند Circuit Breaker و API Gateway می‌تواند به طور چشمگیری عملکرد و قابلیت اطمینان میکروسرویس‌ها را بهبود بخشد. الگوی Circuit Breaker با ایجاد یک مکانیسم حفاظتی در برابر خطاها، از بروز مشکلات مکرر جلوگیری می‌کند. این الگو به سیستم اجازه می‌دهد تا در صورت شناسایی ناکامی‌های متوالی در یک سرویس خاص، به طور موقت درخواست‌ها را متوقف کند و در نتیجه از فشار اضافی بر روی سایر اجزا بکاهد. در نتیجه، این روش نه تنها از وقوع خرابی‌های بیشتر جلوگیری می‌کند، بلکه زمان بازیابی و رفع مشکل را نیز کاهش می‌دهد.[21]

از سوی دیگر، API Gateway به عنوان دروازه‌ای برای مدیریت ترافیک ورودی به میکروسرویس‌ها عمل می‌کند. این الگو امکان کنترل و نظارت بر درخواست‌ها را فراهم می‌سازد و می‌تواند وظایف متعددی مانند احراز هویت، مسیریابی و تجمع پاسخ‌ها را به عهده گیرد. به کمک API Gateway، توسعه‌دهندگان قادر خواهند بود تا بار ترافیکی را به طور مؤثرتری مدیریت کرده و از پیچیدگی‌های تعاملات بین سرویس‌ها بکاهند. این رویکرد، نه تنها به بهبود کارایی سیستم کمک می‌کند، بلکه تجربه کاربری بهتری را نیز برای کاربران نهایی به ارمغان می‌آورد.[22]

- **مدیریت داده‌ها**

مدیریت داده‌ها در معماری میکروسرویس‌ها به عنوان یکی از چالش‌های کلیدی شناخته می‌شود. در این مدل، هر سرویس به طور مستقل عمل کرده و می‌تواند نیازهای خاص خود را برآورده سازد. استفاده از الگوی "Database per Service" به این معناست که هر میکروسرویس پایگاه داده مخصوص به خود را دارد که این امر می‌تواند به جداسازی داده‌ها و کاهش وابستگی‌ها میان سرویس‌ها کمک کند. این جداسازی نه تنها به بهبود مقیاس‌پذیری می‌انجامد، بلکه امکان تطبیق سریع‌تر با تغییرات و نیازهای متنوع کسب‌وکار را نیز فراهم می‌آورد.[23]

با این حال، این رویکرد چالش‌هایی نیز به همراه دارد. یکی از مشکلات اصلی، همگام‌سازی داده‌ها و مدیریت تراکنش‌ها در سطح سرویس‌هاست. به دلیل عدم وجود یک پایگاه داده مرکزی، ایجاد یک روند یکپارچه برای دسترسی و به‌روزرسانی داده‌ها می‌تواند پیچیده باشد. برای حل این مشکلات، توسعه‌دهندگان ممکن است از الگوهایی مانند Saga یا Event Sourcing بهره ببرند که به مدیریت پیچیدگی‌ها و اطمینان از یکپارچگی داده‌ها در سطح کل سیستم کمک می‌کند. در نهایت، انتخاب رویکرد مناسب بستگی به نیازهای خاص پروژه و مقیاس آن دارد.[24]

دوره ۱۰، شماره ۳، پاییز ۱۴۰۳، صفحات ۸۲-۷۲

آینده معماری میکروسرویس‌ها

با توجه به روندهای رو به رشد در توسعه نرم‌افزار و نیاز به مقیاس‌پذیری و انعطاف‌پذیری، می‌توان انتظار داشت که معماری میکروسرویس‌ها در آینده بیشتر از قبل مورد توجه قرار گیرد. همچنین، نوآوری‌هایی مانند سرویس‌های بدون سرور (Serverless) و استفاده از هوش مصنوعی می‌تواند به بهینه‌سازی و تسریع فرایند توسعه کمک کند. این تحولات نه‌تنها به بهبود کارایی منجر می‌شوند، بلکه به سازمان‌ها این امکان را می‌دهند که به سرعت به تغییرات بازار پاسخ دهند و نیازهای مشتریان را به بهترین شکل ممکن برآورده سازند. در این راستا، میکروسرویس‌ها به عنوان یک راهکار مناسب برای تفکیک وظایف و تسهیل فرآیند توسعه شناخته می‌شوند. هر میکروسرویس می‌تواند به صورت مستقل توسعه یابد و به روزرسانی شود، که این امر موجب کاهش زمان توقف سیستم و افزایش قابلیت اطمینان می‌گردد.[25]

علاوه بر این، استفاده از سرویس‌های بدون سرور به تیم‌ها این امکان را می‌دهد که بر روی نوآوری و توسعه ویژگی‌های جدید تمرکز کنند، بدون آنکه نگران مدیریت زیرساخت‌ها باشند. این شیوه جدید، به ویژه برای استارتاپ‌ها و کسب‌وکارهای کوچک که به منابع مالی و انسانی محدودی دسترسی دارند، می‌تواند تحولی شگرف ایجاد کند. همچنین، هوش مصنوعی با تحلیل داده‌ها و یادگیری ماشین می‌تواند به شناسایی الگوهای پیچیده و پیش‌بینی نیازهای مشتریان کمک کند. این تکنولوژی‌ها نه‌تنها به بهینه‌سازی فرآیندها می‌پردازند، بلکه با ارائه راهکارهای شخصی‌سازی‌شده، تجربه کاربری را به سطحی جدید ارتقاء می‌دهند. در نهایت، آینده توسعه نرم‌افزار به سمت ایجاد یک اکوسیستم متصل و هوشمند پیش می‌رود که در آن، همکاری میان میکروسرویس‌ها، سرویس‌های بدون سرور و هوش مصنوعی به یکدیگر کمک می‌کند تا سازمان‌ها بتوانند به بهترین نحو عملکرد خود را بهبود بخشند. این تحولات نه‌تنها بر روی توسعه نرم‌افزار تأثیر می‌گذارند، بلکه به تغییر فرهنگ سازمانی و نحوه تعامل تیم‌ها نیز کمک شایانی خواهند کرد.[26,27]

نتیجه‌گیری

توسعه نرم‌افزارهای مقیاس بزرگ با استفاده از معماری میکروسرویس به‌عنوان یک رویکرد نوین و کارآمد می‌تواند به سازمان‌ها کمک کند تا به‌سرعت به نیازهای بازار پاسخ دهند و در عین حال کیفیت و قابلیت اطمینان سیستم‌های خود را حفظ کنند. با وجود چالش‌های موجود، رعایت بهترین شیوه‌ها و استفاده از ابزارهای مناسب می‌تواند به موفقیت این رویکرد کمک کند. در نهایت، با پیشرفت‌های فناوری و افزایش تمرکز بر روی مقیاس‌پذیری، می‌توان انتظار داشت که میکروسرویس‌ها به بخش جدایی‌ناپذیر از توسعه نرم‌افزار تبدیل شوند.

در این میان، توانایی تقسیم‌بندی سیستم به اجزای کوچک و مستقل، امکان به‌روزرسانی و نگهداری آسان‌تر را فراهم می‌آورد. هر میکروسرویس به‌طور جداگانه می‌تواند توسعه یابد، آزمایش شود و مستقر گردد، که این امر باعث تسهیل در فرآیندهای تیم‌های مختلف می‌شود. به‌علاوه، این رویکرد به سازمان‌ها این امکان را می‌دهد که در صورت نیاز، به سرعت منابع خود را مقیاس‌گذاری کنند و به سادگی بر روی اجزای خاصی از سیستم تمرکز کنند. از سوی دیگر، میکروسرویس‌ها به دلیل ماهیت توزیع‌شده خود، نیاز به مدیریت قوی‌تری دارند. برای مثال، استفاده از ابزارهای نظارت و رصد می‌تواند به شناسایی مشکلات و اختلالات در سیستم کمک کند و در نتیجه، بهبود کارایی و کاهش زمان خرابی را به همراه داشته باشد. همچنین، پیاده‌سازی الگوهای ارتباطی مؤثر، مانند APIهای سبک و پروتکل‌های غیرهمزمان، می‌تواند تعامل میان میکروسرویس‌ها را تسهیل کند و از بار اضافی بر روی سرورها بکاهد.

از دیگر مزایای قابل توجه این معماری، قابلیت یکپارچه‌سازی با فناوری‌های جدید است. به‌عنوان مثال، سازمان‌ها می‌توانند به‌راحتی از خدمات ابری، داده‌های کلان و هوش مصنوعی بهره‌برداری کنند و این امکان را برایشان فراهم می‌آورد که به‌طور مداوم نوآوری کنند و به نیازهای متغیر مشتریان پاسخ دهند. اما در کنار این مزایا، چالش‌هایی نیز وجود دارد. مدیریت پیچیدگی‌های ناشی از ارتباطات میان میکروسرویس‌ها و اطمینان از سازگاری و یکپارچگی داده‌ها، از جمله مسائلی است که باید به‌طور جدی مورد توجه قرار گیرد. سازمان‌ها باید با استفاده از ابزارهای مناسب، مانند سیستم‌های مدیریت کانتینر و اورکستراسیون، این چالش‌ها را به حداقل برسانند و به حداکثر بهره‌وری دست یابند. با توجه به این تحولات و تغییرات سریع در دنیای فناوری، به نظر می‌رسد که آینده توسعه نرم‌افزار به سمت افزایش استفاده از معماری میکروسرویس‌ها پیش می‌رود. این رویکرد نه تنها به سازمان‌ها کمک می‌کند تا در برابر تغییرات بازار انعطاف‌پذیرتر باشند، بلکه امکان نوآوری مستمر و ارائه خدمات بهتر به مشتریان را نیز فراهم می‌آورد. بنابراین، با شناخت صحیح از این معماری و به‌کارگیری شیوه‌های مناسب، سازمان‌ها می‌توانند در مسیر موفقیت و رقابت در دنیای دیجیتال گام بردارند.

منابع:

1. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Commun. Surv. Tutor. 2015, 17, 2347–2376.
2. Cisco Annual Internet Report, 2018–2023. Available online: https://www.cisco.com/c/en/us/solutions/collateral/executiveperspectives/annual-internet-report/white-paper-c11-741490.html (accessed on 25 February 2022).
3. The Internet of Things 2020. Available online: https://www.businessinsider.com/internet-of-things-report (accessed on 25 February 2022).
4. How You Contribute to Today's Growing DataSphere and Its Enterprise Impact. Available online: https://blogs.idc.com/2019/11/04/how-you-contribute-to-todays-growing-datasphere-and-its-enterprise-impact/ (accessed on 25 February 2022).
5. Butzin, B.; Golatowski, F.; Timmermann, D. Microservices approach for the internet of things. In Proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, Germany, 1–6 September 2016; IEEE: Piscataway, NJ, USA; pp. 1–6.
6. Pattern: Microservice Architecture. Available online: https://microservices.io/patterns/microservices.html (accessed on 2 March 2022).
7. Kul, S.; Sayar, A. A Survey of Publish/Subscribe Middleware Systems for Microservice Communication. In Proceedings of the 2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 21–23 October 2021; IEEE: Piscataway, NJ, USA; pp. 781–785.
8. Newman, S. Building Microservices; O'Reilly Media, Inc.: Neewtown, MA, USA, 2021.
9. Baldoni, R.; Contenti, M.; Virgillito, A. The evolution of publish/subscribe communication systems. In Future Directions in Distributed Computing; Springer: Berlin/Heidelberg, Germany, 2003; pp. 137–141.
10. Eugster, P.T.; Felber, P.A.; Guerraoui, R.; Kermarrec, A.M. The many faces of publish/subscribe. ACM Comput. Surv. CSUR 2003, 35, 114–131.
11. Razzaque, M.A.; Milojevic-Jevric, M.; Palade, A.; Clarke, S. Middleware for internet of things: A survey. IEEE Int. Things J. 2015, 3, 70–95.
12. Shi, Y.; Zhang, Y.; Jacobsen, H.-A.; Tang, L.; Elliott, G.; Zhang, G.; Chen, X.; Chen, J. Using Machine Learning to Provide Reliable Differentiated Services for IoT in SDN-Like Publish/Subscribe Middleware. Sensors 2019, 19, 1449.
13. Sun, L.; Li, Y.; Memon, R. A. An open IoT framework based on microservices architecture. China Commun. 2017, 14, 154–162.
14. Khazaei, H.; Bannazadeh, H.; Leon-Garcia, A. End-to-end management of IoT applications. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017; IEEE: Piscataway, NJ, USA; pp. 1–3.
15. Villari, M.; Fazio, M.; Dustdar, S.; Rana, O.; Chen, L.; Ranjan, R. Software defined membrane: Policy-driven edge and internet of things security. IEEE Cloud Comput. 2017, 4, 92–99.

16. Datta, S.K.; Bonnet, C. Next-generation, data centric and end-to-end iot architecture based on microservices. In Proceedings of the 2018 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), Jeju, Republic of Korea, 24–26 June 2018; IEEE: Piscataway, NJ, USA; pp. 206–212.
17. Alam, M.; Rufino, J.; Ferreira, J.; Ahmed, S.H.; Shah, N.; Chen, Y. Orchestration of microservices for IoT using Docker and edge computing. IEEE Commun. Mag. 2018, 56, 118–123.
18. Lv, W.; Wang, Q.; Yang, P.; Ding, Y.; Yi, B.; Wang, Z.; Lin, C. Microservice Deployment in Edge Computing Based on Deep Q Learning. IEEE Trans. Parallel Distrib. Syst. 2022, 33, 2968–2978.
19. Cherradi, G.; Bouziri, A.E.; Boulmakoul, A.; Zeitouni, K. Real-time microservices based environmental sensors system for Hazmat transportation networks monitoring. Transp. Res. Procedia 2017, 27, 873–880.
20. Alanezi, K.; Mishra, S. Utilizing Microservices Architecture for Enhanced Service Sharing in IoT Edge Environments. IEEE Access 2022, 10, 90034–90044.
21. Rahimian, F.; Girdzijauskas, S.; Payberah, A.H.; Haridi, S. Vitis: A Gossip-based Hybrid Overlay for Internet-scale Publish/Subscribe Enabling Rendezvous Routing in Unstructured Overlay Networks. In Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, Anchorage, AK, USA, 16–20 May 2011; pp. 746–757.
22. Chockler, G.; Melamed, R.; Tock, Y.; Vitenberg, R. Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication. In Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, Toronto, ON, Canada, 20–22 June 2007; ACM: New York, NY, USA, 2007; pp. 14–25.
23. Girdzijauskas, S.; Chockler, G.; Vigfusson, Y.; Tock, Y.; Melamed, R. Magnet: Practical subscription clustering for internet-scale publish/subscribe. In Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems (DEBS), Cambridge, UK, 12–15 July 2010.
24. Gascon-Samson, J.; Garcia, F.; Kemme, B.; Kienzle, J. Dynamoth: A Scalable Pub/Sub Middleware for Latency-Constrained Applications in the Cloud. In Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems, Columbus, OH, USA, 29 June–2 July 2015; pp. 486–496.
25. An, K.; Khare, S.; Gokhale, A.; Hakiri, A. An autonomous and dynamic coordination and discovery service for wide-area peer-to-peer publish/subscribe: Experience paper. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, Barcelona, Spain, 19–23 June 2017; pp. 239–248.
26. Atlam, H.F.; Walters, R.J.; Wills, G.B. Fog computing and the internet of things: A review. Big Data Cognit. Comput. 2018, 2, 10.
27. Campello, R.J.; Moulavi, D.; Sander, J. Density-based clustering based on hierarchical density estimates. In Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, Gold Coast, Australia, 14–17 April 2013.