## مقدمه

الگوهای معماری طراحی روش‌های سنجیده‌شده و آزمون‌پس‌داده برای ساختاردادن به سیستم‌های نرم‌افزار پیچیده می‌باشند که راه‌حل‌هایی را برای مسائل و مشکلات تکرارشونده در حین طراحی ارائه می‌دهند. استفاده از این الگوها، منجر به مقیاس‌پذیری نرم‌افزارها، سهولت نگه‌داری از آنها و در نهایت کاهش هزینه‌ها و بهبود محصول نهایی خواهد شد. تمرکز اصلی این مقاله بر توضیح معماری خاصی به نام **معماری رویداد محور** می‌باشد.

معماری رویداد محور (Event-driven architecture یا EDA به اختصار) الگویی برای طراحی نرم‌افزارها می‌باشد که برای ساخت برنامه‌های مقیاس‌پذیر[1] و کم‌وقفه[2] استفاده می‌شود. در این معماری، به جای اتصال مستقیم اجزای مختلف سیستم به یک‌دیگر، از یک جور سیستم پیام‌رسان استفاده می‌شود که اطلاعات مرتبط با رخدادها توزیع کرده و هر جزء وابسته به کاربرد خود، به برخی رویدادها توجه می‌کند.

## معماری رویداد محور چیست؟

در EDA، هر اتفاق یا تغییر قابل‌توجه یک رویداد به حساب می‌آید. برای مثال، کلیک موس کاربر و یا افزودن اقلام به سبد خرید، هر دو رویداد به حساب می‌آیند. این رویدادها به محض رخداد، توسط یک سیستم میانجی معروف به **کارگزار رویداد**[3] بین اجزاء مرتبط سیستم توزیع می‌شوند؛ سپس، هر جزء وابسته به کارکرد خود، پس از بررسی هر رخداد تصمیم می‌گیرد که واکنشی نشان بدهد یا خیر. این تکنیک برای برقراری ارتباط میان اجزاء، با دو نکته‌ی مثبت اساسی همراه است:

1. از آنجا که اجزاء مختلف کاملاً از هم جداشده شده‌اند (یا به اصطلاح Decouple شده‌اند)، مقیاس‌دهی، توسعه و راه‌اندازی هر بخش به طور مستقل از دیگر بخش‌ها به راحتی قابل‌انجام خواهد بود.
2. این سیستم پیام‌رسان که مانند باس سخت‌افزاری مادربرد عمل می‌کند (و اتفاقا به آن Event BUS هم گفته می‌شود)، در عمل باعث تسریع اجزاء می‌شود، چون اجزاء به طور ناهمزمان[4] می‌توانند به رویدادها پاسخ دهند.[1]

معماری رویداد محور یک بخش اساسی از برنامه‌های مدرنی است که بر پایه‌ی ریزسرویس‌ها[5] ساخته شده‌اند. هر ریزسرویس می‌تواند خود از اجزاء مختلفی تشکیل‌شده باشد و زبان برنامه‌نویسی توسعه‌ی هر بخش با دیگری فرق کند. در اینجا، کارگزارهای رویداد با مدیریت روابط بین این اجزاء جدا از پیاده‌سازی آنها (اعم از زبان برنامه‌نویسی، تکنولوژی، ...) این اجازه را می‌دهند که کارایی اصلی هر زیربرنامه را بتوان بدون اضطراب حول نحوه‌ی توزیع رویدادها توسعه داد. [2]

## تاریخچه و انگیزه

اگرچه برنامه‌نویسی رویداد محور مفهوم جدیدی نیست، EDA به عنوان یک الگوی معماری در سال‌های ابتدایی هزاره‌ی دوم میلادی شهرت یافت.[3] این بازه دقیقاً مطابق با شروع افزایش محبوبیت ریزسرویس‌ها و افزایش حجم داده‌هایی مورد نیاز

---

[1] Scalable
[2] Low latency
[3] Event Broker
[4] Asynchronous
[5] Microservices

پردازش می‌باشد (عملاً عصری که اینترنت به‌همراه آورد).
معماری رویداد محور چندین دلیل قوی برای جایگزینی روش‌های قدیمی‌تر به همراه داشت:

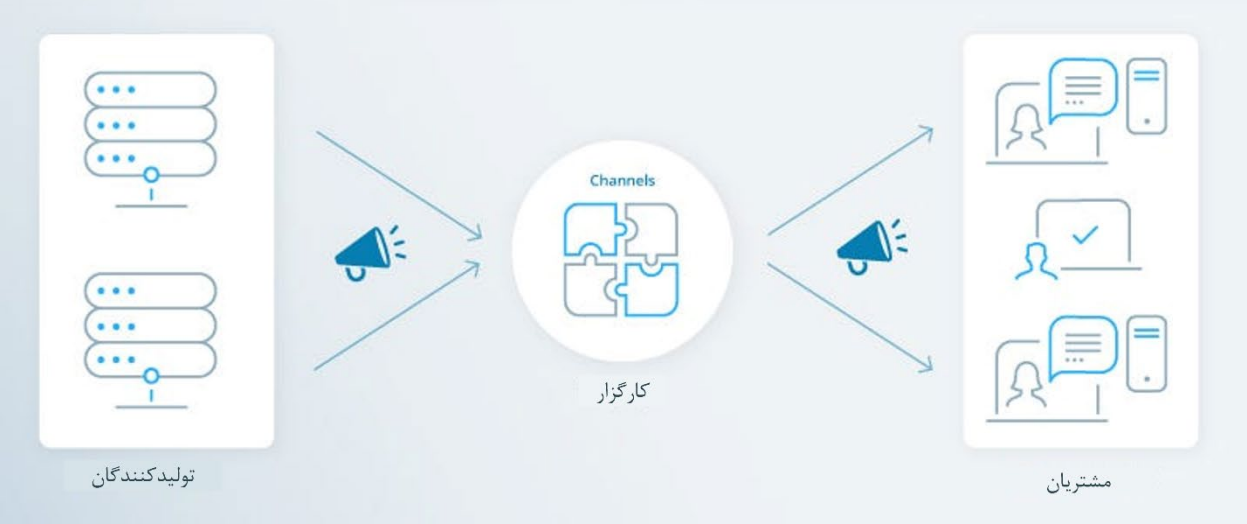
1. با افزایش حجم داده‌ها، نیاز به پردازش آنی آنها کاهش پیدا نکرد. جدا بودن اجزاء مختلف یک برنامه این قابلیت را مهیا کرد که قسمت‌هایی که تحت بار بیشتری بودند مقیاس داده شوند و بدون وابستگی به دیگر اجزاء توسعه داده شوند، که منجر به ارائه‌ی راه‌حل‌های موفق به این مشکل شد.
2. نکته‌ی دوم که در ادامه‌ی همین خصلت مستقل‌بودن اجزاء می‌باشد، مقاومت بالاتر سیستم در مقابل شکست‌ها[6] می‌باشد، زیرا در صورتی که یک جزء دچار مشکل اساسی شود، لزوماً کل سیستم از کار نمی‌افتد (یا به اصطلاح Down نمی‌شود).[4]
3. مزیت اصلی سوم در کاهش پیچیدگی پیاده‌سازی هر جزء از زاویه‌ای به جز جداسازی آن از دیگر اجزاء می‌باشد. از آنجا که هر جزء دیگر درگیر برنامه‌ریزی برای تعداد مختلفی داده‌ی ورودی نیست، کافیست برنامه برای مدیریت یک رویداد برنامه‌نویسی شود. به این نحو، وابستگی هر جزء به تعداد ورودی‌ها کمتر می‌شود و نیاز به توسعه‌ی نرم‌افزارهایی که بیش از حد کلی هستند (یعنی برای مثال برای هندل‌کردن چند میلیون کاربر طراحی‌شده‌اند در صورتی که شاید مدت زیادی طول بکشد تا این برنامه به این تعداد کاربر برسد) را کاهش می‌دهد و منجر به صرفه‌جویی در هزینه‌های تولید نرم‌افزار می‌شود.[5]

## اجزاء و برهمکنش‌ها

یک معماری رویداد محور در حالت کلی شامل سه بخش کلیدی می‌باشد:[6]

1. **تولیدکنندگان رویداد**[7]: همانطور که از اسمشان پیداست، اینها اجزایی هستند که رویداد تولید می‌کنند. برای مثال محیط کاربری، سنسورها، برنامه‌ها و یا ریزسرویس.
2. **جهت‌دهنده‌ی رویداد**[8]: این جزء همان کارگزار رویدادها می‌باشد که میان تولیدکنندگان رویداد و مشتریان رویداد میانجی‌گری می‌کند. وظیفه‌ی این جزء، دریافت رویدادها از تولیدکنندگان، فیلترکردن آنها بر اساس معیارهای از پیش مشخص و سپس ارسال آنها به مشتریان مرتبط است. دو نمونه از جهت‌دهنده‌های معروف (یا روترها) Apache Kafka و RabbitMQ می‌باشند.
3. **مشتریان رویداد**[9]: این اجزاء، رویدادها را دریافت کرده و آنها را پردازش می‌کنند. آنها می‌توانند برنامه‌ها، ریزسرویس‌ها و یا خود انسان‌ها باشند.

[6] Failure
[7] Event producers
[8] Event routers
[9] Event consumers




روابط بین این اجزاء غالباً به طور ناهمزمان می‌باشد. [7] هنگامی که یک رویداد رخ می‌دهد، تولیدکننده آن را به روتر رویدادها منتشر[10] می‌کند. سپس روتر این رویداد را فیلتر می‌کند و به مشتریان مرتبط می‌فرستد. مشتریان نیز می‌توانند مشترک[11] انواع مشخصی از رویدادها شوند. به این روند از انتقال پیام، استراتژی Publish-Subscribe یا به اختصار PubSub نیز گفته می‌شود.

بخش بسیار مهم دیگری علاوه بر این اجزاء اصلی، ویژگی‌های یک رویداد است که شامل موارد زیر است:

1. **زمان:** مهر زمانی[12] هنگامی که رویداد رخ داده است.
2. **منبع:** جزء یا موجودی که رویداد را تولید کرده است.
3. **کلید:** یک مشخص‌کننده‌ی منحصر به فرد برای رویداد.
4. **هدر**[13]: فرا اطلاعاتی[14] شامل نوع رویداد، نسخه[15]، مکان ثبت رویداد یا ...
5. **محموله**[16]: اطلاعات اصلی رویداد.

این خصلت‌ها به هر رویداد ماهیت و معنای مشخصی می‌دهند که به کارگزار و مشتریان امکان تحلیل و پردازش موثر آنها را می‌دهند.

## نقاط ضعف و قوت

تا کنون تعدادی از نقاط قوت این معماری را بیان کرده‌ایم. به طور مختصر این موارد را مرور می‌کنیم:[8]

[10] Publish
[11] Subscriber
[12] Timestamp
[13] Header
[14] Metadata
[15] Version
[16] Payload

1. **مقیاس‌پذیری:** معماری رویداد محور اجازه‌ی مقیاس‌دهی افقی[17] را می‌دهند، زیرا اجزاء از هم جدا هستند و در صورت نیاز، می‌توانیم صرفاً تعداد بیشتری از هر جزء مورد نیاز را اجرا کنیم.
2. **انعطاف:** به دلیل جدا بودن اجزاء، توسعه و تغییر اجزاء فعلی بدون دست‌کاری دیگر بخش‌های سیستم ممکن است که این باعث این می‌شود که به راحتی بتوان قابلیت‌های جدیدی را در قالب جزءهای جدید به برنامه اضافه کنیم.
3. **کاهش پیچیدگی در اجزاء:** در ادامه‌ی مورد دوم، عدم نیاز به تنظیم هم‌زمان چند جزء به توسعه‌دهنده اجازه می‌دهد که روی کارکردهای اصلی هر جزء به طور تنها تمرکز کند (و نه به نحوه‌ی ارتباط این جزء با دیگر اجزاء).
4. **پاسخ‌دهی آنی:** طبق ماهیت ناهمزمان دریافت و پردازش رویدادها، پاسخ‌دهی به رویدادهای متنوع می‌تواند بسیار سریع انجام شود، زیرا احتمال این که یک رویداد در یک خط لوله پشت سر اجزاء نامرتبط به خودش گیر کند کمتر است.

حال به بررسی برخی نقاط ضعف معماری رویداد محور می‌پردازیم:[9]

1. **پیچیدگی روتر:** کاهش پیچیدگی میسرشده توسط جداسازی اجزا که یکی از نقاط قوت این معماری بود، با هزینه‌ای پنهان رو به رو می‌باشد. از آنجا که هماهنگی بین اجزاء اکنون تنها به دست کارگزار رویدادها می‌باشد، ساخت کارگزاری که به نیازهای سیستم پاسخ دهد می‌تواند دشوار باشد، زیرا یک کارگزار موثر نه تنها باید تاخیر کمی در تحویل رویدادها داشته باشد، بلکه باید از الگوریتم‌ها و ابزاری برای ایجاد اطمینان حول سازگاری[18] رویدادها استفاده کند تا مثلاً این گونه نباشد که رویدادی پیش از رسیدن به مصرف‌کننده‌ی مرتبط از دست برود و یا یک رویداد چندین بار به مشتریان برسد.
2. **دیباگ‌کردن و رفع مشکلات:** مسئله‌ی دیگر، حل مشکلات و باگ‌هایی می‌باشد که در یک سیستم رویداد محور می‌تواند رخ دهند. از آنجا که پردازش رویدادها به طور ناهمزمان است، دنبال‌کردن وقایع در طی یک سیستم می‌تواند دشوار باشد و نیازمند ابزار مخصوص برای این کار است. نکته‌ی دیگر این است که اگرچه اجزاء به طور مستقیم به یک دیگر وابسته نیستند، اما ممکن است برخی اجزاء به رویدادهای تولیدشده توسط اجزایی دیگر حساس باشند و عدم کارکرد صحیح در اجزاء تولیدکننده، منجر به مشکل در اجزاء مصرف‌کننده شود اما تشخیص منشا مشکل سخت باشد.

## نمونه‌هایی از معماری رویداد محور

تعداد بسیاری از ابرشرکت‌های امروزی از معماری‌های رویداد محور استفاده می‌کنند:[10]

1. **Netflix**: شرکت Netflix از EDA برای مدیریت برهمکنش‌های پیچیده‌ی بین ریزسرویس‌های خود استفاده می‌کند. به خصوص، آنها از Apache Kafka به عنوان کارگزار رویدادها استفاده می‌کنند.
2. **Amazon**: شرکت Amazon برای مدیریت اقلام انبارها، به‌روزرسانی آنها و اتصال این آمار به وبسایت فروش، از EDA استفاده می‌کند.
3. **McDonald’s**[11]: شرکت McDonald’s نیز برای بهبود کارکرد نرم‌افزارهای فروش خود از EDA استفاده می‌کند.

[17] Horizontal
[18] Consistency

## معماری‌های مشابه

چندین معماری بسیار نزدیک دیگر به EDA موجود است که البته ایجاد تفکیک مشخص بین آنها دشوار می‌باشد. برای مثال، ریزسرویس‌ها خود یک نوع معماری به حساب می‌آیند اما ارتباط بسیار نزدیکی با معماری رویداد محور دارند و غالباً با یکدیگر استفاده می‌شوند. الگوهای PubSub و واکنشی (Reactive) نیز به همین شکل هستند.

یک الگوی طراحی معماری قابل مقایسه با معماری رویدادمحور (**Event-driven Architecture**) الگوی **معماری پیام‌محور (Message-Driven Architecture)** است.

هر دو الگو بر ارتباطات ناهمزمان و کاهش وابستگی اجزا تأکید دارند. در معماری رویدادمحور، رویدادها باعث انجام اعمال (Actions) می‌شوند و سیستم از طریق یک گذرگاه یا کارگزار رویداد به این رویدادها واکنش نشان می‌دهد و بر پاسخ‌های بلادرنگ تمرکز دارد. در مقابل، معماری پیام‌محور از پیام‌ها برای ارتباط بین سیستم‌های توزیع‌شده استفاده می‌کند و بر قابلیت اطمینان و تحویل تضمینی پیام‌ها (اغلب از طریق صف پیام) تأکید دارد. در حالی که معماری رویدادمحور برای سیستم‌های بلادرنگ و کم‌وابسته مانند اینترنت اشیاء مناسب است، معماری پیام‌محور برای سیستم‌هایی که نیاز به ارتباطات قابل اطمینان و تراکنشی دارند، مانند پردازش پرداخت، کاربرد دارد.

## جمع‌بندی

معماری رویداد محور یکی از راهکارهای پیشرفته برای طراحی سیستم‌های مقیاس‌پذیر و کارآمد است که به طور گسترده در حوزه‌های مختلف استفاده می‌شود. این معماری با کاهش وابستگی اجزا، امکان توسعه و مدیریت آسان‌تر سیستم را فراهم کرده و به پاسخ‌دهی سریع‌تر کمک می‌کند. هرچند که چالش‌هایی نظیر پیچیدگی مدیریت رویدادها و اشکال‌زدایی نیز وجود دارد، مزایای قابل توجه آن باعث شده است تا در بسیاری از شرکت‌ها و کاربردهای مدرن به کار گرفته شود. آینده این معماری با پیشرفت فناوری‌های مرتبط بسیار روشن به نظر می‌رسد.

# مراجع

[1] Confluent, "What Is Event-Driven Architecture?," Confluent, [Online]. Available: https://www.confluent.io/learn/event-driven-architecture/.

[2] Solace, "The Complete Guide to Event-Driven Architecture," [Online]. Available: https://solace.com/what-is-event-driven-architecture/.

[3] "Event Driven Architecture Definition," [Online]. Available: https://www.scylladb.com/glossary/event-driven-architecture/.

[4] Wikipedia, "Event-driven architecture," [Online]. Available: https://en.wikipedia.org/wiki/Event-driven_architecture.

[5] P. Tsilopoulos, "Event-Driven Architectures: Motivation," [Online]. Available: https://www.panos.tech/event-driven-0/.

[6] AWS, "What is an Event-Driven Architecture?," [Online]. Available: https://aws.amazon.com/event-driven-architecture/.

[7] Microsoft Azure, "Event-driven architecture style," [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven.

[8] M. Patel, "A Guide to Event-Driven Architecture Pros and Cons," [Online]. Available: https://solace.com/blog/event-driven-architecture-pros-and-cons/.

[9] R. Doshi, "Event-Driven Architecture : Benefits, Challenges, and Examples," [Online]. Available: https://medium.com/@rohitdoshi9/event-driven-architecture-benefits-challenges-and-examples-c957c269420a.

[10] Rising Wave, "Real-World Applications of Event-Driven Architecture: 10 Examples," [Online]. Available: https://risingwave.com/blog/real-world-applications-of-event-driven-architecture-10-examples/.

[11] D. Comartin, "McDonald’s Journey to Event-Driven Architecture," [Online]. Available: https://codeopinion.com/mcdonalds-journey-to-event-driven-architecture/.