**https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven**

**Event-driven architecture style**

An event-driven architecture consists of **event producers** that generate a stream of events, and **event consumers** that listen for the events.

معماری رویداد محور شامل تولیدکنندگان رویداد است که جریانی از رویدادها را تولید می کنند و مصرف کنندگان رویداد که به رویدادها گوش می دهند.

Events are delivered in near real time, so consumers can respond immediately to events as they occur. Producers are decoupled from consumers — a producer doesn't know which consumers are listening. Consumers are also decoupled from each other, and every consumer sees all of the events. This differs from a [Competing Consumers](https://docs.microsoft.com/en-us/azure/architecture/patterns/competing-consumers) pattern, where consumers pull messages from a queue and a message is processed just once (assuming no errors). In some systems, such as IoT, events must be ingested at very high volumes.

رویدادها تقریباً در زمان واقعی تحویل داده می شوند، بنابراین مصرف کنندگان می توانند بلافاصله به رویدادهایی که رخ می دهند پاسخ دهند. تولیدکنندگان از مصرف‌کنندگان جدا شده‌اند - یک تولیدکننده نمی‌داند کدام مصرف‌کننده به آن گوش می‌دهد. مصرف کنندگان نیز از یکدیگر جدا شده اند و هر مصرف کننده ای همه رویدادها را می بیند. این با الگوی مصرف‌کنندگان رقیب متفاوت است، که در آن مصرف‌کنندگان پیام‌ها را از یک صف می‌کشند و یک پیام فقط یک بار پردازش می‌شود (با فرض عدم وجود خطا). در برخی از سیستم‌ها، مانند اینترنت اشیا، رویدادها باید با حجم بسیار بالایی دریافت شوند.

* **Pub/sub**: The messaging infrastructure keeps track of subscriptions. When an event is published, it sends the event to each subscriber. After an event is received, it cannot be replayed, and new subscribers do not see the event.
* **Event streaming**: Events are written to a log. Events are strictly ordered (within a partition) and durable. Clients don't subscribe to the stream, instead a client can read from any part of the stream. The client is responsible for advancing its position in the stream. That means a client can join at any time, and can replay events.

Pub/Sub: زیرساخت پیام‌رسانی اشتراک‌ها را پیگیری می‌کند. هنگامی که یک رویداد منتشر می شود، رویداد را برای هر مشترک ارسال می کند. پس از دریافت یک رویداد، نمی توان آن را دوباره پخش کرد و مشترکین جدید رویداد را نمی بینند.

جریان رویداد: رویدادها در یک گزارش نوشته می شوند. رویدادها کاملاً سفارشی (در یک پارتیشن) و بادوام هستند. مشتریان در جریان مشترک نمی شوند، در عوض مشتری می تواند از هر بخشی از جریان بخواند. مشتری مسئول پیشبرد موقعیت خود در جریان است. این بدان معناست که یک مشتری می تواند در هر زمان بپیوندد و می تواند رویدادها را دوباره پخش کند.

On the consumer side, there are some common variations:

* **Simple event processing**. An event immediately triggers an action in the consumer. For example, you could use Azure Functions with a Service Bus trigger, so that a function executes whenever a message is published to a Service Bus topic.
* **Complex event processing**. A consumer processes a series of events, looking for patterns in the event data, using a technology such as Azure Stream Analytics or Apache Storm. For example, you could aggregate readings from an embedded device over a time window, and generate a notification if the moving average crosses a certain threshold.
* **Event stream processing**. Use a data streaming platform, such as Azure IoT Hub or Apache Kafka, as a pipeline to ingest events and feed them to stream processors. The stream processors act to process or transform the stream. There may be multiple stream processors for different subsystems of the application. This approach is a good fit for IoT workloads.

در سمت مصرف کننده، برخی از تغییرات رایج وجود دارد:

پردازش رویداد ساده یک رویداد بلافاصله باعث ایجاد یک عمل در مصرف کننده می شود. برای مثال، می‌توانید از توابع Azure با یک تریگر Service Bus استفاده کنید، به طوری که هر زمان که پیامی در یک موضوع Service Bus منتشر می‌شود، یک تابع اجرا می‌شود.

پردازش رویداد پیچیده یک مصرف کننده با استفاده از فناوری هایی مانند Azure Stream Analytics یا Apache Storm، مجموعه ای از رویدادها را پردازش می کند و به دنبال الگوهایی در داده های رویداد می گردد. برای مثال، می‌توانید خوانش‌های یک دستگاه تعبیه‌شده را در یک پنجره زمانی جمع‌آوری کنید و اگر میانگین متحرک از آستانه خاصی عبور کرد، اعلان ایجاد کنید.

پردازش جریان رویداد. از یک پلت فرم جریان داده، مانند Azure IoT Hub یا Apache Kafka، به عنوان خط لوله برای جذب رویدادها و تغذیه آنها به پردازشگرها استفاده کنید. پردازشگرهای جریان برای پردازش یا تبدیل جریان عمل می کنند. ممکن است چندین پردازنده جریانی برای زیرسیستم های مختلف برنامه وجود داشته باشد. این رویکرد برای حجم کاری اینترنت اشیا مناسب است.

The source of the events may be external to the system, such as physical devices in an IoT solution. In that case, the system must be able to ingest the data at the volume and throughput that is required by the data source.

In the logical diagram above, each type of consumer is shown as a single box. In practice, it's common to have multiple instances of a consumer, to avoid having the consumer become a single point of failure in system. Multiple instances might also be necessary to handle the volume and frequency of events. Also, a single consumer might process events on multiple threads. This can create challenges if events must be processed in order or require exactly-once semantics. See [Minimize Coordination](https://docs.microsoft.com/en-us/azure/architecture/guide/design-principles/minimize-coordination).

منبع رویدادها ممکن است خارج از سیستم باشد، مانند دستگاه های فیزیکی در یک راه حل IoT. در آن صورت، سیستم باید بتواند داده ها را با حجم و توانی که منبع داده مورد نیاز است، دریافت کند.

در نمودار منطقی بالا، هر نوع مصرف کننده به صورت یک جعبه نشان داده شده است. در عمل، برای جلوگیری از تبدیل شدن مصرف کننده به یک نقطه شکست در سیستم، داشتن چندین نمونه از یک مصرف کننده معمول است. همچنین ممکن است چندین نمونه برای کنترل حجم و فراوانی رویدادها ضروری باشد. همچنین، یک مصرف کننده واحد ممکن است رویدادها را در چندین رشته پردازش کند. اگر رویدادها باید به ترتیب پردازش شوند یا دقیقاً یک بار معنایی نیاز داشته باشند، می تواند چالش هایی ایجاد کند. به حداقل رساندن هماهنگی مراجعه کنید.

**When to use this architecture**

* Multiple subsystems must process the same events.
* Real-time processing with minimum time lag.
* Complex event processing, such as pattern matching or aggregation over time windows.
* High volume and high velocity of data, such as IoT.

زمان استفاده از این معماری

چندین زیرسیستم باید رویدادهای یکسانی را پردازش کنند.

پردازش بلادرنگ با حداقل تاخیر زمانی

پردازش رویداد پیچیده، مانند تطبیق الگو یا تجمیع در طول پنجره های زمانی.

حجم بالا و سرعت بالای داده مانند اینترنت اشیا.