The Architect's Guide to Real-Time Retail

How retail IT professionals can architect for resilience and rapid response by leveraging real-time data streaming and event-driven architecture

Sumeet Puri
Chief Technology Solutions Officer

solace.

As retailers strive to stay connected to their customers, they must embrace a digital-first mindset and upgrade their IT systems to provide real-time responses to consumers. Event-driven architecture is the enabling paradigm for this mindset."

About Sumeet Puri

Sumeet Puri is the Chief Technology Solutions Officer at Solace, where he helps CIOs, CTOs and Chief Architects on their real-time, event-driven technology transformation journeys. Sumeet is a regular public speaker in technology forums. Follow Sumeet on LinkedIn here.

Table of Contents

Introduction 4

The Event-First Mindset in Retail 6

Step 1: Culture, Awareness, and Intent 8

Step 2: Identify Candidates for Real-Time 9

Step 3: Build Your Eventing Foundation 10

Step 4: Pick a Pilot Application 17

Step 5: Decompose the Event Flow Into Asynchronous, Event-Driven Microservices 19

Step 6: Get a Quick Win 29

Conclusion 30

Introduction

Most retailers struggle to connect, analyze, and make use of the information streaming from a growing number of sources. With data spread across brick-and-mortar stores, ERP systems, data centres, clouds, and new applications constantly being added – it's becoming increasingly difficult to get a consistent and trusted real-time view that can power a rapid, personalized customer response.

Traditional retail IT architectures were not designed to support real-time data scenarios and are being pushed to their limits. The desire to share your customer transaction and behavioral data across different geographic regions, clouds, and edge scenarios, has further created new data management challenges.

The lack of connectivity between these distributed and siloed data locations is largely at fault. It forces your applications to send data messages directly between sender and receiver, often without any buffering in between and sometimes only once a day. These types of synchronous, tightly coupled messaging processes result in slow responses, stale data and in the worst of cases – lost customers.

To become more responsive and to take advantage of new technologies (cloud, IoT, microservices, etc.), your enterprise IT architecture needs to support real-time, event-driven interactions. Whether it's local or global, you can:

- Adjust pricing on products more efficiently based on market costs
- Bring new products to the market faster and increase stock turnover
- Offer differentiated value to your customers by remaining relevant and personalized with compelling in-store and online choices

All of these scenarios are examples of how knowing about events (any change of state) as they happen, can provide business value across an organization. An “event” can broadly be described as a change notification. These changes can have a variety of forms, but all have the common structure of an action that has occurred on an object. Event-driven architecture (EDA) is just a way of building a retail enterprise IT system where related applications or services can produce and consume events in real time.

Contrary to popular belief, implementing the EDA paradigm across the entirety of your retail organization doesn’t require an overhaul of all systems and can start with a few small, simple steps. To get started down this path, you need to have a good understanding of your data, but more importantly, **you need to adopt an event-first mindset.**

Sumeet Puri, Chief Technology Solutions Officer, explains a surefire strategy for how your retail business can move towards embracing event-driven architecture as a future-proof blueprint for real-time IT architecture. With his field-proven six-step process, you will be on your journey towards leveraging the value behind business events across your retail stores, systems, and applications as they occur, and unleashing the true power of real-time retail.

The Event-First Mindset in Retail

Event-driven architecture is not new; GUIs and capital markets trading platforms have always been built this way. The reality is that it's just becoming more mainstream now, especially in retail scenarios where data transfer can be time-sensitive (such as with customer transactions or inventory counts) or rapid-response critical (in-store personalization).

But, to make real-time a true reality, a mindset change is required.

Everything that happens across your distributed retail organization needs to be thought of as a "digital event" and treated as first-class citizens in your IT infrastructure.

With event-driven architecture, applications and microservices talk through events or event adapters. Events are routed among these applications in a publish/subscribe manner – meaning they are published (sent) to those 'subscribers' that see value in knowing that a change in that event has occurred (and most likely will do something with this knowledge).

In other words, rather than relying on batch processing or an enterprise service bus (ESB) orchestrating a flow, business flows are dynamically *choreographed* based on each business logic component's interest and capability. This makes it much easier to add new applications, as they can tap into the event stream without affecting any other system, do their thing, and add value.

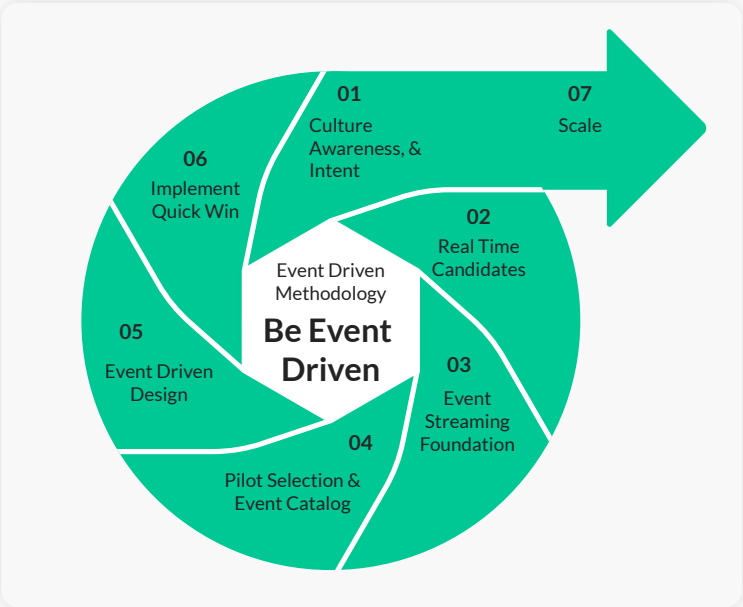
So how do you do that? What should your strategy be?

At a high level, if your retail organization wants to become truly real-time, you need to do these three things:

1. Event-enable your existing systems (inventory, purchasing, etc.)
2. Modernize your platform to support streaming across environments
3. Alert and inform internal and external stakeholders

So, the next question is – how do you put these steps into practice?

The following six steps have been proven to make the journey to EDA faster, smoother, and less risky in many real-world implementations.



solace.

Step 1: Culture, Awareness, and Intent

Are you ready, aware, and have the intent to implement event-driven architecture?

French supermarket chain **Les Mousquetaires built a hybrid integration platform to better enable customer experience.**

Watch how.

I'm sure if you're reading this, then the answer is either "yes!" or "I don't know!" However, most mainstream people in IT were trained in school to think procedurally. Whether you started with Fortran, C, Java, or Node, most IT training and experience has been around synchronous function calls or RPC calls or Web Services to APIs – all synchronous. There are exceptions to the rule – capital markets' front office systems have always been event-driven because they had to be real-time from the start! But often, IT needs a little culture change to fully embrace EDA.

Microservices don't need to be calling each other, creating a distributed monolith. As an architect, if you know which applications/microservices consume and produce which events, you can just choreograph using a publish/subscribe event broker – or a distributed network of brokers (also known as an event mesh) – rather than orchestrate with an ESB.

It's important to ponder a little, read up, educate yourself, and educate stakeholders about the benefits of EDA: agility, responsiveness, and better customer experiences.

Build support, strategize, and ensure that the next project – the next transformation, the next microservice, the next API – will be done the event-driven way.

You will need to think about which use cases can be candidates, look at them through the event-driven lens, and articulate a go-to approach to realize the benefits.

Think real-time. Think event driven.

Step 2: Identify Candidates for Real-Time

Not all systems need change or can be changed to real-time. But the majority will benefit from an event-driven approach.

You've probably already thought about a bunch of projects, APIs, or candidates that would benefit from being real-time.

What are the real-time candidates in *your* retail business?

- The troublesome order management system?
- The next generation payments platform you're building?
- Would it make better sense to push master data (such as price or inventory adjustments) to downstream applications, instead of polling for it?
- Are you driven by the possibility of what a real-time loyalty points service could do for your business if you could offer rewards the moment someone physically walks into your store?

There will be multiple candidates with different priorities and challenges, so look for projects which will:

- Remove pain (i.e., brittleness, performance issues, new functionality)
- Cause a medium to high business impact
- Serve as a quick win that will deliver business value and breathe optimism into your team

By starting with one small project, you can begin to find the quirks unique to your organization that will inevitably be present on a larger scale. Keep in mind that not every data source will offer business value being real-time and event driven. Find a system that has robust data generation capabilities and can be easily modified to submit messages. These messages will be harnessed as your events later in the journey.

Make a shortlist of real-time candidates for your event-driven journey. It's also important to bring in the stakeholders at this early stage because events are often business-driven, so not all your stakeholders will be technical. Consider which teams will be impacted by the transformation, and which specific people will need to be involved and buy-in to make the project a success from all sides.

Step 3: Build Your Eventing Foundation

Once the first project is identified, it's time to think about architecture and tooling.

An event-driven architecture will depend on decomposing flows into microservices and putting in place a runtime fabric that lets the microservices talk to each other in a

publish/subscribe, one-to-many, distributed fashion.

It's also important to start the design-time right and have the tooling to ensure that events can be described and cataloged, and have their relationships visualized.

You'll want to have an architecture that's modular enough to meet all your use cases, and flexible enough to receive data from everywhere that you have data deployed (on-premises, private cloud, public cloud, etc.). This is where the eventing platform comes in with the event mesh runtime.

Having an eventing platform in place (even the most basic pieces of it) allows your first project to leverage it as microservices and events start to come online and communicate with each other, rather than via REST, resulting in a distributed monolith.

The following runtime and design-time pieces are essential:

- Event Broker
- Event Mesh
- Event Portal
- Event Taxonomy

Event Broker

An event broker is the fundamental runtime component for event routing in a publish/subscribe, low latency, and guaranteed delivery manner. Applications and microservices

are decoupled from each other and communicate via the event broker. Events are published to the event broker via topics following a topic hierarchy (or taxonomy), correspondingly subscribed to by one or more applications or microservices, or analytics engines or data lakes.

An ideal event broker uses open protocols and APIs to avoid vendor lock-in. With open standards, you have the flexibility of choosing the appropriate event broker provider over time. Think about the independence TCP/IP gave to customers choosing networking gear – open standards made the internet happen. By leveraging the open-source community, it's easier to create on-the-fly changes, and you're not stuck having to consult closed documentation or sit in a support queue.

Lastly, an ideal event broker is simple. It offers simplicity in deployment, event governance, and scalability, which gives you the freedom to focus on what matters: your events.

Event Mesh

While you might start with a single event broker in a single location, modern applications are often distributed. Whether it's on-premises, in multiple clouds, global warehouses, or retail stores – applications, microservices, and insight capabilities are distributed.

An event originating in a single retail store may have to go to the local store's systems, the centralized ERP, the cloud-based data lake, and to an external partner. As such, event distribution should be transparent to producers and consumers – they should be connecting to their local event broker, just like you or I would connect to our home WiFi router to access all websites, no matter where they are hosted.

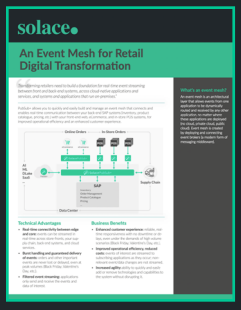
Event-driven architecture is a constantly evolving system, so if you identify event sources on-premises today, you may find that those events live in the cloud tomorrow.

An event mesh is a network of event brokers that dynamically routes events between applications no matter where they are deployed – on-premises or in any cloud or event at IoT edge locations. Just like the internet is made possible by routers converging routing tables, the event mesh converges topic subscriptions with various optimizations for distributed event streaming.

There are multiple ways an event mesh supports your application architecture:

- Connects and choreographs microservices using publish/subscribe, topic filtering, and guaranteed delivery over a distributed network
- Pushes events from on-premises to cloud services and applications
- Enables digital transformations for Internet of Things (IoT)
- Enables Data as a Service (DaaS) across lines of business for insights, analytics, ML, and more
- Gives you a much more reactive and responsive way to aggregate events

Of course, how you deploy your event mesh will help determine the kind of event broker you're looking for.



Event Mesh for Retail Digital Transformation Datasheet

Click to read more.

Event Portal

An event portal – just like an API portal – is your design and runtime view into your event mesh. An event portal gives architects an easy GUI-based tool to define and design events in a governed manner and offers them for use by publishers and subscribers. Once you have defined events, you can design microservices or applications in the event portal and choose which events they will consume and produce by browsing and searching the event catalog.

As your events get defined, they are enlisted in an event catalog for discovery. An event catalog gives you visibility into your events. Although it's more of a documentation step, this will help to visualize and describe the events that you're able to process. When building out the system for more event sources and different ways to consume them, the catalog is a great reference to know what's already been built so you can avoid duplication of effort.

Event Taxonomy

Topic routing is the lifeblood of an event-driven architecture. Topics are metadata of events; tags of the form a/b/c – just like an HTTP URL or a File Path – which describe the event. The event broker understands topics and topic hierarchy, and can route events based on who subscribed to them, including wildcard subscriptions.

As you start your event-driven architecture journey, it's important to pay attention to topic taxonomy – setting up a topic naming convention early on and governing it. A solid taxonomy is probably the most important design investment you will make, so it's best not to take shortcuts here. Having a good taxonomy will significantly help with event routing, and the conventions will be obvious to application developers.

Udders Ice Cream Example: Event Taxonomy

Let's look at an order example for Udders Ice Cream. An order for the flavor rum and raisin has come in from the Lazada ecommerce website in Singapore:

Event	Topic
New Order	order/init/1.1/icecream/udders/rumraisin/sg/lazada
Validated Order	order/valid/1.1/icecream/udders/rumraisin/sg/lazada
Order Shipment	order/shipped/1.1/icecream/udders/rumraisin/sg/lazada

Because events from the publisher (microservice, application, legacy-to-event adapter) have topics as metadata, consumers can use it to subscribe to events.

Event	Publisher	Subscriber
New Order	Lazada ecommerce store site	Order validation microservice
Validated Order	Order Validation Microservice	Order processor for ice creams in Singapore
Order Shipment	Any upstream	Data lake or AI/ML ingestor

Topic subscriptions would be:

- Consume and validate all orders of version 1.1 and publish the order valid message: **order/init/1.1/>**
- Consume and process all valid orders for ice cream originating in Singapore: **order/valid/*/icecream/*/*/sg/***
- All orders no matter what stage, category, location: **order/>**

Organizational Alignment

Event-driven architecture starts small and grows, but over time, the organization must evolve to event-first thinking. This requires some thought leadership to get buy-in. The ESB team needs to start thinking about choreography rather than orchestration. The API teams need to start thinking about event-driven APIs rather than just request/reply.

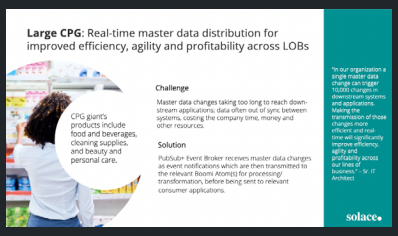
Step 4: Pick a Pilot Application

The next step for implementing your event-driven architecture is to determine which event flow to get started with first.

Essentially, an event flow is a business process that's translated into a technical process. The event flow is the way an event is generated, sent through your broker, and eventually consumed.

By getting started with a pilot project, the event catalog will start to take shape automatically as you identify events and the event flow. Whether you maintain the catalog in a simple spreadsheet or in an event portal, the initial event catalog also serves as a starting point of reusable events – which applications in the future will be able to consume off the event mesh.

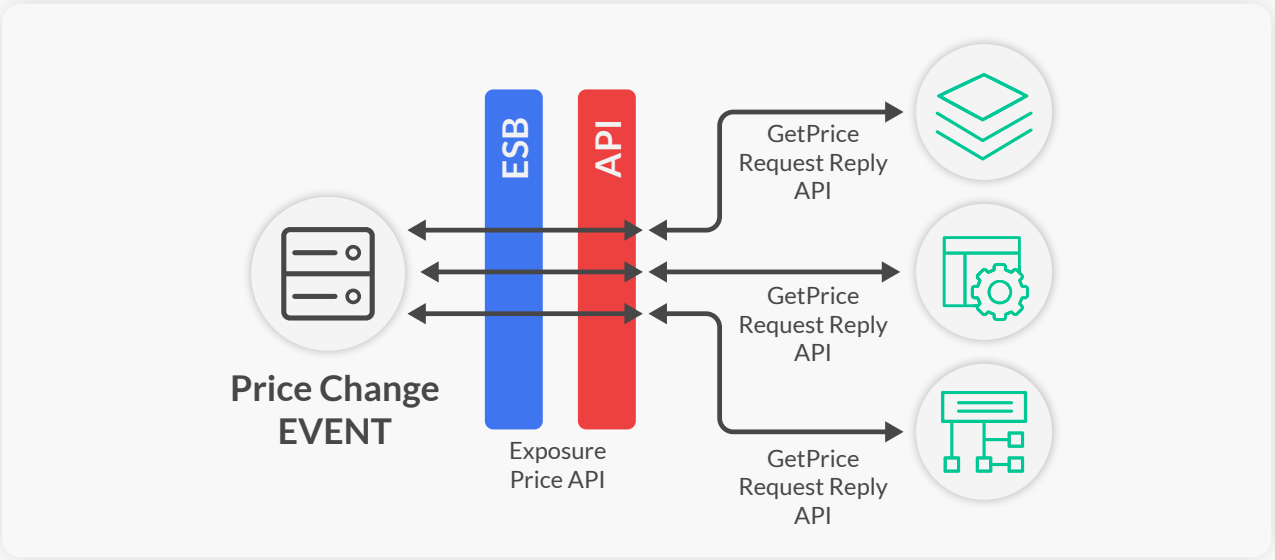
You want to choose a flow that makes the most sense for your current state modernization or pain reduction. An inflight project or an upcoming transformation make ideal candidates, whether it's for innovation or technical debt reduction via performance, robustness, or cloud adoption. The goal is for it to become a reference for future implementations. Pick a flow that can be your quick win.



Learn more about how a large CPG company is using Solace for real-time data distribution.

Read the case study.

Consider this product price change flow as an example:



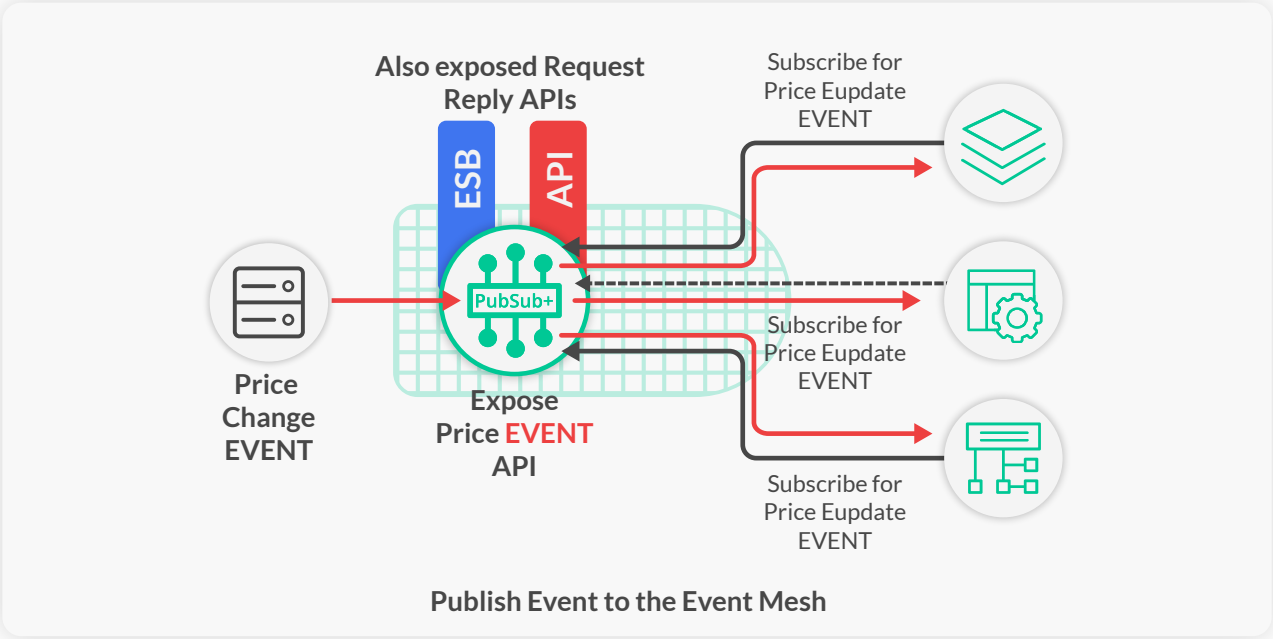
The price change itself is considered the event – 'price' being the noun, and 'change' being the verb.

In an existing architecture that is *not* event-enabled, the price change event is:

- **Not pushed** – a Request/Reply API is required, or probably even a batch process
- **Not real-time** – downstream applications don't know about the price change until they *ask*
- **Not cost effective to scale**
- **Bursty** – event applications continuously call the API, which can cause load/burst

By event-enabling the price change and implementing event-driven architecture with an event mesh, downstream apps *subscribe* to the price change events and receive event notifications as they happen in real-time. The event mesh filters and only *pushes* the events that the downstream applications have subscribed to (in accordance with the topic taxonomy).

Events are queued and throttled to reduce load, making it easier to scale. They are also delivered in a lossless, guaranteed manner.



Step 5: Decompose the Event Flow Into Asynchronous, Event-Driven Microservices

Once pilot flows have been identified and an initial event catalog is starting, the next step is to start an event-driven design by decomposing the business flow into event-driven microservices and identify events in the process.

Decomposing an event flow into microservices reduces the total effort required to ingest event sources, as each microservice will handle a single aspect of the total event flow. New business logic can be built using microservices, while existing applications – SAP, mainframe, custom apps, etc. – can be event-enabled with adapters.

Microservices Orchestration vs. Choreography

There are two ways to manage your microservices in your event flows: orchestration and choreography.

- With orchestration, your microservices work in a call-and-response (request/reply) fashion, and they're tightly coupled, i.e., highly dependent on each other, tightly wired into each other.
- With event routing choreography, microservices are reactive (responding to events as they happen) and loosely coupled — which means that if one application fails, business services not dependent on it can keep on working while the issue is resolved.

In this step, you'll need to identify which steps must occur synchronously and which ones can be asynchronous. Synchronous steps are the ones that need to happen when the application or API invoking the flow is waiting for a response, or blocking.

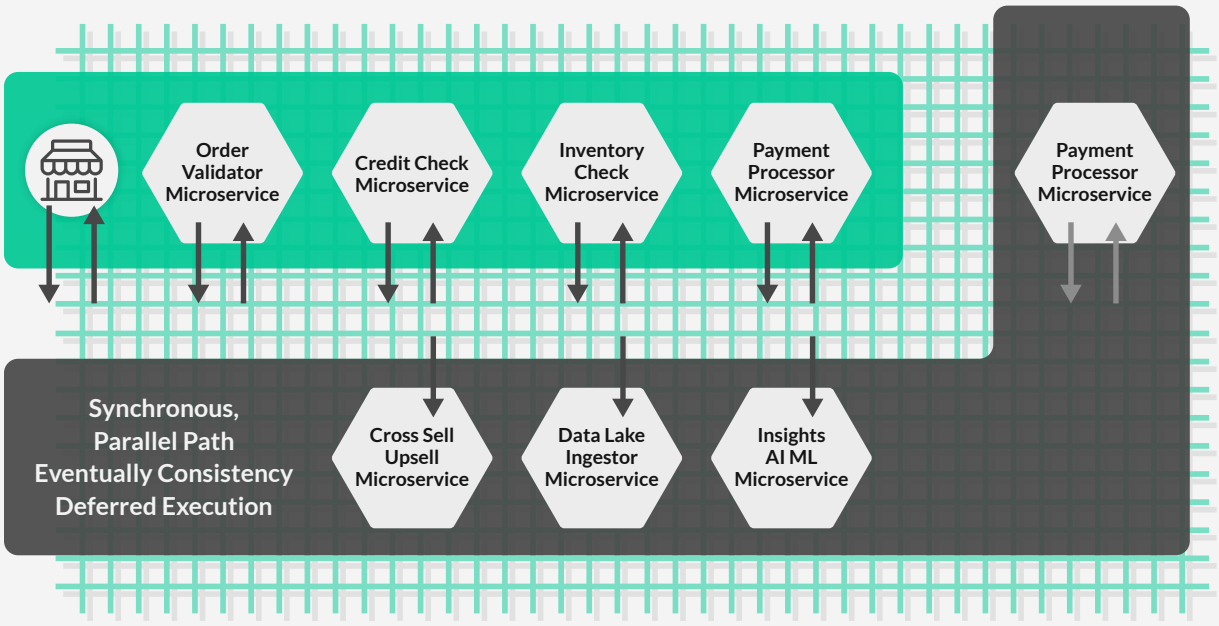
Asynchronous events can happen after the fact and often in parallel – such as logging, audit, or writing to a data lake. In other words, applications that are fine with being "eventually consistent" can be dealt with asynchronously. The events float around, and microservices choreography determines how they get processed. Because of these key distinctions, you should keep the synchronous parts of the flow separate from the asynchronous parts.

This modeling of event-driven processes can be done manually at first or with an event portal, where you can visualize and choreograph the microservices.

Udders Ice Cream Example: Eventually Consistent

One of the problems with a RESTful synchronous API-based system is that each step of the business flow – each microservice – is inline. When the Udders Ice Cream POS system submits an order, should it wait for all the order processing steps to be completed, or should only a few mandatory steps be inline?

If you think about it, all the "insights" processes do not need to happen before the point of sale systems get an order confirmation – they are just going to slow down the overall response time. In reality, only the order ingestion and validation need to be inline – everything else can be done asynchronously.



The event mesh provides guaranteed delivery, so the non-inline microservices can get the data a little after the fact, but in a throttled, guaranteed manner. The event stream flows into the systems in a parallel manner, thereby further improving performance and latency.

Parasitic Listeners

As events are flowing and are cataloged, they can also be consumed by analytics, audits, compliance, and data lakes. Authorized applications will receive a wildcard-based, filtered stream of events in a distributed manner – with no change to the existing microservice, and no ESB or ETL changes.

Event Sources and Sinks – Dealing with Legacy Applications

There are a couple of event sources and sinks that you should be aware of. Some sources and sinks are already event-driven, and there's little processing you'll need to do to consume these events. Let's call the others "API-ready." API-ready means they have an API that can generate messages that can be ingested by your event broker, usually through another microservice or application. Although there's some development effort required, you can get these sources to work with your event broker easily. Then there are legacy applications that aren't primed in any way to send or receive events, making it difficult to incorporate them into event-driven applications and processes.

Click to Tweet!

Event-driven architecture starts small and grows, but over time, the organization must evolve to event-first thinking.

Here's a breakdown of the three kinds of event sources and sinks:

	Event-Driven	API-Ready	Legacy
API Status?	Natively event-driven	Have a REST/SOAP API, with schemas	No standards-based API, though there might be various ways to connect
Push/Pull Abilities?	Can push events, can consume events	Request/reply, no notion of topics or push	Data needs to be pulled, or polled, but may also be triggered
Adapters for Streaming?	None needed	Need microservices or streaming pipeline to transform request-reply source into a streaming destination, and publish intelligent topic	Need an integration adapter (JDBC, MQ, JCA, ASAPIO, Striim, legacy ESBs) installed close to the source of the destination of the event to transform and pub/sub events

Event-Driven Applications

Event-driven applications can publish and subscribe to events and are topic and taxonomy aware. They are real-time, responsive, and leverage the event mesh for event routing, eventual consistency, and deferred execution.

API-Ready Applications

While API-ready applications may not be able to publish events to the relevant topic, they can publish events that can be consumed via an API. In this case, an adapter microservice can be used to subscribe to the "raw" API (an event mesh can convert

REST APIs to topics), inspect the payload, and publish the message to appropriate topics derived from the contents of the payload. This approach works better than content-based routing, as content-based routing requires the governance of payloads in very strict manners down to semantics, which is not always practical.

Legacy Applications

You'll probably have quite a few legacy systems that are not even API-enabled. As most business processes consume or contribute data to these legacy systems, you'll need to determine how to integrate them with your event mesh. Are you going to poll the system? Or invoke them via adapters when there is a request for data or updates? Or are you going to off-ramp events and cache them externally? In any case, you'll need to figure out how to event-enable legacy systems that don't even have an API to talk to.

The key to accommodating legacy systems is identifying them early and getting to work on them as quickly as possible.

Go Cloud-Native As You Can

The accessibility and scalability of the cloud reflects that of an event mesh. They're perfect candidates to deploy together. That said, you can't expect to have all of your events generated or processed in the cloud. To ignore on-premises systems would be a glaring oversight. With an event mesh implemented, it doesn't matter where your data is, so you can more easily take a hybrid cloud approach.

For example, your data lake might be in Azure while your AI and ML capabilities might be in GCP. You might have manufacturing and logistics in China or Korea and a market in India or the USA. With 5G about to unleash the next wave of global connectivity, your event-driven backbone needs to work hand-in-hand with two patterns, 180 degrees apart: hybrid/multi-cloud and an intelligent edge.

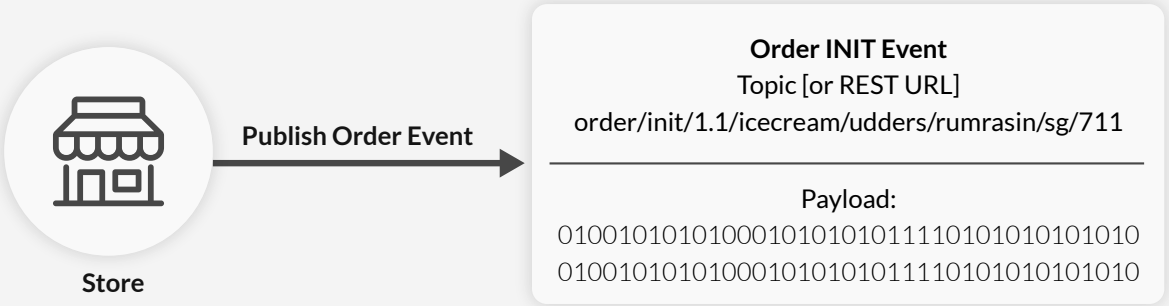
By using standardized protocols, taxonomy, and an event mesh, you are free to run business logic wherever you want.

Udders Ice Cream Example: Order Management

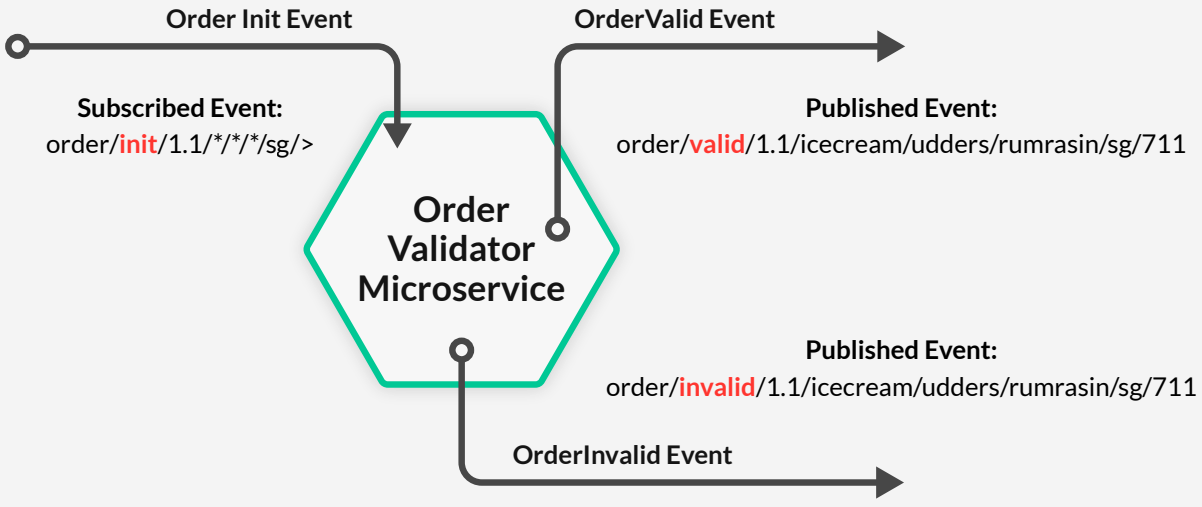
An order management process for Udders Ice Cream may see these microservices in an event flow following a point of sale:

- Order validator
- Credit check
- Inventory check
- Payment processor
- Order processor

An order for rum raisin ice cream is initiated when a point of sale system makes an API call to submit it:



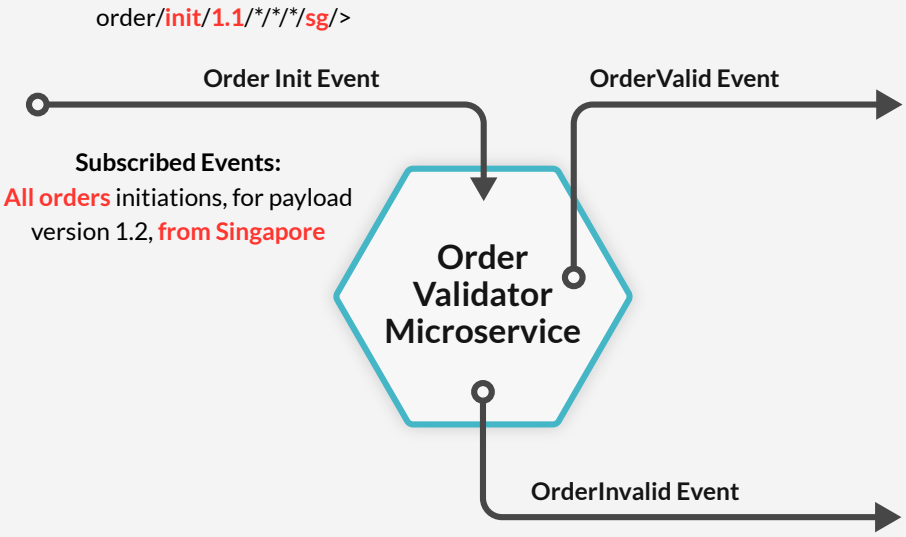
The Order Validator microservice consumes the new order event, and produces the valid order, or invalid order event. The valid order event is then consumed by the Credit Check microservice, which similarly produces the next events.



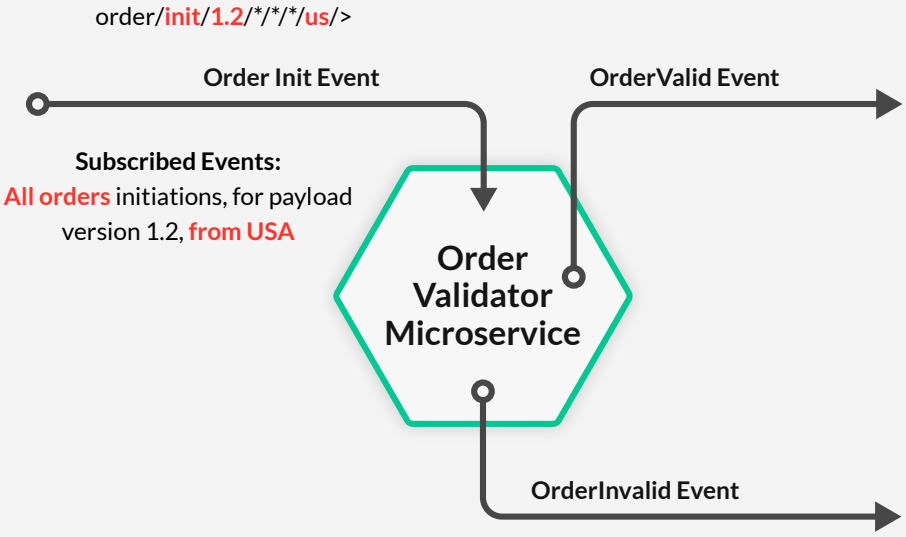
- **Subscribed Events:** all order initiations, irrespective of version and product
- **Published Events:** order validation status with other meta data

Udders Ice Cream Example: Event Routing

Issue: Order validation rules have been changed from Singapore to the US and payload has changed from JSON to protobufs.

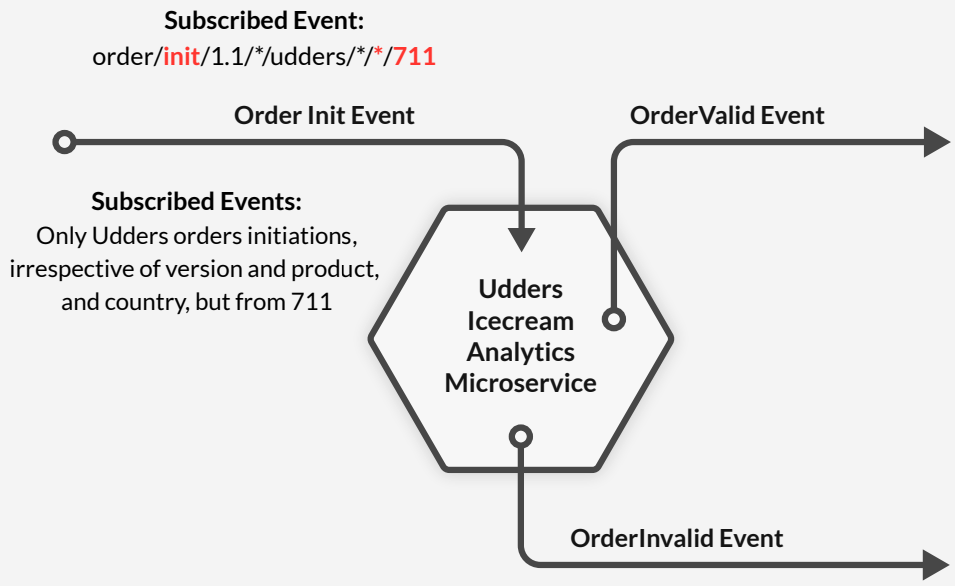


Solution: A new (reused) microservice is created to serve the business logic of validating orders from the US with the new payload and rewire topic subscriptions. The new microservices start listening to the relevant topic, consuming order initiation events from US of the version 1.2. Nothing else changed!



Issue: Want to monitor all orders coming from the 711 chain.

Solution: Implement a microservice with the relevant Insights business logic, look up the event in the event catalog, and start subscribing to events using wildcards!



solace

Step 6: Get a Quick Win

Once the design is done and the first application gets delivered as an event-native application, the event catalog also starts to get populated.

Getting a quick win with the event catalog as a main deliverable is just as important as the other business logic, and it drives innovation and reuse.

Hopefully, with the ability to do more things in real-time, you can demonstrate agility and responsiveness of applications, which in turn leads to a better customer experience. With stakeholder engagement, you can truly transform to real-time retail!

Bonus Step: Rinse & Repeat

Now that you have your template for becoming an event-driven enterprise and have your first quick win in place, you can rinse and repeat. Go ahead and choose your next event flow!

From here, you can walk through the steps again for the next few projects. As you go, the event catalog in your event portal will be populated with more and more events. Existing events will start to get reused by new consumers and producers. A richer catalog will open up more opportunities for real-time processing and insights.

Culture change is constant but gets easier with showcasing success. The integration/middleware team might own the event catalog and event mesh, while the application and line of business (LOB) teams use it/contribute to it. LOB-specific event catalogs and localized event brokers are also desirable patterns, depending on how federated or centralized the organization's technology teams and processes are.

As you scale, more applications produce and consume events, often starting with reusing existing events. That is how the event-driven journey snowballs as it scales!

Conclusion

Keeping up to constantly changing demands in the retail space requires one thing: digital transformation. The world is not slowing down, so your best bet is to identify ways you can – cost effectively and efficiently – upgrade your enterprise architecture to keep up with the times. But it's not an easy task.

Architects and developers need a platform and a set of tools to help them work together to achieve the real-time, event-driven goals for their organizations.

With these six steps, you can make the leap with the correct strategy and tools in hand that will support your real-time, event-driven journey in retail. Solace PubSub+ Platform helps enterprises design, deploy, and manage event-driven architectures and can be deployed in every cloud and platform as a service. Solace is the only stable and performant solution that fits the unique needs of architects looking to implement event-driven architecture within their organizations.

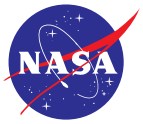
Ottawa | Toronto | New York | Chicago | Atlanta | Silicon Valley | London | Paris | Zurich
Tokyo | Seoul | Hong Kong | Shanghai | Singapore | Mumbai | New Delhi | Melbourne | Sydney

About Solace

Solace helps large enterprises become modern and real-time by giving them everything they need to make their business operations and customer interactions event driven. With PubSub+, the market's first and only event management platform, the company provides a comprehensive way to create, document, discover and stream events from where they are produced to where they need to be consumed – securely, reliably, quickly, and guaranteed. Learn more at solace.com.

Follow us on

A Few of Our Customers



Our Featured Partners

VMware Tanzu