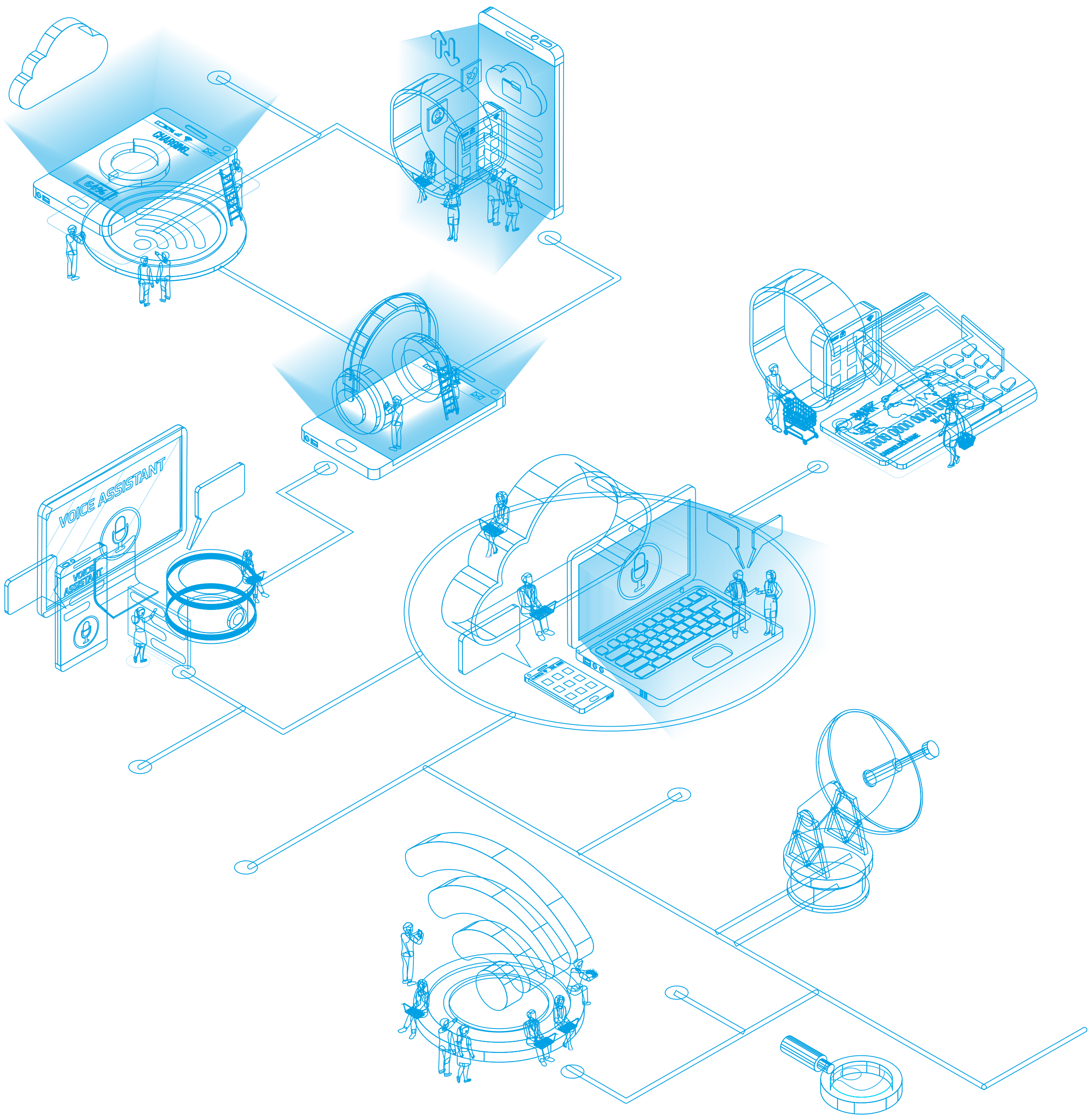
CODIFICATION

Codification's guide to Event-Driven Architecture



October 2022

Field Guide

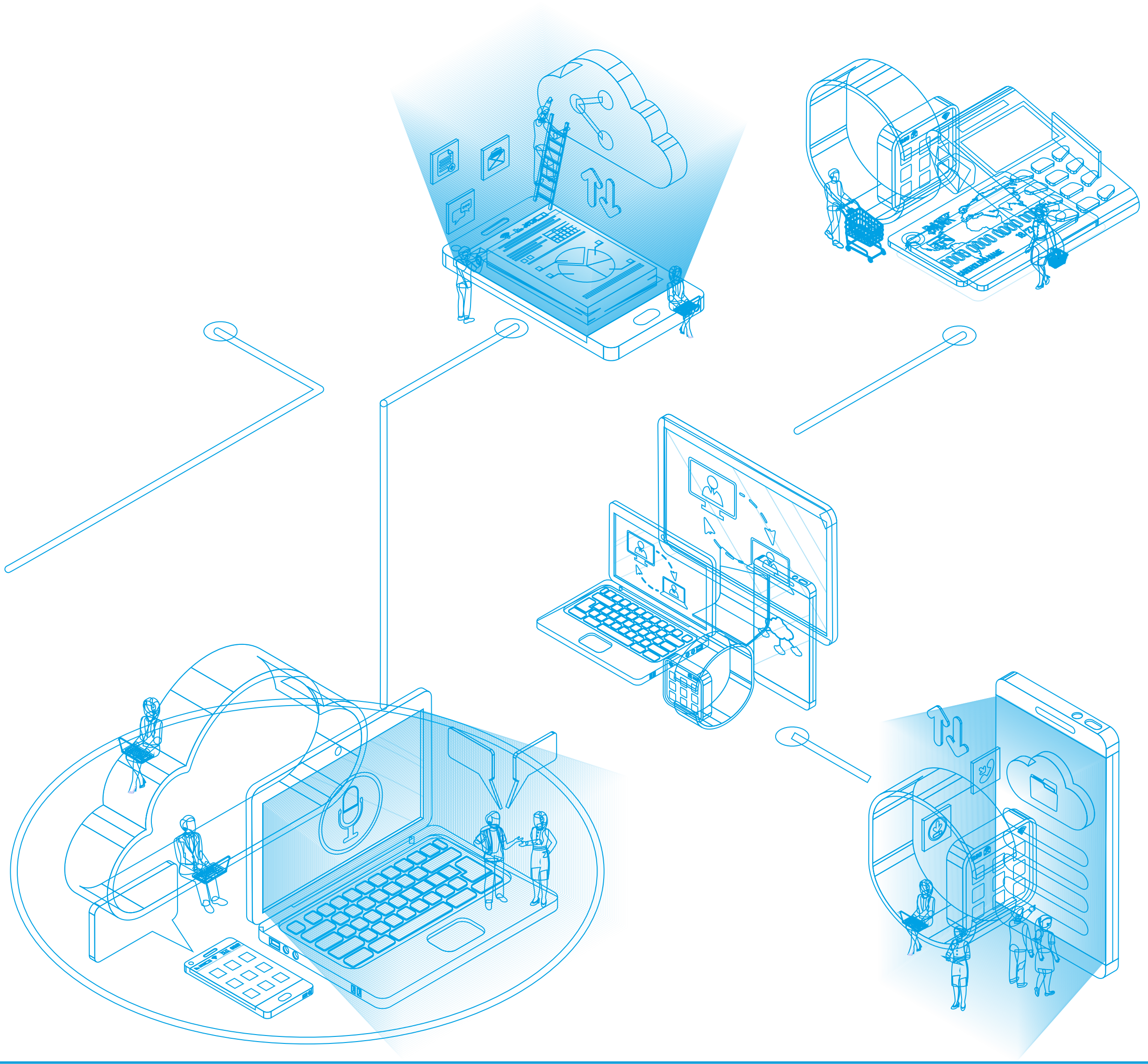
Table of Contents

Introduction 1
What is Event-driven Architecture? 2
How Event-driven Architecture is Developed 3
What is an Event? 5
How Event-driven Architecture Works 7
Event-driven Applications 8
Benefits of Using Event-driven Architecture 9
Further outcomes 10
Event-driven Architecture Models 11
When to use Event-driven Architecture? 12
Event-driven Architecture & Microservices 13
Is Event-driven Architecture for You? 13
Summary 14

Introduction

Modern applications created using microservices often use an event-driven architecture, which uses events to initiate and facilitate communication between disconnected services. An item added to a shopping basket on an e-commerce website is an example of an event; which is a change in state or an update. Events can either be identifiers or convey the state (the item purchased, its price, and a delivery address) (a notification that an order was shipped).

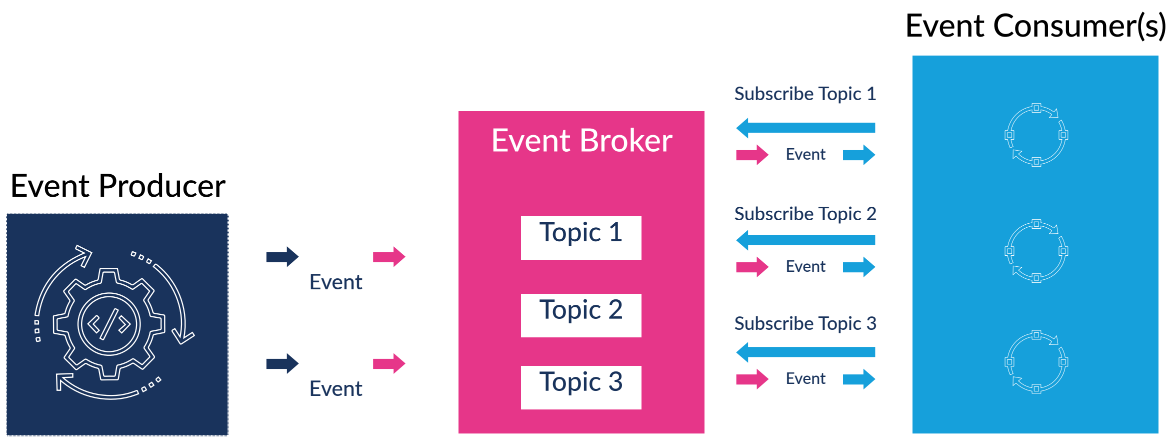
Event-driven architectures are made up of three main parts: event producers, event routers, and event consumers. An event is published by a producer and sent to the router, which filters and sends it to consumers. Due to the decoupling between producer and consumer services, each can be grown, modified, and deployed separately.



What is Event-driven Architecture?

An organisation can identify "events" or crucial business moments (such as a transaction, site visit, shopping cart abandonment, etc.) using event-driven architecture (EDA), a software design pattern, and respond to them in real-time or almost real-time.

The previous "request/response" architecture, in which services had to wait for a response before moving on to the next task, is replaced with this pattern. Events control the flow of event-driven architecture, which is built to react to them or take action in response to an event.



Asynchronous communication is a common term used to describe event-driven architecture. As a result, neither the sender nor the recipient must wait for the other to complete their current task. That particular message is not what drives the systems. For example, a phone call is thought of as synchronous or more like the traditional "request/response" design. You receive a call from someone asking you to execute a task. You wait while the other person finishes, and then both sides hang up. Text messaging is an example of an asynchronous event. When you send a message, you might not even be aware of the recipient or whether anyone is listening, but you are not anticipating a response.

How Event-Driven Architecture is Developed

In recent years, there has been a shift away from service-oriented architecture’s emphasis on data at rest and toward an emphasis on events (event-driven architecture). Moving away from data aggregation and data lakes, we are concentrating on data in flight and tracking it as it travels from one location to another. The majority of systems run on what is called the “data-centric model,” in which the data serves as the ultimate authority.

When using an event-driven architecture, a model that is based on events must replace a model that is based on data. In the event-driven paradigm, the events are more important than the data, which still has value.

The prevention of data loss was given top emphasis in the service-oriented approach, however. The main concern with event-driven architecture is making sure you react to events as they happen. Events have a law of decreasing returns, which means that as they age, their value decreases. Today, however, event-driven architecture and service-oriented architecture are frequently used in tandem.

An analogue of a log is frequently used in event-driven architecture to keep track of things. Analysts refer to events as unchangeable things that have happened. You can also replay the log to learn more about what transpired in the past. In contrast, the data-centric paradigm places a greater emphasis on the data's current state. And lastly, analysts often compare data-centric and event-centric systems to a company's nervous system, which sends messages to all parts of the company, to show how different the two are.

When you use an event-driven architecture, you have event producers that create and send out event notifications, as well as one or more consumers that, when they receive an event, start processing the event. Consider the scenario where a new movie from Netflix has just been uploaded. Multiple programmes could be watching or listening for that notice, which would then cause their systems to tell their users about what happened. Applications continue to run, and even if they are listening for this event, they don't stop doing anything while they wait for it. This is different from normal request-response messaging in that applications don't stop doing anything while they wait. When the message is published, they can also reply. As a result, numerous services may operate concurrently.

What is an Event?

Any noteworthy occurrence or state change for system software or hardware is referred to as an event. A message or notice provided by the system to inform another component of the system that an event has occurred is not the same thing as an event.

An event might have internal or external inputs as its source. Events can originate from a user, such as a keyboard or a mouse click, an external source, like a sensor output, or the system, such as the loading of a programme.

To illustrate further, someone might buy something, someone else might check in for a flight, or a bus might be running late. And no matter what industry one is in, events are everywhere and happen all the time. They are present in every business. An event is something that generates a message through production, publication, detection, or consumption. Since the message is the travelling notification that conveys the occurrence, the message and the event are distinct from one another. **An event in an event-driven architecture is likely to cause one or more actions or processes to run in response to it.** An instance of an event could be:

- Requesting a password reset
- Deliveries were made of a package to its intended location.
- Unauthorised login attempt being denied

Each of these events is likely to result in one or more subsequent reactions or processes. Simply logging the event for later review could be one option. Others could be:

- The customer receives an email with instructions on how to change the password
- The ticket for the delivery is closed
- The compromised account is locked and informed to security personnel.

When an event notification is given, the system records that something happened, such as a change in status, and then waits to respond to whoever needs it, whenever they request it. This is known as event-driven architecture. The programme that got the message has two options: it can reply right away, or it can hold off until the desired state change has taken place.

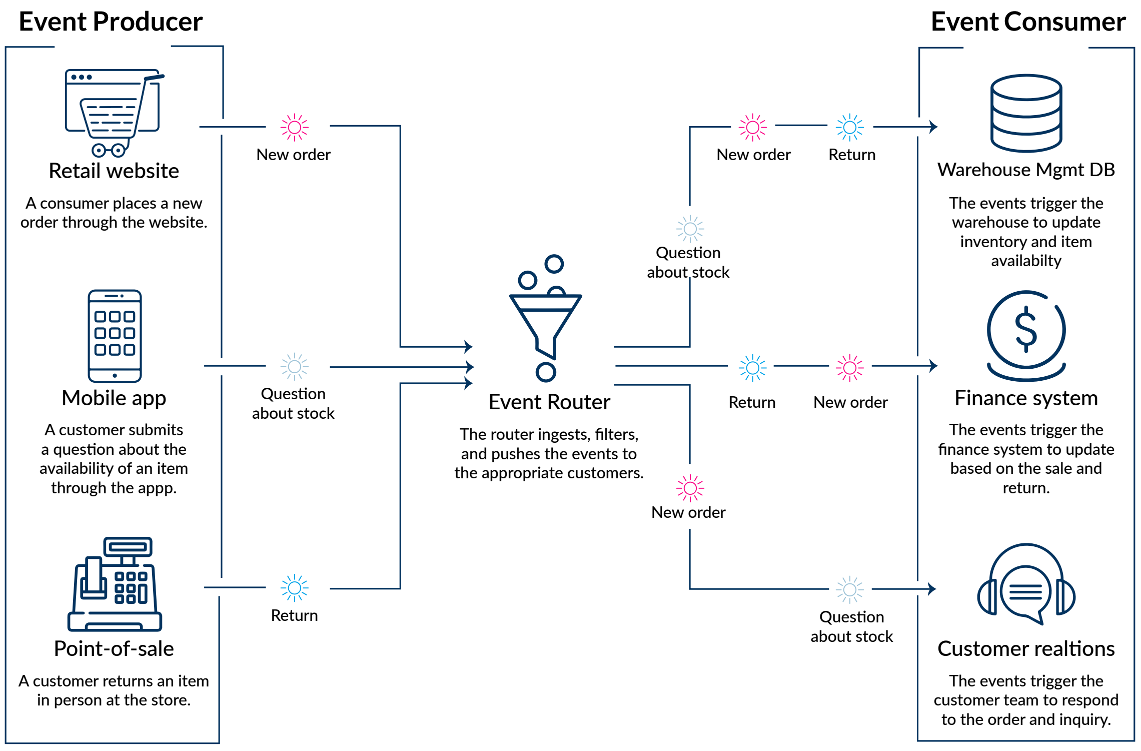
Digital business applications can be made more flexible, scalable, contextual, and responsive by using applications built on an event-driven architecture.

How Event-driven Architecture Works

Three aspects can make up an event-driven architecture: a producer, consumer, and broker. When you have a single producer and a single consumer who are in direct contact with one another and the producer just provides the events to the consumer, the broker may not be necessary. As an illustration, consider a producer who simply sends to a database or data warehouse, where the events are gathered and kept for analysis. In most businesses, several sources will advertise a variety of events, with one or more customers showing interest in some or all of them.

Let’s examine a case in point. If you are a retailer, you may be compiling all of the transactions that take place at all of your locations throughout the globe. You transmit them to a credit card processor or whichever subsequent actions are required, or you feed them into your event-driven architecture, which is keeping an eye out for fraud. A manufacturer can monitor events in real-time and take actions, such as predicting failures or planning maintenance, based on the data that is coming off their equipment, which tells them facts like temperature and pressure.

Below is an example of an e-commerce website using an event-driven architecture. During high demand, this architecture allows the site to respond to changes from multiple sources without crashing or overprovisioning resources.


Event Producer
Retail website
A consumer places a new order through the website.
Mobile app
A customer submits a question about the availability of an item through the appp.
Point-of-sale
A customer returns an item in person at the store.
New order
Question about stock
Return
Event Router
The router ingests, filters, and pushes the events to the appropriate customers.
New order
Return
Question about stock
Return
New order
New order
Question about stock
Event Consumer
Warehouse Mgmt DB
The events trigger the warehouse to update inventory and item availabilty
Finance system
The events trigger the finance system to update based on the sale and return.
Customer realtions
The events trigger the customer team to respond to the order and inquiry.

Event-Driven Applications

Applications can react to data as it is generated using an event-driven architecture. **The growth of data sources that produce events (like IoT sensors) and the creation and acceptance of event-streaming technologies like Hazelcast Jet and Apache Kafka® have contributed to the event-driven approach's recent popularity.** Instead of focusing on a few indicators in a weekly or quarterly report, businesses can use the event-driven approach to view operations and data as ongoing events.

Let's take the fictional characters Jason and Natalie, who have lived in several places, as an example. If we used a traditional batch-based method with data updates to look up their address, we would see their address as it is right now.

We are still able to ask, "What is Jason and Natalie's address right now?" using an event-driven architecture.

Additionally, we could inquire as to where the address Jason and Natalie lived in 2014 was.

Alternatively, "What was Natalie's previous address before she shared one with Jason?"

Event-driven applications are often used for the Internet of Things (IoT), detecting fraud, processing payments, monitoring websites, and real-time marketing, among other things. Data is frequently treated as immutable, or unchangeable, in event-driven applications, which makes it simple to search for the values of data from earlier times. As a result, anytime information "changes," a new data point with a new time period is formed.

Not every event requires the application to take action. Think about the situation of IoT sensor data. There may be millions of non-anomalous events in an application that scans sensor data for anomalies, but none of these events ever cause the application to take any action.

Benefits of Using Event-Driven Architecture

With the aid of event-driven architecture, organisations can achieve a flexible system that can react to changes and make decisions in real-time. With real-time situational awareness, all of the data that is now available and reflects the state of your systems may be used to inform business choices, whether manual or automated.

Event producers and event consumers can share status and response information in real time because events are captured as they happen from event sources, including Internet of Things (IoT) devices, applications, and networks.

Organisations can increase the scalability, reactivity, and access to the data and context required for better business choices by incorporating event-driven architecture into their systems and applications.

Further outcomes:

1. Ability to scale independently and fail

Your services are isolated from one another and are only aware of the event router as a result. As a result, even if one of your services fails, the others will continue to function. This means that your services are interoperable. An elastic buffer that can adapt to increases in workload is what the event router does.

2. Agile development

The event router will automatically filter and push events to consumers; you no longer need to create custom codes to poll, filter, and route events. Additionally, the router eliminates the need for intensive coordination between producer and consumer services, accelerating your development cycle.

3. Easy auditability

An event router serves as a central hub for defining policies and conducting application audits. These policies can limit who can publish to and subscribe to a router as well as manage who and what resources are allowed access to your data. Additionally, you can encrypt events both in transit and at rest.

4. Saving costs

With push-based, event-driven architectures, everything happens as soon as the event shows up in the router. In this manner, you avoid paying for ongoing polling to look for an event. This results in fewer SSL/TLS handshakes, less CPU use, less idle fleet capacity, and less network bandwidth consumption.

Event-Driven Architecture Models

Either a pub/sub model or an event stream model can serve as the foundation for an event-driven architecture.

Pub/sub model

This messaging system is built around event stream subscriptions. With this strategy, subscribers who need to be informed are notified after an event occurs or is published.

Event streaming model

Events are logged using an event streaming approach. An event stream is not subscribed to by event consumers. Instead, they are free to join the stream at any time and read from any point within it.

There are several forms of event streaming, including:

- Event stream processing ingests events and processes or transforms the event stream using a data streaming platform, such as Apache Kafka. Meaningful patterns in event streams can be found via event stream processing.
- Simple event processing is when an event prompts the event consumer to respond right away.
- Complex event processing requires a consumer of events to examine several events to find patterns.

When to use Event-driven Architecture?

1. Replication of data across accounts and regions

Using an event-driven design, systems can be coordinated among teams working in and deploying across multiple locations and accounts. You can independently create, scale, and deploy services by using an event router to transport data between systems.

1. Alerting and monitoring of resource state

You can use an event-driven architecture to monitor and receive notifications on any abnormalities, changes, and updates rather than constantly checking on your resources. These resources may comprise computing nodes, serverless functions, database tables, storage buckets, and more.

1. Parallel processing and fanoutt

With an event-driven architecture, you can spread an event across multiple systems in response to an event without having to write code for each consumer to receive it. The systems will get the event from the router, which they can then handle in parallel for various goals.

1. Bringing together diverse systems

An event-driven architecture can be used to transmit information between systems that are running on different stacks without coupling. Because of the event router's establishment of indirection and interoperability, the systems can communicate messages and data while remaining agnostic.

Event-driven Architecture & Microservices

Applications using microservices are particularly favoured by event-driven architectures. Microservices are designed to carry out specific tasks, frequently ones that depend on the happening of an event. As a result, the foundation of microservices is frequently formed by event-driven architectures.

Is Event-driven Architecture for you?

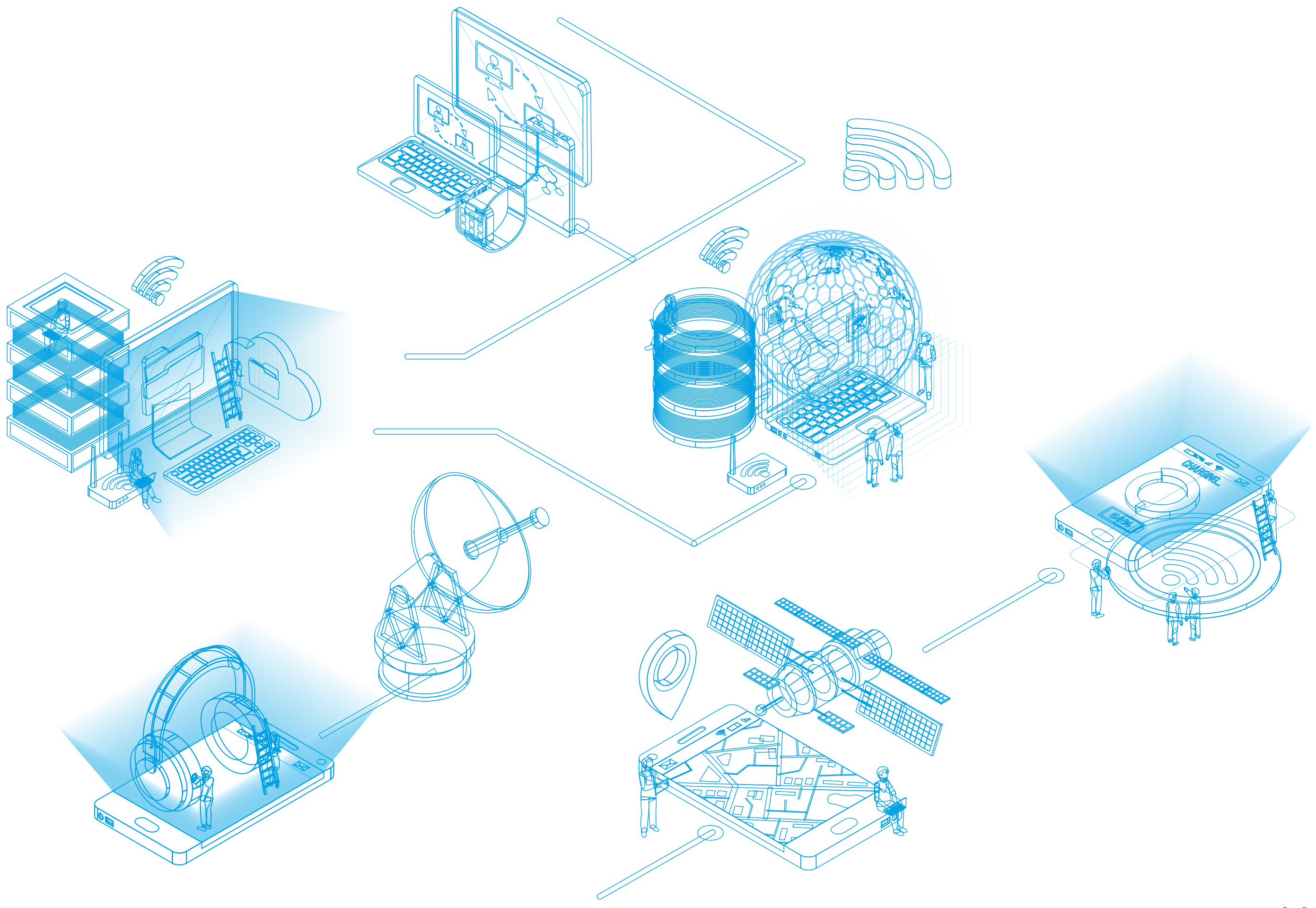
The best architectures for increasing agility and speed are event-driven ones. They are often used in modern programmes that use microservices or any other application with separate parts. You might need to reconsider how you see your application design when implementing an event-driven architecture. Consider the following to position yourself for success:

- Your event source's resilience. If you need to process every single event, your event source needs to be dependable and offer delivery guarantees.
- Your needs for performance control. You should be able to manage the asynchronous nature of event routers in your application.
- You're tracking the event flow. An event-driven architecture's indirection allows dynamic tracking through monitoring services but not static tracking through code analysis.
- Your event source's data. Your event source should be deduplicated and sorted if you need to rebuild the state.

Summary

Let's summarise the several essential event-driven architecture tenets:

- Make sure the right "things" receive the right events using event brokers (creating an event mesh).
- Use topics to ensure that you send and receive only the information you require (event filtering).
- Use deferred execution with event broker persistence to let consumers process events when they're ready.
- Keep in mind that this indicates that not everything is current (eventual consistency).
- To further distinguish the many components of a service, use topics (command query responsibility segregation).



Get in touch with Codification

Visit our website to learn more: www.codification.io/services

About Codification

Codification is a Cloud Native transformation consultancy, with a team of over 100 engineers, consultants and business professionals distributed across the world. We were founded in 2019 in the United Kingdom. We have grown since then to have a presence in Europe, the Middle East and Asia, serving leading multinational corporations, government institutions, global banks, and industry giants with our consultancy and expertise.

Through our experience, we have noticed that visionary leaders want to transform their organisations into technology companies to thrive in the new digital-first economy. Here, businesses want to release software faster, improve quality and build a continuous improvement culture where the best ideas win. At Codification, we establish the direction of a company's technological transformation journey and help implement new technologies and processes, resulting in a modernised digital-ready organisation.

CODIFICATION

Codification United Kingdom
The Core
Bath Lane
Newcastle upon Tyne
NE4 5TF

Phone: +44 01670 223994

Web: www.codification.io/