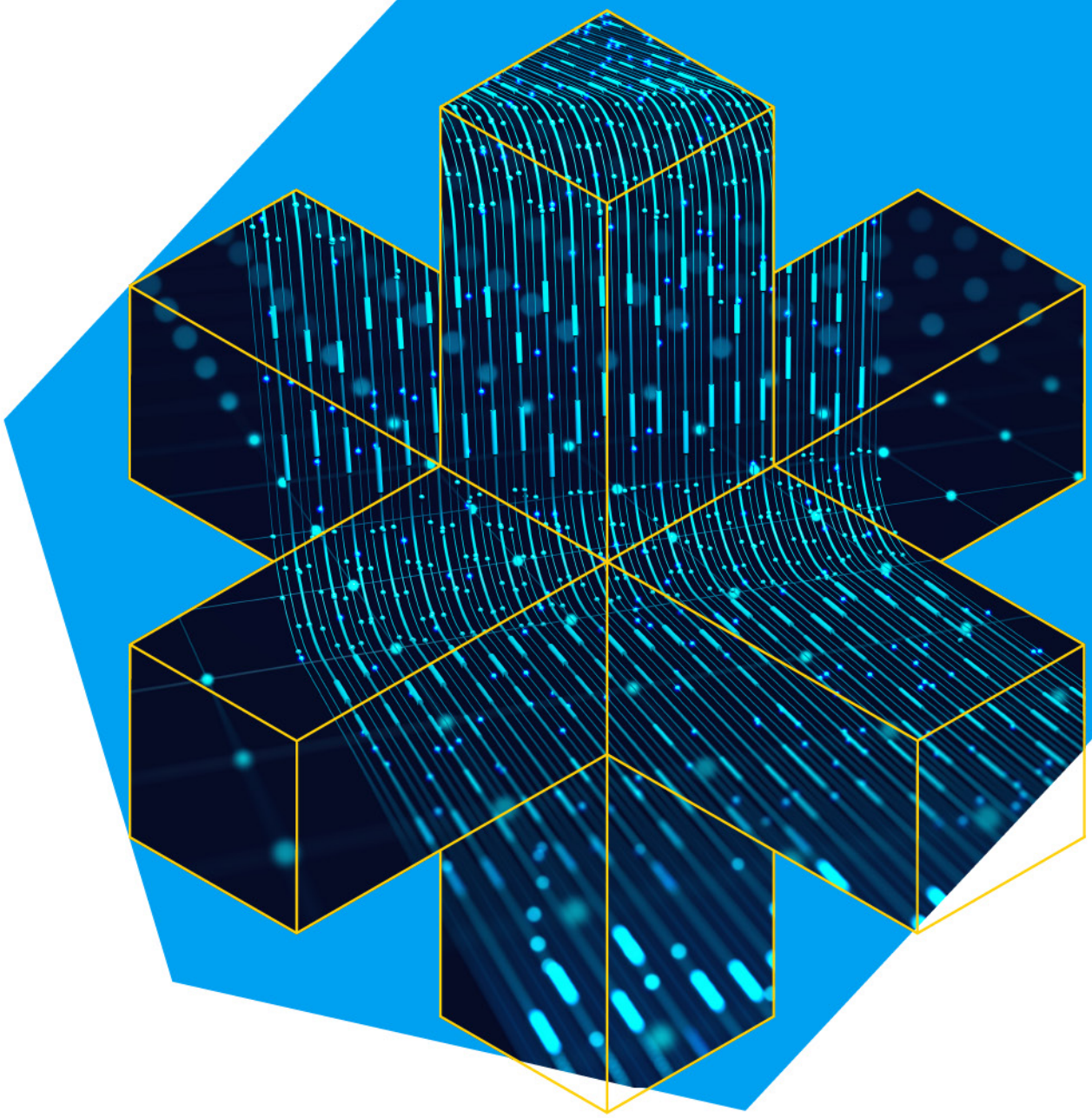
commercetools
Next generation commerce



A Simple Guide to Modern Commerce Terminology

All you need to know about platforms, frameworks, APIs and more

commercetools.com

Table of contents

A Simple Guide to Modern Commerce Terminology

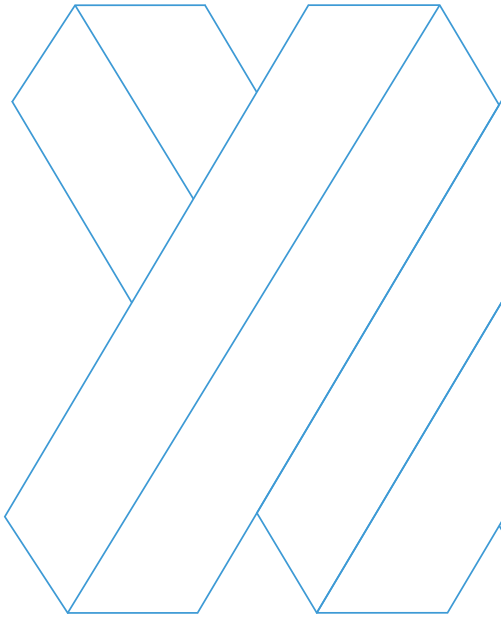
1. Understanding modern commerce terminology

When exploring digital commerce solutions available in the market, it's not always clear which underlying technology stack commerce vendors are based on. Some are described as a framework, others as a platform, yet others are API-based.

There's a great deal of confusion when trying to pin down what's what – and that complicates the selection process for business and technical leaders.

The first step in this journey is to become acquainted with frequently mentioned terms in commerce technology.

This guide aims to unpack the difference between a platform and a framework, define what APIs are, and clarify the lingo for modern commerce's underlying technologies.



2. A snapshot of tech terms

Before jumping into definitions, let's embark on an everyday scenario in the life of a developer.

A developer has to build an **eCommerce website**.

First, the developer needs a **platform** that provides the hardware and the software tools required to build and run an application throughout its lifecycle.

At the same time, the developer needs a **framework**. In simple words, a framework acts as a skeleton in any application. A software framework provides a standard way to build and deploy applications, so the developer doesn't have to start from scratch. A platform builds on the skeleton of the framework. The actual work – connecting the dots – is done by the application.

Platform providers typically offer **Software Developer Kits (SDKs)** written in various programming languages to build an application with ease. An SDK usually contains a set of classes, utilities for debugging, libraries and sample code to make integration faster and easier.

While creating the **frontend** (the customer-facing eCommerce website), the developer has to connect it to a **backend** (the data access layer that holds products, orders and customer information).

There are two ways commerce platforms bring frontends and backends together that affect the eCommerce website and user experience performance: monolithic or headless platforms.

Monolithic platforms manage backends and frontends as tightly coupled systems. While the developer can quickly deploy online stores – albeit with a template-like look and feel – these monolithic platforms lack the flexibility and speed for bespoke requirements.

Headless platforms decouple the frontend and backend, enabling communication between them via **Application Programming Interfaces (APIs)**. With a headless platform, the developer can easily create new designs, perform UX changes and add new touchpoints independently from the backend. It also gives the developer more freedom and creativity to develop unique experiences without worrying about how changes to the frontend affect the backend.

Now that we understand the most critical technical elements from a developer's point of view, let's dive deeper into these definitions and other relevant terms to your eCommerce journey.

3. What is a platform?

A platform provides both the hardware and the software tools required to run an application – be it a standalone program or one built on a framework. Mostly, a platform comes in the flavor of Platform-as-a-Service (PaaS), meaning that the code-base of the platform software itself is not distributed or licensed. Rather, it is part of a hosted solution running in a cloud that can be accessed via APIs or GUIs.

Therefore, a platform is an environment where developers build the software and where it runs throughout its lifecycle.

Developers might shape the communication between application and platform by using direct API calls or an SDK to do the heavy lifting.

Real-life platform examples are Windows, OS X, Android, iOS, etc.

SaaS versus PaaS models

Enterprises look to cloud adoption to maximize cost savings, reduce risk and innovate rapidly. Modern platforms, eCommerce and otherwise, revolve around SaaS (Software-as-a-Service) and PaaS (Platform-as-a-Service).

SaaS	PaaS
• Provides a ready-to-use, out-of-the-box solution for your business needs entirely managed by a vendor and accessed via a web browser. • The vendor controls the entire stack (frontend and backend). • Businesses depend on the provider to add extensions and improve features and integration options.	• Provides a platform to build, deploy, migrate and manage its software products. • Ideal for projects that require a high degree of customization and integration flexibility. • No limits to customization or integration. • Usually API-based systems.

Businesses can use SaaS and PaaS in combination. In eCommerce, you would choose a SaaS solution for your online store and extend it with custom extensions, such as Java code. The Java code would be made available via a PaaS. The same would be conceivable for additional database services that go beyond the functionality offered by the SaaS offering.

Cloud computing deployment models: On-premise versus cloud

A platform can be hosted on-premise, using the public cloud or a hybrid model. The fundamental difference between cloud and on-premise software is where it resides.

On-premise	Cloud
• Installed locally on a business' computers and servers, keeping the IT infrastructure on-site. • The company manages the IT infrastructure by itself or through a third party, paying for computing power and having complete control over its data. • In addition to high operating costs, on-premise installations are hard to scale, especially in the commerce environment, where there may be peaks in traffic in a short timeframe.	• Hosted on a vendor's server such as Google Cloud Platform (GCP), Amazon Web Services (AWS) or Microsoft Azure. • Increased reliability, scalability and performance through distributed data centers. • Improved redundancy against downtime and better response times since user requests are directed to the geographically closest server. • Distributed capabilities act as a safety mechanism against attackers, who can't easily compromise an entire system if they gain access to one component.

As public cloud infrastructure goes, you can spot two significant differences:

On the cloud	In the cloud
• When a platform is hosted within a virtual environment (VM) on the cloud. • While this offers an improved speed and flexibility than on-premise, the software running in a virtual machine does not have direct access to the hardware, which degrades performance and adds a step between cloud-native applications and other (micro)services.	• Also known as **cloud-native**, the software sits directly in a cloud-based infrastructure such as AWS or GCP. • Full advantage of economies of scale, multi-zone redundancy and maximum uptime. • Since updates are deployed seamlessly and in the background, there's no need to keep up with versions and deal with disruptive update processes.

Overall, cloud-native applications provide better maintainability and security, autoscale capabilities and lower total cost of ownership (TCO) from the outset.

Multi-tenancy versus single tenancy

Platforms are often used as scalable multi-tenancy systems, with a single software instance and infrastructure that serves multiple customers. Think of single tenancy as a row of shops with each business operating in its own building. Multi-tenancy, on the other hand, is a high-rise office building with different companies spanning various floors.

That means running costs are shared and performed by experts, so you don't need to worry about maintenance, upgrades or additional fees. Multi-tenant systems don't require the provisioning of dedicated environments either.

Frontend versus backend

The frontend and backend are integral parts of commerce software. While the frontend is about the customer-facing presentation layer (aka the "head" or experience), the backend is the data access layer, where data such as prices, promotions, product images, order details and fulfillment are stored.

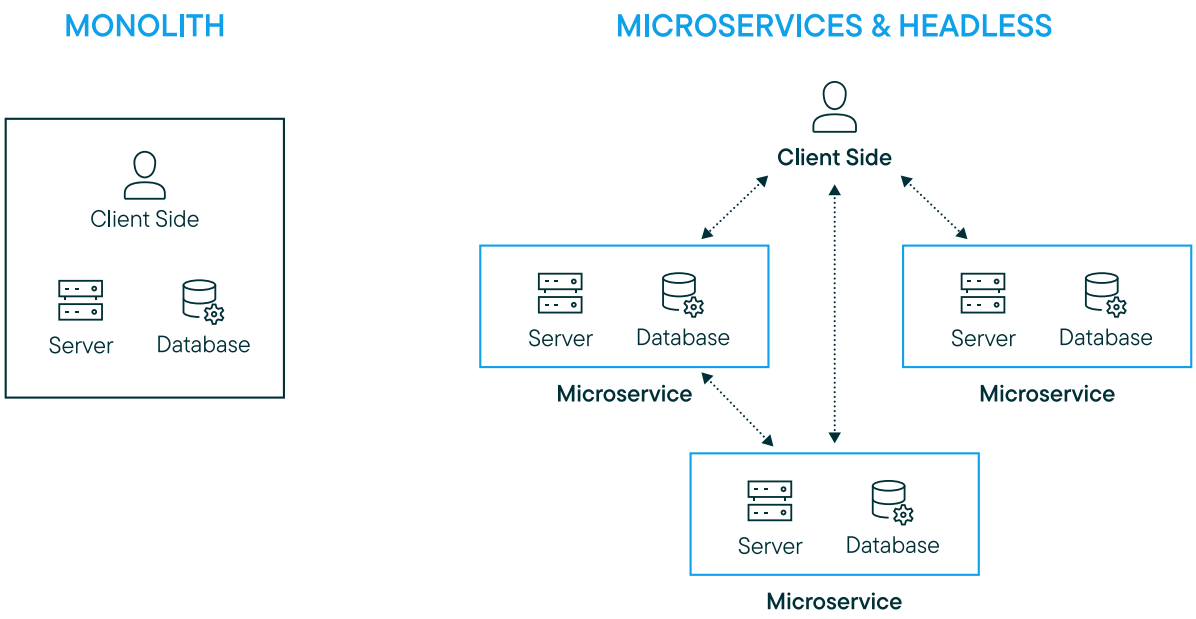
In short, the frontend defines how the online shopping experience feels, and the backend defines how it functions.

In context: Platforms in digital commerce

Gartner defines the digital commerce platform as "the core technology that enables customers to purchase goods and services through an interactive and self-service experience. The platform provides necessary information for customers to make their buying decisions, and uses rules and data to present fully priced orders for payment. The platform must have out-of-the-box (OOTB) capability or the APIs to support a self-service, interactive commerce experience."

Since the early 2000s, commerce technologies have faced numerous innovation waves. Today's digital jungle adds significant pressure on brands to change at a faster pace than ever before. As identified by Gartner, there are two co-existing streams of commerce technologies:

- Legacy, monolithic platforms that provide out-of-the-box (OOTB) capability.
- Modular, composable, API-based digital commerce solutions based on based on headless and microservices architecture.



The main difference between the two is how they connect the frontend and backend:

Monolithic platforms

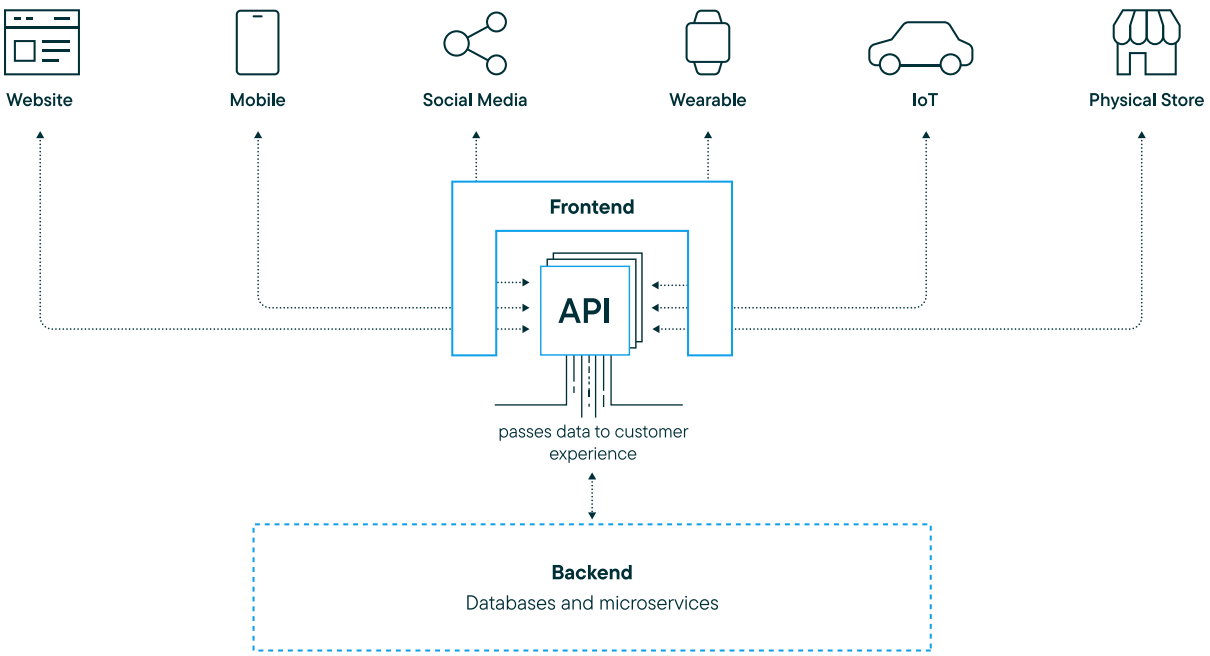
Also known as software suites, legacy platforms or closed SaaS systems, monolithic platforms bring the frontend and backend together into tightly coupled systems.

At a glance:

- Changes in the frontend affect the backend, and vice-versa.
- Functions such as pricing and orders are in the same code-base and deployed as a single, large application.
- Out-of-the-box, ready-made, template-like solution leads to lackluster experiences.
- Vendor lock-in.

Headless platforms

Also known as pre-integrated or API-based platforms, a headless solution decouples the frontend and backend.



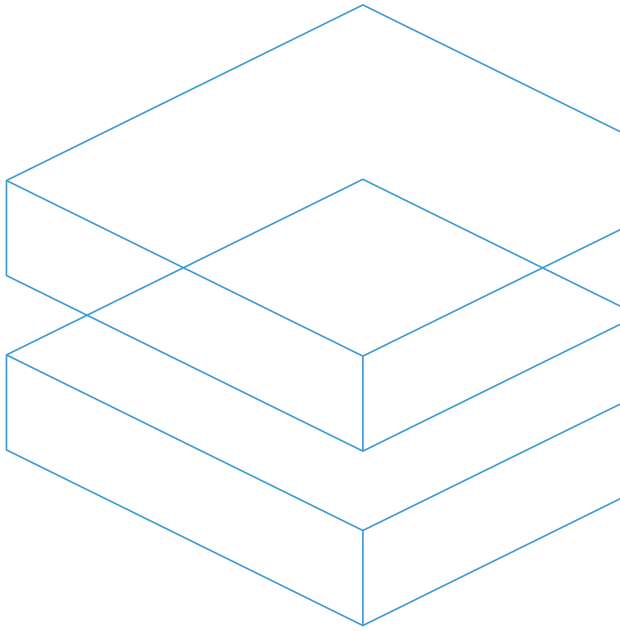
At a glance:

- Made up of unique services, each with its own code-base, in a mix of SaaS and PaaS.
- Releasing new functions, updates, etc., is done without dependencies.
- A microservice-based approach breaks down functionalities into granular pieces, so they are nimbler and more reliable while delivering faster responses and can be deployed more frequently.
- Frontend and backend communicate via granular APIs.
- Provide modular, composable commerce based on a best-of-breed approach.
- No vendor lock-in.

Modern commerce is synonymous with **API-first headless** commerce, enabling businesses to achieve higher reliability, automatic scalability and lower TCO to move the needle toward growth.

Takeaways

- A platform provides both the hardware and the software tools required to run an application – be it a standalone program or one built on a framework. Modern platforms of any kind revolve around SaaS, PaaS or a combination of both.
- A platform can be hosted on-premise or using a public cloud. To be cloud-native, a company's software and platform sit directly in a cloud-based infrastructure such as AWS, GCP or Azure.
- A piece of software is composed of a frontend (consumer-facing presentation layer) and the backend (data access layer). Commerce platforms differ in how frontends and backends are connected.
- Traditional commerce platforms, also known as monoliths, bring the frontend and backend into tightly coupled systems. While they were pioneers decades ago, their architecture doesn't easily allow for customizations, extensibility and other critical requirements of today's commerce.
- Headless commerce platforms decouple the frontend and backend, enabling companies to release new functions, updates and extensions at high velocity. Frontends and backends communicate via granular APIs, enabling companies to stay ahead of the curve.



4. What is a framework?

A software framework is a skeleton that includes preset tools, libraries, software development kits (SDKs) and other components. Consider a framework as a standardized template with components that include standard repetitive tasks for a specific programming project, removing unnecessary and uncreative busywork. The crowd wisdom of open-source frameworks to answer developers' questions also saves time.

Just like programming languages, a framework serves specific purposes; some will be designed for frontend, such as Vue.js and Angular, while others are meant for the backend, such as Spring (used by SAP Hybris), Ruby on Rails (used by Shopify), Django, Node JS and more.

Framework versus software

Software is a set of instructions that run on a machine. A framework is also software developed and used by developers, such as web applications, data science and mobile development frameworks, but works as a basis for developing other software.

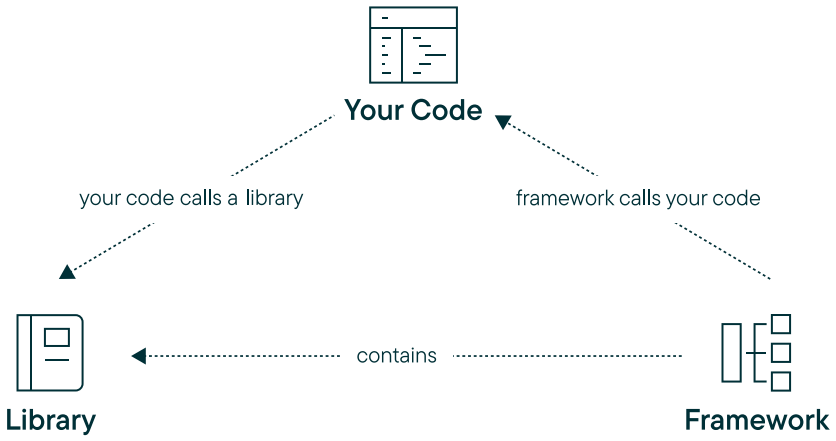
Framework versus programming language

While a framework is the organizational structure of any application's code, a **programming language** is a set of written rules, phrases and patterns to provide commands to computers on what to do next. It's like French and English, but for computers. Examples include Java, C#, Python, PHP, etc.

Framework versus library

A **library** is a packaged, reusable chunk of code designed to perform a specific function – or set of closely related functions. A developer inserts a library into an application and calls it via an API when that function is required, without writing the code from scratch. Like frameworks, a library enables developers to lower development time and costs by encouraging code reuse. Think of a framework as an abstract form of a library.

At the same time, a framework can contain one or more libraries, and libraries can also be standalone and don't need to sit within a framework to be functional.



Framework versus toolkit

Toolkits only contain classes and don't specify the structure as a framework, working as a more focused library with a defined and specific purpose. This term has fallen out of favor and is used almost exclusively for graphical widgets and GUI (graphical user interface) components.

Framework versus SDK

While a framework serves as the foundation for developing applications, an **SDK** (Software Development Kit) provides the necessary tools for application development, consisting of a set of classes, utilities for debugging and sample code to make building a platform application easier. It's like a meal package at a fast-food restaurant containing many things neatly packed.

As with frameworks, developers download these SDKs to their systems and start working locally. In contrast to frameworks, however, an SDK has no use on its own but is only helpful when writing applications for one particular software or platform. In other words, an SDK is only a tool to write an application, whereas a framework becomes part of an application.

In summary, SDKs and frameworks complement each other, and the choice of SDK depends on the framework or programming language used.

Framework versus platform

Sometimes framework and platform are used interchangeably, though they aren't the same thing. While the framework provides a generic structure as a skeleton to code software, a platform offers both the hardware and the software tools needed to build and run an application.

In context: Frameworks in digital commerce

Frameworks, in general, allow developers to optimize their work; however, the level of optimization and substantiality of results depends on the environment. When the frontend and backend are decoupled, developers can create the software from scratch and have more freedom to produce outstanding experiences.

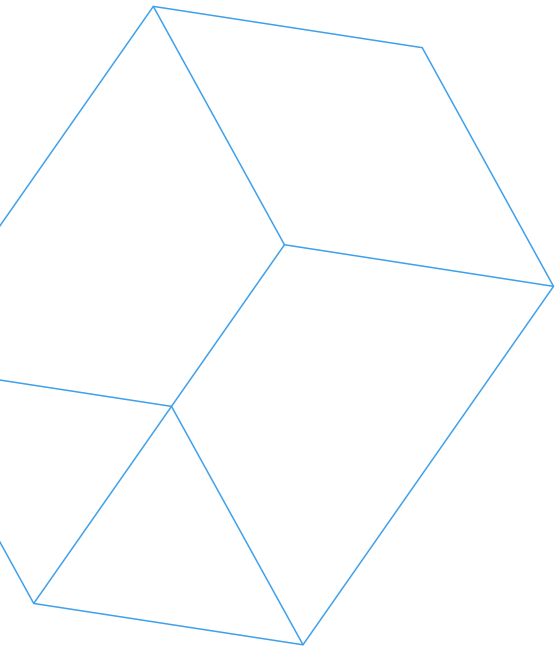
Frameworks are helpful for developers working on monolithic platforms, too. Yet, they mainly fulfill small chunks of code for specific functions due to the fixed nature of the system.

For frontend developers, for instance, using a framework such as Vue.js or Angular means creating a unique look and feel in an eCommerce website while avoiding repetitive tasks such as persistence, routing and session management. This is particularly beneficial when combined with a headless commerce solution.

The use of SDKs in the context of digital commerce is to allow easy integration of a company's eCommerce application with the API provided by a headless solution.

Takeaways

- A software framework is a skeleton that includes preset tools, libraries, software development kits (SDKs) and other components.
- In commerce, frameworks enable developers to create unique software for frontends and backends without running into repetitive tasks that slow down their pace.



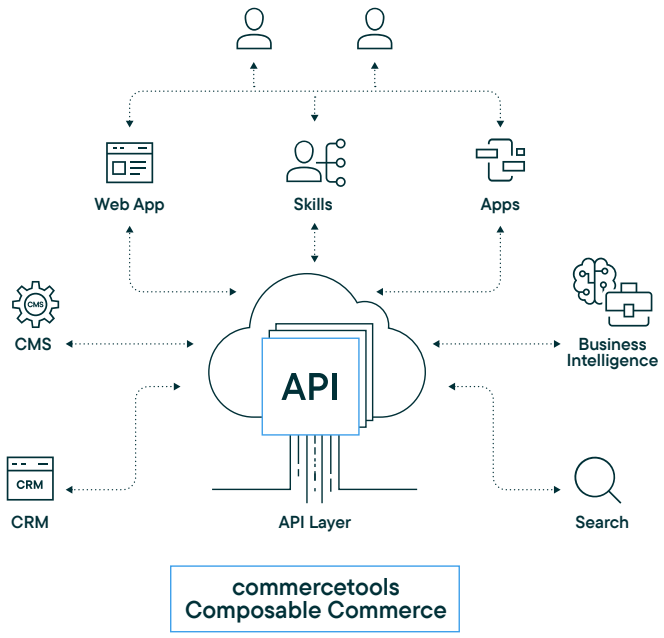
5. What is an API?

Application Programming Interfaces, or APIs, are interfaces built on top of the software, an application or a library that allow two systems to connect and communicate.

To illustrate how APIs work, imagine you're in a restaurant and want to order a dish. The waiter collects your order, delivers it to the chef, and then brings your order when ready. In this scenario, the waiter represents the API – the communication bridge between two parties.

At a glance, an API:

- Is an interface on top of an application or library that allows callers to execute functionality (e.g., calculate tax, query inventory, etc.) or create/read/update/delete data (products, orders, prices, etc.).
- Enables companies to consume functionality and data as a service over an API, rather than downloading, installing, configuring, running and maintaining large stacks of software and hardware.
- Allows outside developers to wire performance and data into their applications with granular pieces of functionality, rather than adopting a large one-size-fits-all software package.
- Lets different "heads" (frontends) request the same content from a backend and render it differently, according to the needs of a particular device or channel.
- Is the thread that ties functionality between your backend and frontend to power your commerce experience.



REST APIs

Representational State Transfer (REST) is a set of rules developers follow when creating an API, applying HTTP methodologies to retrieve, edit, create and remove data. Existing HTTP methodologies mean developers do not need to install libraries or additional software to take advantage of their design. REST APIs are the default means of exposing commerce-related data and functionality from microservices.

Microservices

The microservices architecture combines loosely coupled, independently deployable small components or "services" to compose a single, more complex application. These applications deliver faster responses, are more reliable and can be deployed frequently. Microservices have a single purpose: to solve a granular problem and interact with the world via well-defined APIs.

APIs versus GraphQL

Although REST APIs provide an excellent backend solution, they pose problems for frontend developers when identifying the right APIs to perform core functions and how to interpret data. Over-fetching or under-fetching data are recurring API problems that may cause performance issues.

Imagine if an Apple Watch fetches an entire two-megabyte payload over a cellular network every time a user wants to look at product details. The same thing happens when the API under-fetches data. This requires multiple, serial round-trip calls with limited computing power or bandwidth, which can be very expensive or consume too much power.

This is what **GraphQL** is primed for: It allows developers to retrieve the exact data from any source. As a data query language for APIs, GraphQL gives developers the accurate information they need, removing the need to handle large quantities of unnecessary data and creating workarounds to reduce API calls.

Since it serves as a layer on top of APIs, GraphQL complements instead of replaces APIs.

APIs versus webhooks

As an API enables two-way communication between software applications driven by requests, a webhook is a lightweight API that powers one-way data sharing triggered by events. Instead of one application requesting another to receive a response, a webhook is a service that allows one program to send data to another as soon as a particular event takes place. For example, a webhook sends a notification when an event occurs, such as a processed payment.

In context: APIs in digital commerce

Commerce APIs enable communication between frontends (website, mobile apps, IoT devices, etc.) and the backend (data layer), and manage typical commerce functions:

- **Inventory API:** Read and adjust inventory information for product variants using an API.
- **Order API:** Access order history for one-time purchases, subscription orders, import orders from third-party distribution channels, etc.
- **Product API:** Manage physical products, including their variants and images.
- **Transaction API:** Access financial transactions for orders and donations.

API-first, microservice-based commerce platforms offer a flexible way for businesses to tailor experiences. With the hybrid "build and buy" approach, companies can use commoditized functions (e.g., cart, orders) as an API provided by a headless platform and build unique features on top to differentiate.

Using GraphQL and webhooks on top of APIs, companies can maximize efficiency and improve performance for specific use cases, channels and devices.

Headless platforms are increasingly unbundling large offerings into granular APIs, such as checkout, discounts, etc., addressing the real needs of big brands and retailers to build their commerce stacks from scratch for awe-inspiring customer experiences.

Takeaways

- Application Programming Interfaces, or APIs, are interfaces built on top of the software, an application or a library that allow two systems to connect and communicate. APIs are the thread that ties functionality between your backend and frontend to power your commerce experience.
- GraphQL takes a step forward in pinpointing the exact data for faster performances, laying on top of APIs.
- API-based platforms are taking over the monolithic approach as the digital commerce solution of the future.

6. Modern commerce starts with MACH and commercetools

Microservices-based | **API-first** | **Cloud-native** | **Headless**

These core concepts form **MACH**, the underlying architecture of modern commerce technology.

MACH architecture was created to provide a high level of control and agility for any business that wants to innovate rapidly. In this modular environment, companies can ditch monolithic, legacy platforms and respond to customer needs on the fly.

This innovative architecture brings a raft of benefits:

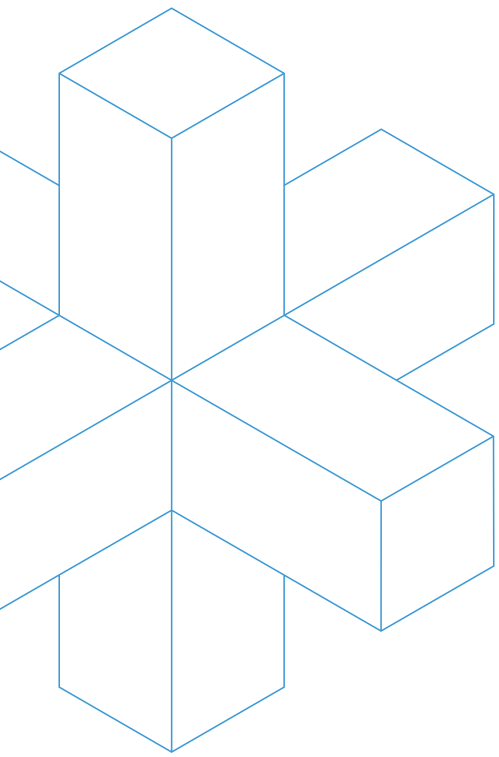
- With **microservices**, companies deploy granular services that serve a specific purpose. Because microservices are by default independent services exposed via APIs, it's much faster and easier to implement new features, extensions and updates, creating a bespoke experience from end to end.
- With an **API-first approach**, companies easily bridge communications between various frontends and a centralized backend. GraphQL helps pinpoint the exact data needed in those interactions for leaner and faster frontend performance.
- With a **cloud-native solution**, companies autoscale infinitely and say goodbye to website crashes due to traffic spikes.
- With **headless**, companies decouple the ever-growing number of frontends from the backend for maximum flexibility and speed. It helps create fast websites, handle intricate product data, customize products, launch short notice promotions and release new functions without headaches.

Composable commerce is how businesses consume MACH, picking and choosing the **best-of-breed** components in a commerce solution, frontend and payment processor that perfectly fit their requirements. Think of it as LEGO bricks that provide infinite combinations, offering custom, unique ways to differentiate your business.

This is what commercetools, the inventor of MACH, can deliver for you.

commercetools offers headless commerce solutions, services and a portfolio of many products that can be individually or collectively used:

- **commercetools Composable Commerce** is the backend. It functions as the core of the commerce ecosystem and consists of over 300 of the best commerce APIs.
- **commercetools Frontend** helps you create stunning frontends with rich content experiences to differentiate your brand, as it is quite flexible and easy to customize.
- **commercetools for Growth:** growing merchants and brands competing against larger retailers can go headless with pre-bundled kits (backend + frontend + search) that offer the best tech stack components to build your MACH-based commerce architecture.



7. Glossary

- **Application Programming Interface (API):** An interface built on top of the software, an application or a library that allows two systems to communicate.
- **Backend:** The data layer where the product and customer data is stored and managed.
- **Best-of-breed:** A concept whereby you can pick and choose the best commerce components that fit your business needs without vendor lock-in.
- **Cloud infrastructure:** When a company's platform is hosted on a vendor's server such as Google Cloud Platform (GCP), Amazon Web Services (AWS) or Microsoft Azure.
- **Cloud-native:** When a company's software sits directly in a cloud-based infrastructure.
- **Commerce APIs**: REST APIs enable communication between eCommerce frontends and the backend, managing commerce functions like inventory, order, etc.
- **Composable commerce:** A development approach of selecting best-of-breed commerce components and "composing" them into a custom application built for specific business needs. It's also how businesses consume MACH.
- **Framework:** A software-only skeleton that includes preset tools, libraries, SDKs, etc.
- **Frontend:** One or more consumer-facing channels, such as a website, IoT device, etc.
- **GraphQL:** A data query language for APIs to retrieve the exact data from any source.
- **Headless:** A headless commerce solution decouples the frontend UX/UI from the backend, making it easier and faster to build cutting-edge shopping experiences.

- **Library:** A collection of classes and functions.
- **MACH:** The underlying architecture of modern commerce platforms based on microservices, APIs, cloud-native and headless. Invented by commercetools.
- **Microservices:** Architecture that combines loosely coupled, independently deployable small components to compose an application. It's exposed via well-defined APIs.
- **Monolithic platforms:** All-in-one commerce SaaS solution sold as one unit.
- **Multi-tenancy:** A single software instance and infrastructure that serves multiple customers.
- **On-premise:** When a company's platform is installed locally, on a business' computers and servers, keeping the IT infrastructure on-site.
- **Platform:** Provides both the hardware and the software tools needed to run an application – be it a standalone program or one built on a framework.
- **Platform-as-a-Service (PaaS):** Provides a platform to build, deploy, migrate and manage software products.
- **Programming language:** A set of written rules, phrases and patterns to provide commands to computers on what to do next.
- **Representational State Transfer (REST):** A set of rules to create APIs.
- **Software:** A set of instructions that run on a machine.
- **Software-as-Service (SaaS):** Provides out-of-the-box services managed by a vendor.
- **Software Development Kit (SDK):** Provides the necessary tools for application development in a neat package.
- **Software suites:** A synonym to monolithic platforms.
- **Toolkit:** Contains classes and doesn't specify the structure, working as a more focused library with a defined and specific purpose.
- **Webhook:** A lightweight API that powers one-way data sharing triggered by events, allowing one program to send data to another as soon as a particular event occurs.

About commercetools

commercetools is a next generation software technology company that offers a true cloud commerce platform, providing the building blocks for the new digital commerce age. Our leading-edge API approach helps retailers create brand value by empowering commerce teams to design unique and engaging digital commerce experiences everywhere – today and in the future. Our agile, componentized architecture improves profitability by significantly reducing development time and resources required to migrate to modern commerce technology and meet new customer demands.

The innovative platform design enables commerce possibilities for the future by offering the option to either use the platform's entire set of features or deploy individual services, la carte over time. This state-of-the-art architecture is the perfect starting point for customized microservices, enabling retailers to significantly reduce time-to-market for innovative commerce functionalities.

commercetools has offices across the US, Europe, and Asia Pacific, with headquarters in Germany. Since its founding in 2006, commercetools software has been implemented by Fortune 500 companies across industries, from retail to manufacturing and from telecommunications to fashion.

www.commercetools.com

Munich - Berlin - Jena - Cologne - Amsterdam - Zurich - London - Valencia - Durham NC - Melbourne - Singapore - Shanghai