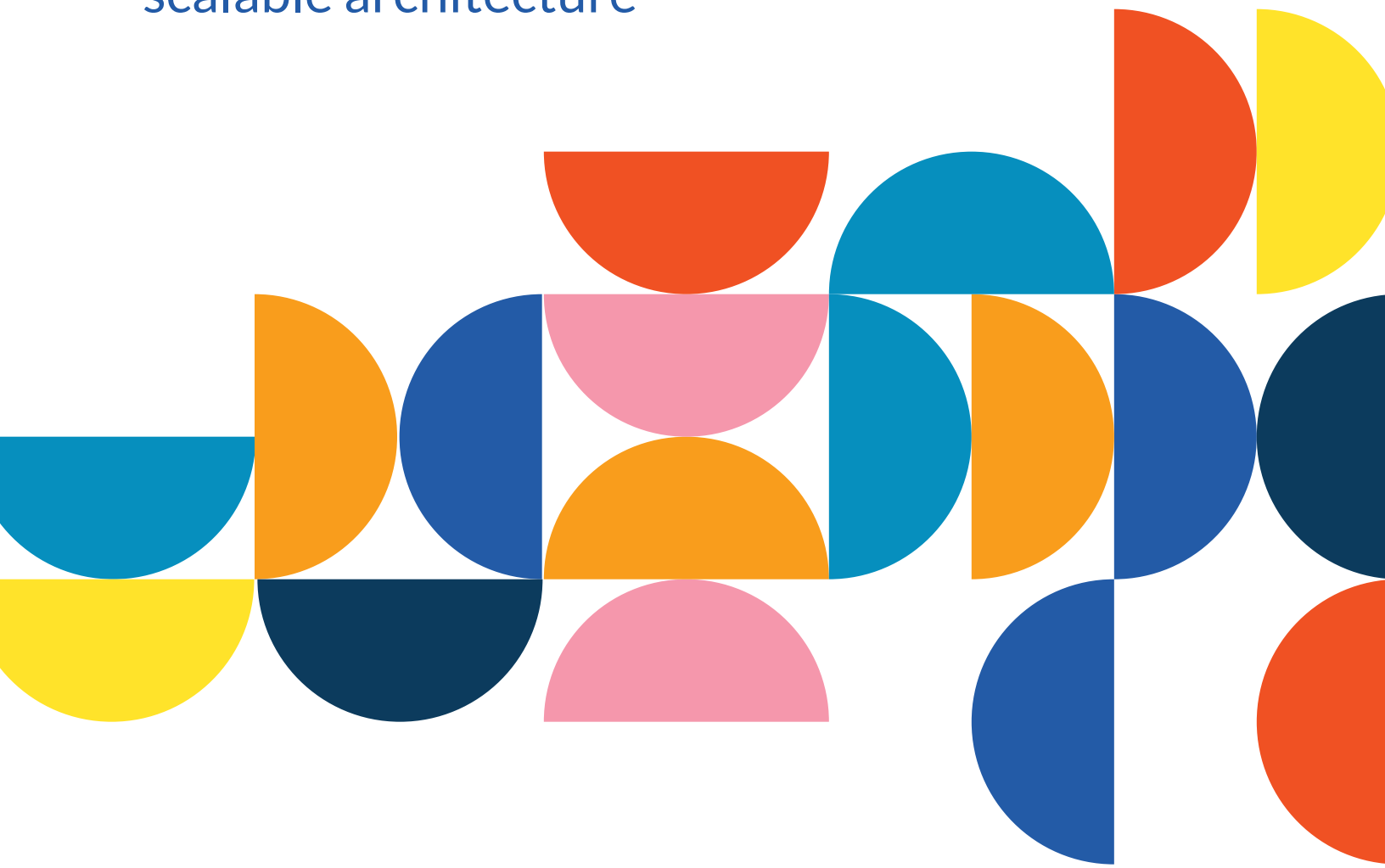
Building the composable enterprise

From legacy monolith to modern, scalable architecture



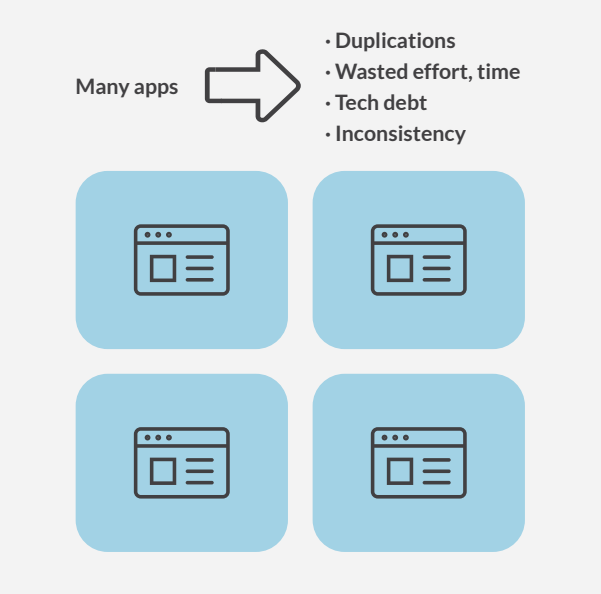
Fusion Alliance

We've reached a fork in the road.

The accelerating pace of business change demands innovation and adaptation, even as customers and employees increasingly expect more contextualized and personalized application experiences.

Many organizations have reached a tipping point. Their current application portfolios were designed to address the challenges of the past—and are unable to sustain today's pressures. In many cases, legacy applications have ceased performing as enablement tools and have become active obstacles to innovation and growth.

The problem with web monoliths



The future of business is composable.

Even as more and more companies feel the pinch of outgrowing their portfolios, they're often unable to make a wholesale switch. The costs and risks associated with a major architecture change can be staggering, especially for mid-market companies.

Smart business leaders are taking a new approach to their application ecosystems. Instead of choosing between a status quo that isn't working or a radical overhaul with high up-front costs, trends point to the value of a more strategic framework for purchasing, implementing, and maintaining applications.

And application vendors are taking notice. In contrast to the monolithic platform models of the past, new offerings are more modular and consumable through different delivery channels, touchpoints, and modalities.

For most organizations, these changes require a new architectural approach: the composable enterprise.

> "As business needs change, organizations must be able to deliver innovation quickly and adapt applications dynamically — reassembling capabilities from inside and outside the enterprise. To do this, organizations must understand and implement the composable enterprise."
>
> – Gartner

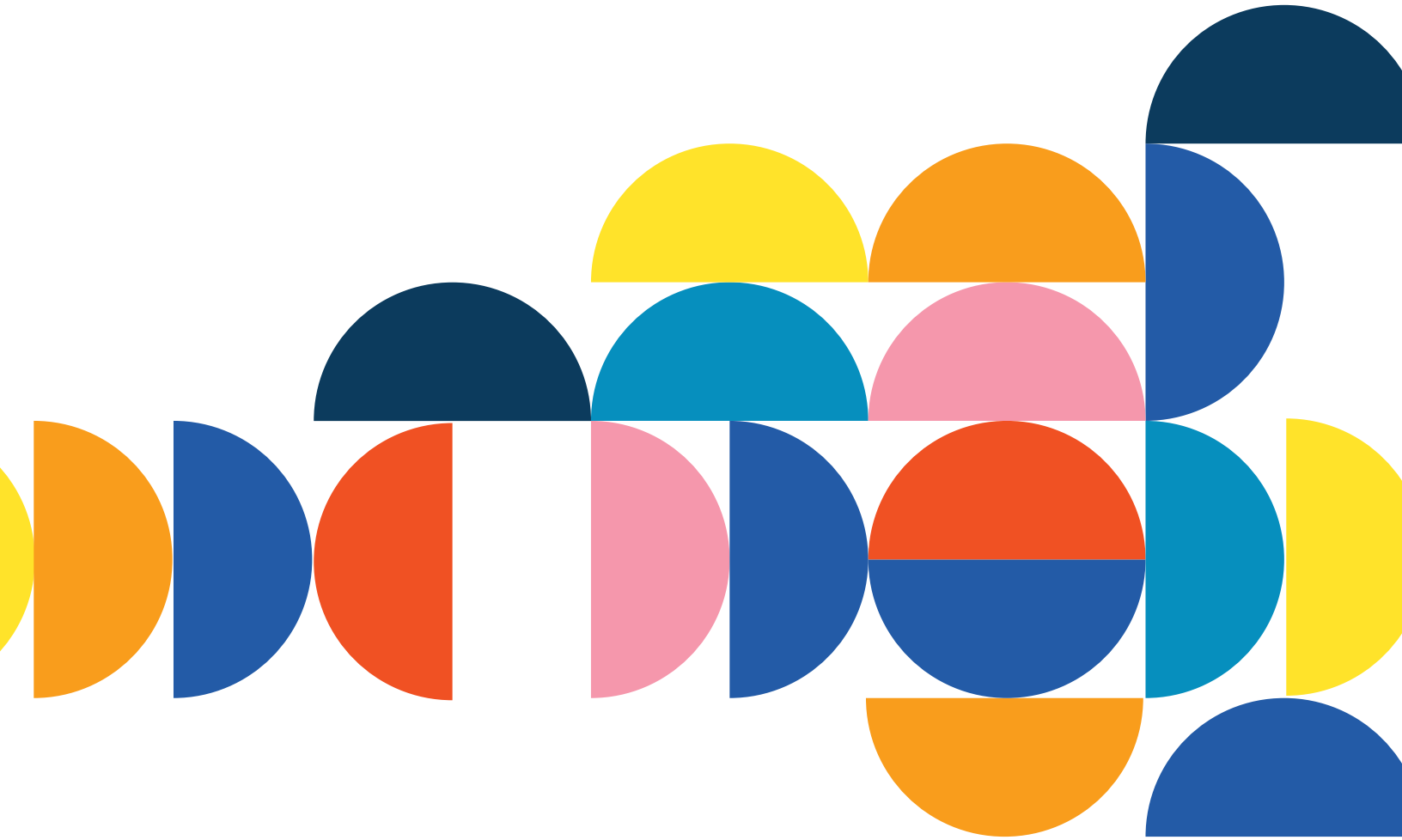
Defining composability

Composability takes a modular approach to applications, allowing businesses to swap out and reuse components on an as-needed basis so that needed upgrades and functionality pivots can happen in waves rather than all at once. As a design approach, composability enables systems — including IT systems, organizations, products — to adapt quickly to changes while staying resilient.

True composability leads to stable systems that are easier to maintain and upgrade over time. Composability makes it possible for companies to limit technical debt and take advantage of new technologies as they come along.

What's new about this approach

Application architecture has evolved over time as digital needs and the business landscape have changed. The project-based packaged application framework popular in the early 2000s gave way to a product-focused delivery style in the 2010s, followed by today's emerging shift to marketplace delivery of composable applications.

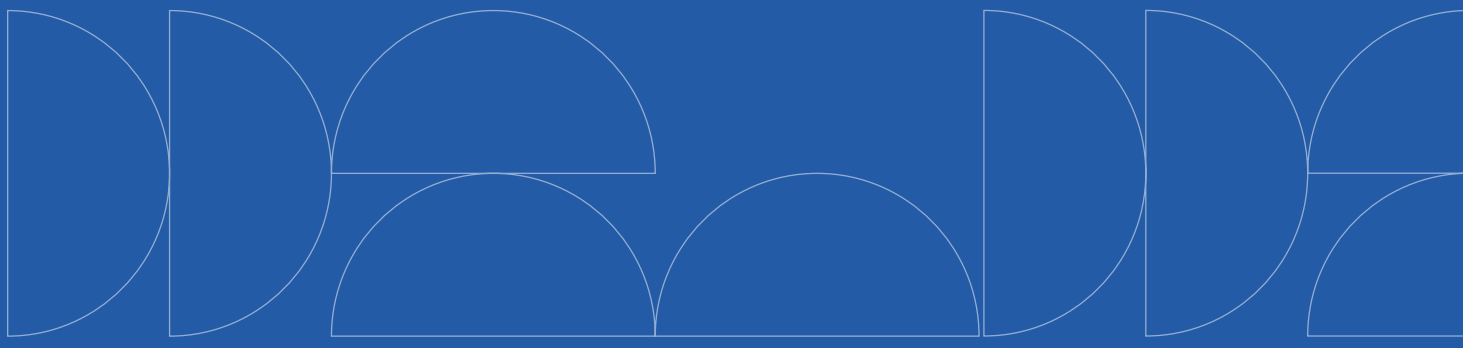


What's new about this approach

Modular software architecture has been available before, but what composability offers is a new and foundational focus on bringing business users to the table and equipping them to influence features and functionality of the tools they use in practice. Where traditionally only professional software engineers were involved with application selection, development, and maintenance, composability involves and enables multidisciplinary teams.

"Neither business nor IT professionals can meet the demands of the increasing pace of business change alone. Composable enterprise architecture must serve the multidisciplinary teams of business and IT professionals as its software engineering constituency."

– Gartner



Benefits of composability

Composability offers today's businesses the flexibility, adaptability, and efficiency they need to stay competitive in rapidly changing business environments.

Agility

Old business models hampered by rigid hierarchies and structures often find it hard to adapt to change. Composability gives organizations the ability to shift gears, migrate workloads and functionality quickly, and adopt new ways of working as circumstances shift.

Traditional vs Composable: Comparing Approaches

	Traditional	Composable
Goal	Efficiency	Flexibility/Agility
Value	Cost leadership with/at scale	Highly attuned to business environment
Approach	Technology drives efficiency and scale	Enable multiple outcomes simultaneously
Governance	Top Down Plan-driven; Approval-based; Safe	Distributed emergent, empirical, continuous, calculated risk
Sourcing	Conventional enterprise services	Composable multi-SME teams
Talent	Generalists and Specialists	"Versatilists"
Culture	Values low-risk, predictable execution	Values rapid responsiveness to internal and external environmental changes
Cycle Times	Long (months/quarters/years)	Hybrid long and short (due to initial modularity efforts)

Adaptability

To stay profitable, companies continually adopt new digital platforms and services. Attempts to resist change can hamper efficiency and innovation and make teams less productive across the business. Because composable architecture reduces the friction associated with change, it simplifies the path to a digital-first workplace and improves employee experience.

Efficiency

Security and change management policies may dictate the use of on-site data storage, private cloud configuration, and other specific environments. Building and maintaining those application-specific architectures can be costly and time-consuming.

By contrast, composable infrastructure operates exclusively in virtual environments, delivering significant efficiency gains in terms of resources, costs, and availability.

Challenges of composability

Complexity

As you move toward composability, the number of modules you create and orchestrate can rise exponentially. You'll need the architecture and tools to manage this complexity, as well as a "marketplace" to implement advanced forms of discovery and to manage advanced forms of metadata.

Control

If you source any software or application modules from outside your organization, it might be more difficult to exercise the control that you need over them within your own space. Your new marketplace must also apply policies to manage the distribution of data.

Risk

If you share modules outside the boundaries of your organization, you risk spreading your data and intellectual property into the ecosystem. Again, you'll need to rely on your marketplace to apply data sharing policies and only allow modules in the endpoints that protect internal data integrity.

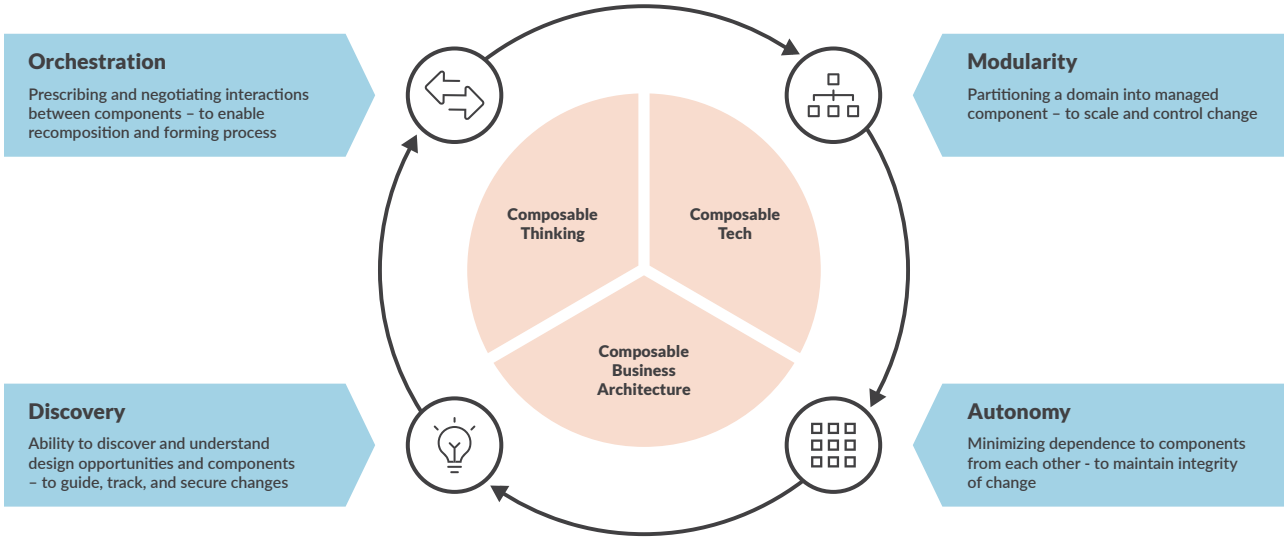
Skills

To succeed at composability, your team will need new skills, competencies, and tools, as well as time to learn and gain experience. For example, teams will require familiarity with APIs and microservices, as well as communication skills to deal with a larger and more diverse stakeholder universe.

Core design principles for composable architecture

Composability requires a new architectural approach. Accepting disruptive change as the norm can be a mindshift for many organizations. Composable architecture build resilience into the model by making things modular, allowing a mix and match approach to business functions that helps you respond creatively to ensure positive outcomes.

Core design principles for composable architecture



Source: Gartner

Modularity

Whether it's applied to software, organization, or business model planning, modularity is central to composability. Whatever building blocks are provided for the composition define the character of the resulting architecture.

One of the key differentiators of the composable enterprise experience is that application design and redesign (composition and recomposition) are performed with direct participation of business and technology professionals operating as multidisciplinary teams. That's why a composable enterprise application must contain a well-defined set of packaged business capabilities as independent building blocks—small enough to maximize agility, but large enough to ensure integrity.

Autonomy

In the composable enterprise, each modular component must be self-contained, so that the removal or replacement of a given component will minimally burden other components in the application. While complete autonomy may not be achievable, the goal is for each module to be capable of operating alone or in new, unanticipated combinations.

Discovery

To achieve composability, teams must be able to easily find, assess, and integrate components. Discoverability requires clear, complete documentation of operational characteristics, performance metrics, and resource consumption factors for each module. This metadata should be easily accessible in your internal component marketplace.

Orchestration

Orchestration is the gauge that measures the quality, openness, safety and controllability of the encapsulation. Each component in your composable enterprise must be prepared for composition, integration and governance in development and runtime environments. Orchestration also measures how well the components are equipped for monitoring, tracking, securing, and DevOps operations, as well as other forms of governance.

Prioritizing components

By nature, composable architecture implies iteration. As your organization moves into composability, you'll need to prioritize segments to modularize first. Successful transitions usually start small and grow over time. Applying composability incrementally will give your business time to demonstrate the value of the concept and train your teams to operate with a new framework.

To move toward composability, identify which parts of your organization, such as customer segments or products, change quickly and often, as you're likely to see the quickest time to benefit there. Look for opportunities to improve, as your team's skills and competencies grow. Then expand the approach to other areas over time.

As you go, establish your internal marketplace as a catalog to institute a controlled environment for a growing collection of endpoints and use cases.

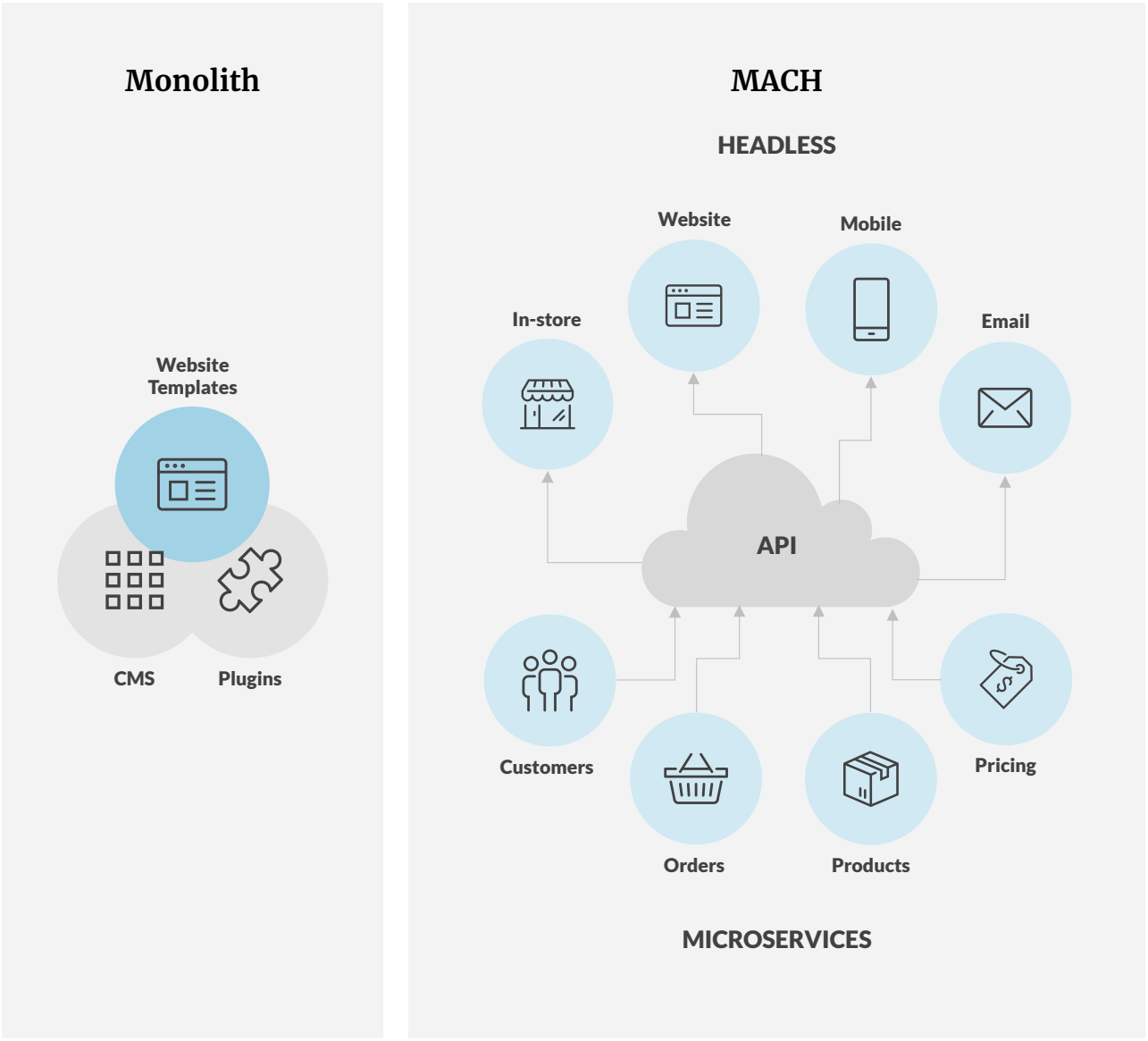
Key features of a composability framework

As your composable architecture evolves, your technology principles may shift to keep up. Industry best practices for building modular enterprise software follow four key principles known by the acronym MACH. Unlike monolithic application approaches, MACH principles build toward agility.

> "Monolithic application experiences no longer meet the requirements, expectations or preferences of innovative business users and their customers, now that they demand continuous business agility."
>
> – Gartner

Key features of a composability framework

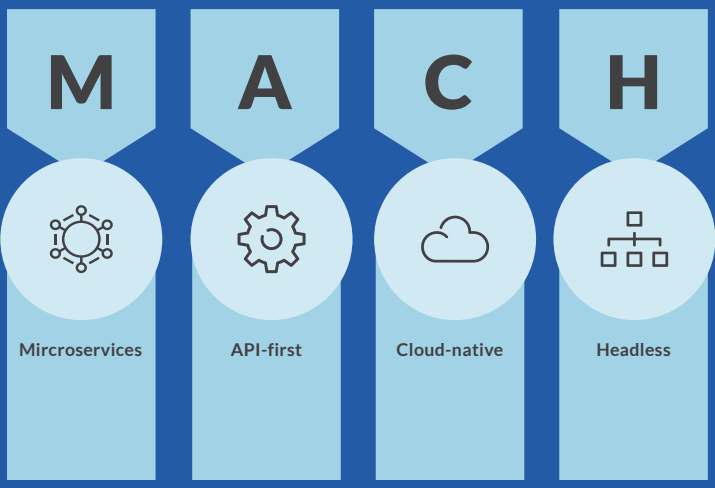
The ideas behind MACH development help your business build toward greater composability. While it's not a silver bullet, these principles can deliver the flexibility you need to stay competitive in a rapidly shifting business landscape. The following sections unpack each aspect of MACH principles and outline the pros and cons that might impact your composable architecture.



Source: machalliance.org

Microservices

Microservices come in all sizes, can be developed and deployed independently, and can be combined with other elements to form a single app or larger applications of the future. Each microservice solution typically serves a single function. You can think of microservices as a series of stores in a shopping mall.



> “Instead of tearing down the mall [using a shopping mall as an analogy for microservices] each time something changes, with microservices you can add or remove “stores” on demand. If you have seasonal demand, you can quickly add pop up shops to handle the volume and tear them down just as easily without impacting the other stores.”

– Developer chatter on Reddit

Mircroservices

There are upsides and downsides to microservices.

Pros

Easy to scale: Microservices can each be scaled independently in real-time based on traffic and demand for each individual service.

Less costly: As each service can be scaled independently you no longer have to over-provision server capacity to anticipate demand; this enables you to optimize your infrastructure costs.

High uptime: Downtime can be isolated to individual microservices in a decoupled architecture ensuring continuing or a slightly reduced service operations in a worst-case scenario. If one service is down, the others are still able to operate. Redundancy will need to be baked in should a critical path microservice go down.

Vendor flexibility: You gain control over your entire architecture, enabling you to make smaller decisions to solve specific problems. This allows you to select best-in-class components for each area of your business, whether that's on the frontend, backend workflow management or logic processing. You can also seamlessly write and integrate your own internal microservices that are independent and under your entire control.

Faster upgrades & release cycles: Each microservice can be updated independently from each other enabling the rapid release and roll out of new features, functionality and bug fixes that do not affect other areas of the system.

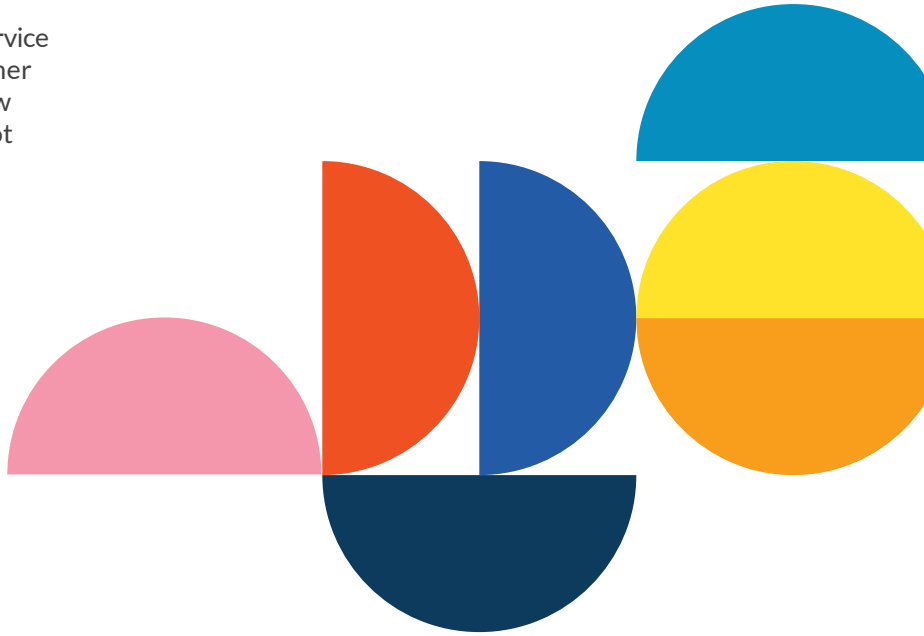
Cons

Complex architecture: An organization will require a technical team to implement and maintain a microservice-based architecture.

Developer bottlenecks: Complex architecture can lead to your technical team becoming a bottleneck when it comes to integrating new services or creating custom functionality.

Failover redundancy: The key weakness of microservices comes down to orchestration. In a decoupled or loosely coupled microservice architecture, one microservice going down (such as the cart or checkout) may lead to a cascading effect within the integration points between microservices. Failover and redundancy will be required to ensure these errors are handled gracefully.

Increased operational overhead: While it's true that the software itself is likely to be more scalable and easier to optimize from a server resource perspective, you will likely need to hire an additional small team of developers to create new functionality and maintain this type of architecture.

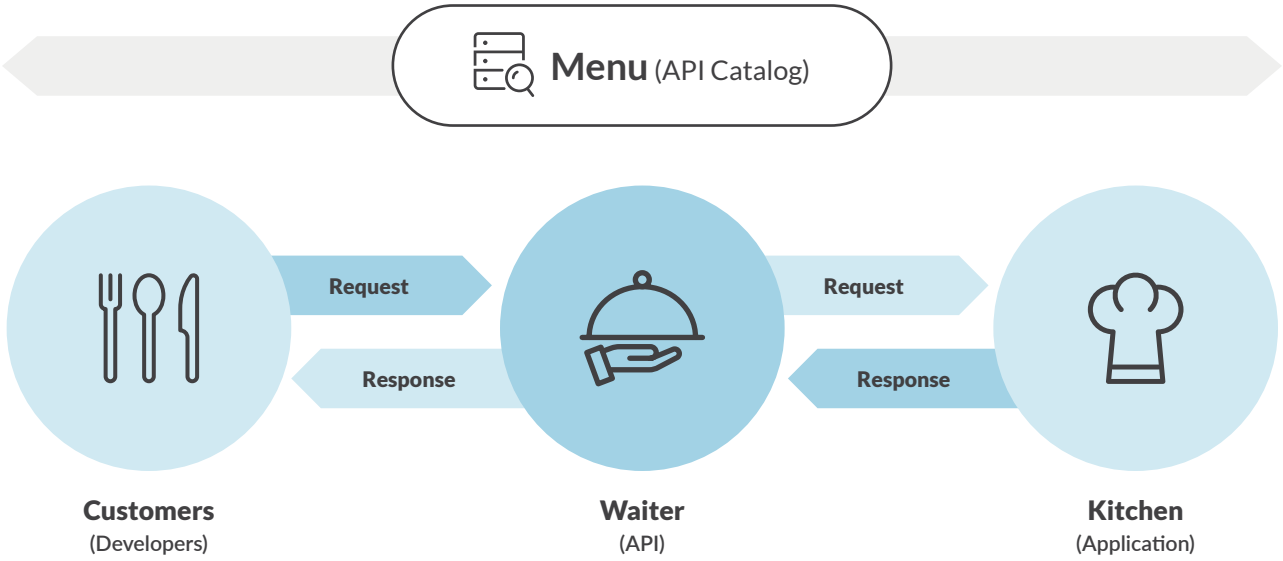


API first

The traditional monolithic approach to software design focused on creating "out-of-the-box" capabilities, and access via APIs was typically an afterthought rather than the primary access point for a given feature. In an API-first solution, APIs are the bedrock of a product offering, ensuring full coverage of all features and functionality at a programmatic API level.

What is an API?

An excellent analogy for an API is a restaurant's wait staff, who act as the go-between between patrons and the kitchen, ensuring that orders, modifications, and special requests are passed on to the chefs.



API first

The API-first strategy also has pros and cons to keep in mind.

Pros

Any frontend: You are free to select any frontend technology or framework while ensuring you'll have access to implement any features required.

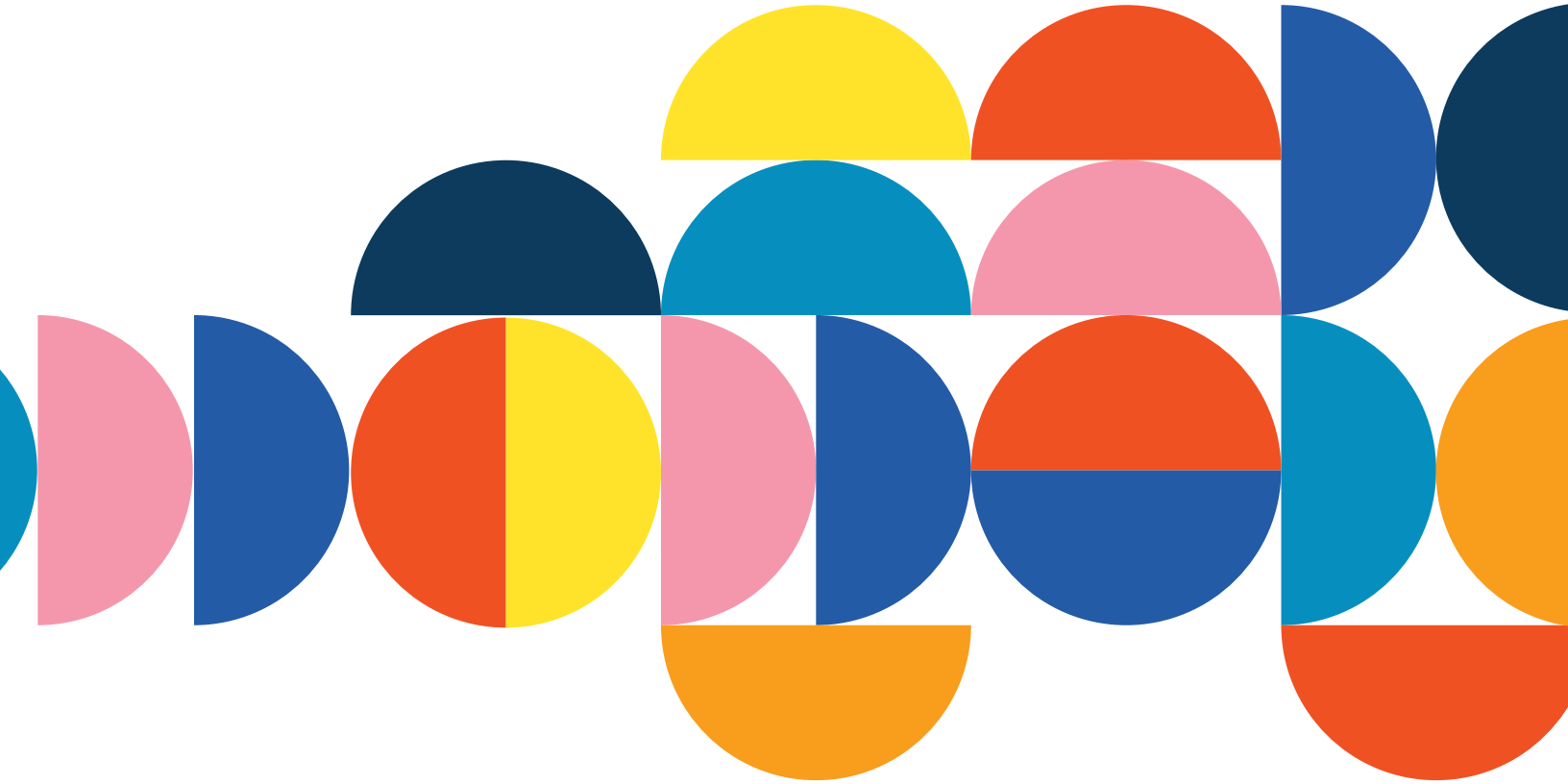
Better developer experience: A well-designed, abstract, and consistent API will reduce the learning curve for developers and mask some of the complex logic that happens behind the API-layer.

Accelerated time-to-market: With a reduced learning curve and the ability to implement any frontend technology or framework, you can deploy new solutions into the market rapidly.

Cons

Developer requirement: You will need a development team on hand to implement. This may increase your operational overhead.

API quality: API-first does not mean the API is well designed. Not all APIs are created equal and a deeper dive into API and design decisions will need to be evaluated on a case-by-case basis by your engineering team.



Cloud native

Cloud-native refers to software delivered via the cloud provider's model by default during product development. This excludes non-cloud-based software that has been put into the cloud after initial development. Many software-as-a-service solutions operate on a cloud native model.

While this approach does offer many benefits, SaaS solutions do have some drawbacks you'll need to be aware of.

Pros

Turn-key: SaaS solutions can be deployed rapidly right out of the box, so to speak.

Reduced complexity: SaaS abstracts complexities of hosting solutions by managing this for you.

Increased scalability: SaaS deployed in the cloud can be auto-scaled to support traffic demands and match your long-term business growth without the headaches of minute-by-minute management.

Robust & reliable: Cloud-native applications provide built-in redundancy by deploying their services to multiple data centers and availability zones to reduce latency and increase uptime and performance.

Automatic upgrades: New feature releases are handled seamlessly on your behalf, reducing overhead.

Cons

Black-box: The most important drawback in a purely cloud-native solution is that they are typically black boxes. This can sometimes be addressed to some extent by ensuring features are flexible, integrations are simple, and the product is well documented.

Limited deployment options: It is unlikely that a cloud-native vendor will offer private cloud or on-premise deployment options.

Data security: Security is no longer in your hands and will need to be managed independently with each SaaS vendor you decide to purchase from.

Troubleshooting: With cloud-native SaaS software it can be more difficult to determine where bugs and faults are located.

Headless

The term “headless” comes from the concept of separating the head (frontend) from the body (backend) and connecting the two pieces through APIs. This principle offers significant benefits, although it does come with a few drawbacks in terms of timing.

When considering headless implementations, you’ll want to consider the following factors.

Pros

Multi-channel: Being headless allows you to deploy multiple frontend experiences (heads) across any channel or device. This enables your brand to connect with customers at any touch point, wherever they are in their customer journey.

Flexibility: Headless solutions empower businesses to select the right frontend tools, frameworks, and languages that match their development team’s skillset and their business requirements.

Lightning-fast load times: Leveraging modern technologies and frameworks can dramatically increase performance.

New business models: Separating backend business logic from frontend views and templates enable businesses to launch new business models and drive revenue growth. By leveraging a headless approach, you could rapidly launch new sales channels such as social commerce, IoT/voice commerce, or curbside pick-up experiences.

Cons

Increased cost and slower time to market: Frontends must be bought and integrated or built from scratch. It may take longer create a traditional grid-based template than to implement a traditional, template-driven SaaS. On the other hand, creating a personalized, unique, and differentiated experience is faster if you start with a headless architecture.

Take the next step

Making the shift to a composable enterprise is a big decision, and it's not right for every organization. Whether you're considering the business case for moving toward composability or are already started down that road, getting an outside perspective can make your work easier.

Fusion Alliance's expert composability teams bring decades of experience in software and application development, DevOps, and cloud strategy and optimization. We're leaders in containerization and composability, but our priority is delivering solutions and partnerships that meet our clients where they are and propel them on their best path to success.

Learn more about maintaining a composable enterprise >>

Ask us a question about composability >>

Schedule a Composability Jumpstart workshop to get started, get unstuck, or get to the finish line >>

Get more information about Fusion Alliance's cloud & technology practice >>