Strangler Pattern

From Monolith to Modern Composable Commerce

Elastic Path

Table of Contents:

The Strangler Pattern

Introduction:

From Monolith to Modern Composable Commerce

Martin Fowler coined the term “Strangler Fig Application”, which was subsequently shortened and popularized as Strangler Pattern. The approach is inspired by the strangler figs of Australia who seed the upper branches of a tree and gradually work their way down to the soil. In a gradual approach they replace the old tree killing it off.

This metaphor is a great description of doing an evolutionary rewrite of an entire system, keeping it working while you make gradual changes. Pushing in from the edges the old system is replaced piece by piece until nothing remains.

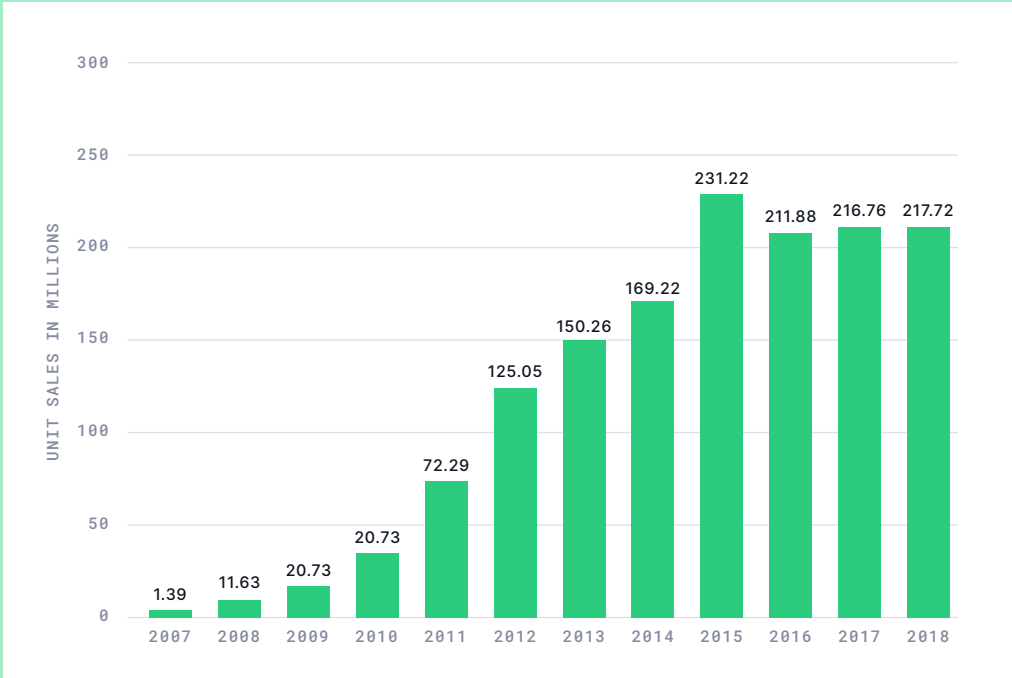
This deployment pattern has become very popular with the recent adoption of modular microservices based architecture. This guide describes different strategies for applying the strangler pattern when moving from a monolith to a modular approach.

Chapter 1:

Current Landscape

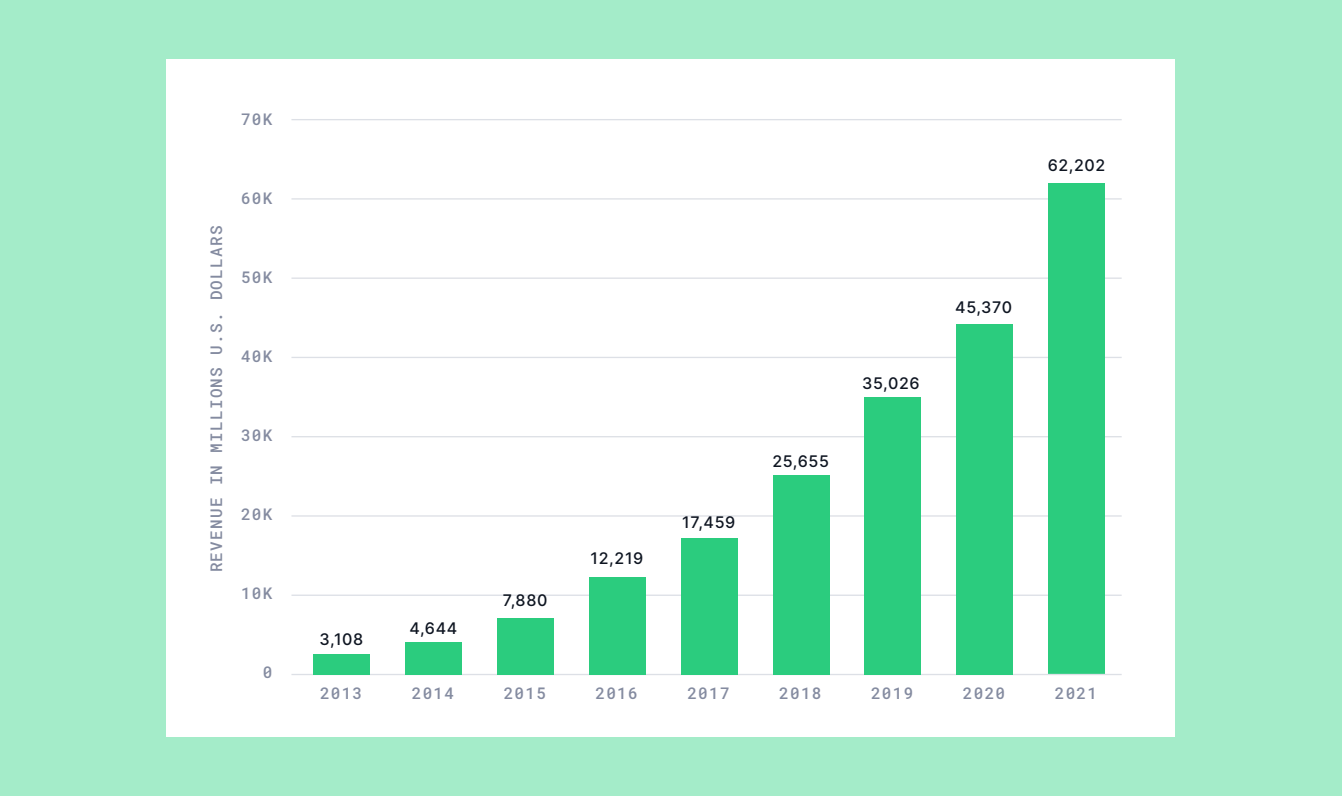
Since the dotcom boom, ATG, Oracle, Hybris, SAP, and IBM have been the major eCommerce suites offering an all-in-one approach to eCommerce. These companies have built massive complex applications with the goal of covering as many possible features as possible. These suites lasted for over a decade before massive technology shifts completely changed the base requirements making them obsolete.

The Apple iPhone ushered in a complete paradigm shift where mobile devices became the central connection between users and online content. These desktop focused solutions were ill equipped to deal with mobile devices. Companies were forced to adopt separate mobile-centric solutions creating a disjointed experience for both employees and consumers. This issue has grown with the adoption of additional touch points including social media, voice, and IoT (internet of things).



SOURCE: HTTPS://WWW.STATISTA.COM/STATISTICS/276306/GLOBAL-APPLE-IPHONE-SALES-SINCE-FISCAL-YEAR-2007/

The introduction of Amazon's AWS transformed IT infrastructure. With the rise of DevOps, the ability to quickly deploy, manage, and horizontally scale containers changed the fundamental approach to building enterprise scale applications. With the decreased effort and cost, building smaller solutions became the de facto standard, with mini or micro-services being the best solution.



SOURCE: HTTPS://WWW.STATISTA.COM/STATISTICS/276306/GLOBAL-APPLE-IPHONE-SALES-SINCE-FISCAL-YEAR-2007/

New software options emerged that tackled a specific requirement (e.g., search, loyalty, personalization) and provided a massive lift in functionality and business value. The revenue lift was so large it offset the cost of integration and the expense of paying for two solutions, the one built-in to the monolithic suite and the add-on dedicated service. These integrations, while still profitable, were expensive, time-consuming, and difficult to maintain.

These new market conditions created a new approach to commerce platforms, built from the ground-up to be integrated with best-of-breed solutions. These new solutions have been dominating the market in recent years. Meanwhile, large monolithic ecommerce suites are not equipped to handle these new requirements. Oracle, who acquired ATG in 2010, has announced its EOL (end of life). IBM sold its product to HCL which has been placed in maintenance mode. SAP Hybris has been steadily losing market share. The other legacy vendors are struggling to address this landscape shift with attempts to tack on APIs in the hopes of staying relevant.

Chapter 2:

Need for Composable Commerce

This new breed of ecommerce solutions follows a few core technology principals.

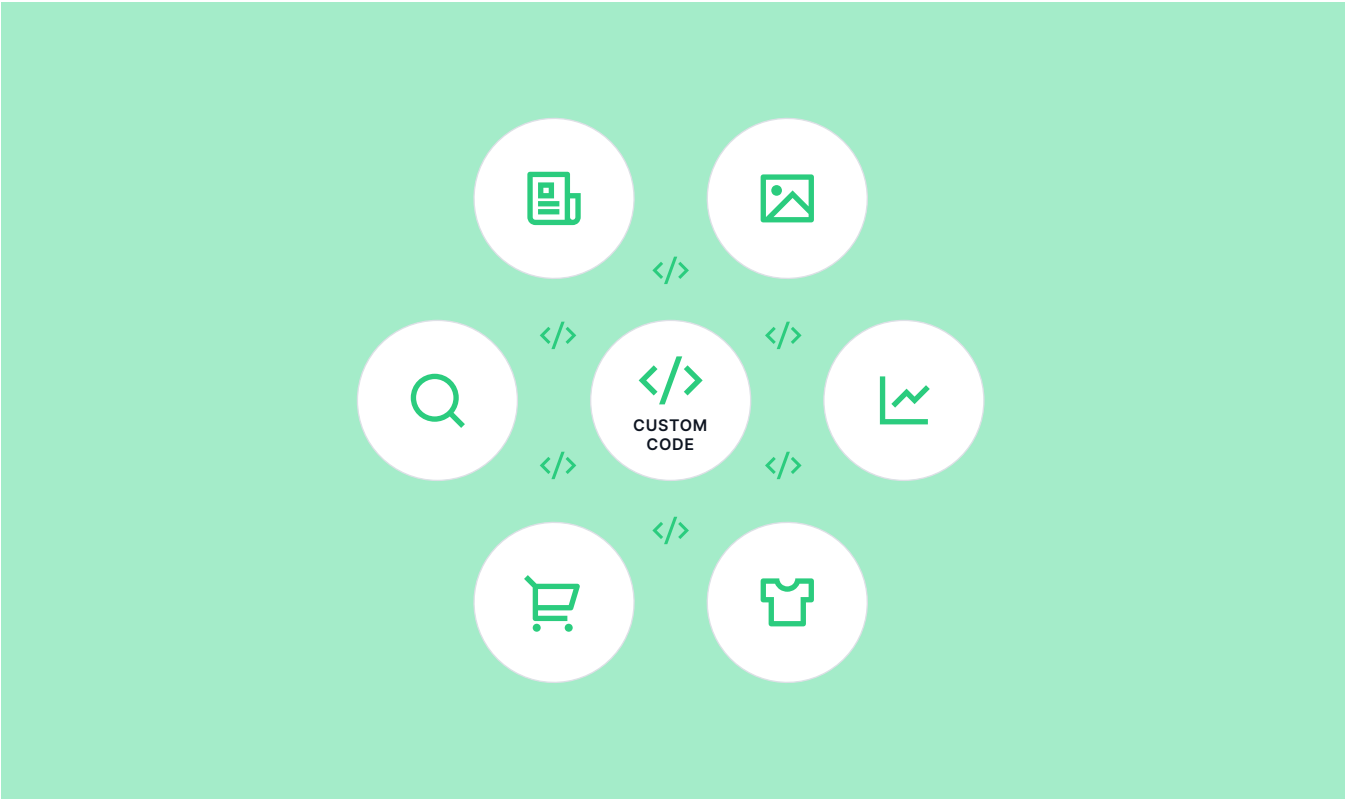
Microservices
Microservices provide multiple distinct benefits, performance, scaling, and modularization. Each microservice can be deployed separately and scaled both vertically and horizontally. This allows for quick updates and near infinite scale and impressive performance.

By breaking commerce into distinct business functions built by dedicated product teams, microservices allow components to be adopted individually. This ensures a best-of-breed solution as each piece can be replaced without the need to overcome vendor lock-in.

API-First (aka headless)
An API-First approach creates clean and simple integrations between multiple services. The underlying technology becomes inconsequential as the API becomes the single touchpoint. Patches can be continuously applied with no downtime or need for upgrades. Multiple customer touchpoints and distinct UIs can be created leveraging a single unified API backend.

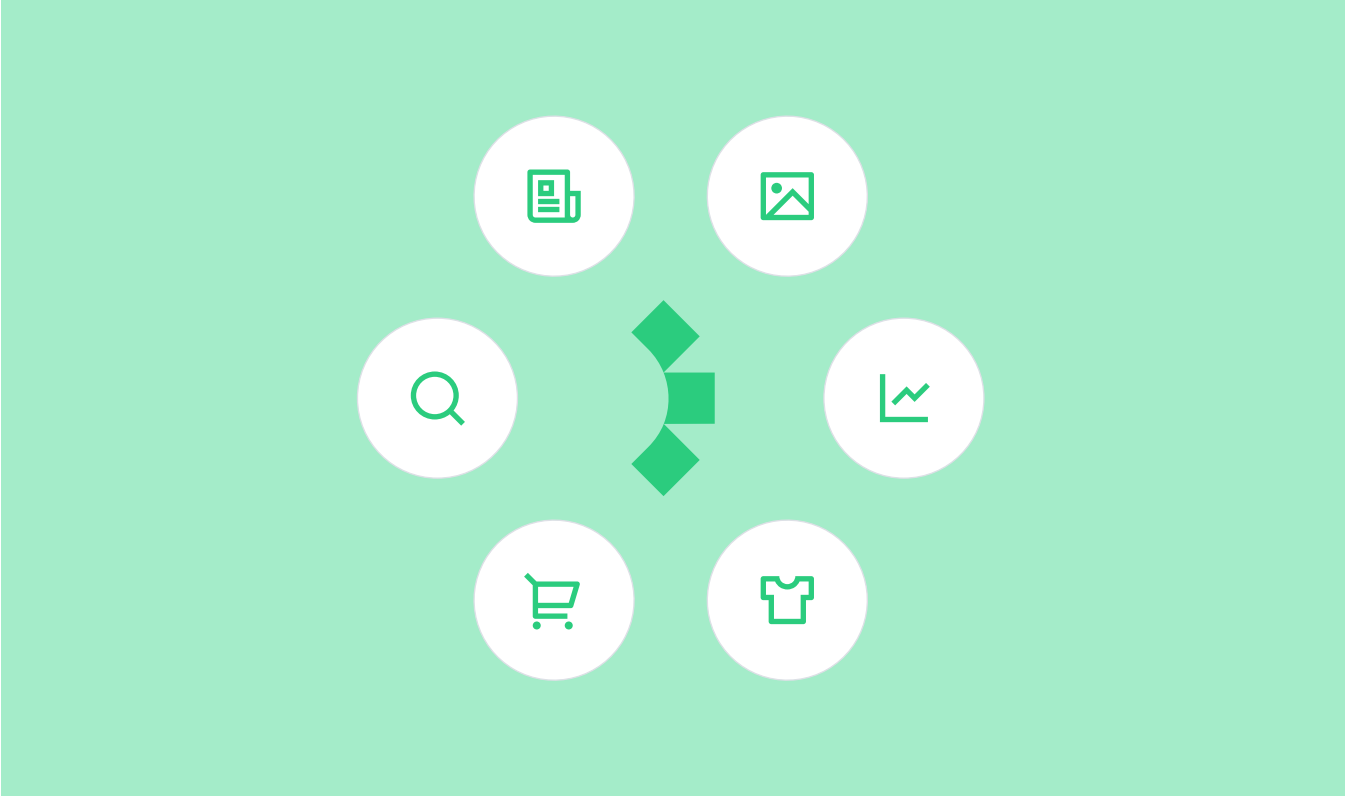
Cloud-Native
While any software can be lifted-and-shifted to the cloud, these solutions are built directly for cloud hosting. They adopt cloud-centric services for containerization, storage, and events. They forego traditional plugins and leverage the public cloud for extending business logic with the options to push messages directly to cloud queues or attach to webhooks. This approach has been coined as MACH technology. While MACH technology can enable a Composable Commerce approach, it only addresses the technical possibility. Composable Commerce has a focus on enabling businesses to compose individual packaged business capabilities to move toward a future-proof digital commerce experience.

So, while it's possible to combine multiple MACH technologies into a single offering, if done via custom integration code, or integrations via acquisition, companies are left with "MACH monoliths". While built from multiple components, replacing or adding new functionality requires a large technical project. Worse still, companies are left with no one to help should anything break or underperform.



In a true Composable Commerce approach, the role of a modern commerce platform has completely shifted, from offering specific features, to enabling the easy composition of best-of-breed offerings and supporting the final solution.

Elastic Path pioneered Composable Commerce by offering a MACH-based SaaS (Elastic Path Commerce Cloud), prebuilt integrations (Integrations Hub), and full support and experience assurance for the final composed solution (Composable Commerce XATM). This combination provides businesses the ability to construct their perfect solution through composable modular components and will full confidence in its long-term performance.



Gartner has fully endorsed Composable Commerce stating that “By 2026, the speed of digital innovation will improve by 60% relative to 2022, for organizations that have established mechanisms to reuse composable digital commerce modules.”

Chapter 3:

Business Case for Strangler Pattern

Leveraging the strangler pattern for migration costs more and takes additional development time. It requires more thought, planning, and modifications to the existing system for a successful migration. Even with this additional overhead, the benefits far outweigh the drawbacks.

Reduced Risk
Adopting the strangler pattern reduces the risk of a big-bang project. By deploying small modular pieces over time, each release is more agile, provides easier rollbacks, and has a smaller impact should things go wrong.

Releasing code consistently also ensures that development does not get stale as your team needs to work on additional priorities. This can prevent expensive rework on handling migrations or matching existing systems.

Realize Benefits Earlier
Companies choose to migrate to a modern API-based approach to realize the many benefits and positive impacts on the business. Whether it's improved conversion rates through site-speed or more engaging experiences, employee efficiency in managing operations, or the ability to react to a dynamic changing market, these benefits can have significant impact on total revenue.

A typical headless commerce implementation takes 12-18 months for completion. Elastic Path, the pioneers of Composable Commerce, have successfully reduced this implementation time to 3-6 months. Even with that decreased development time, a big bang deployment can mean waiting half a year to see an impact on revenue.

Adopting a strangler pattern allows the most impactful pieces to be developed and deployed first. In this way companies see improvements in weeks instead of months.

It's imperative that the business ensure the project plan prioritize high-impact items. Development firms may opt for a cart-first approach as it's simple to get started, but the most impact is typically seen at the beginning of the eCommerce funnel. A revamp of products and product discovery will influence more customers, improve operations, and streamline the rest of the project.

Chapter 4:

Strangler Approach

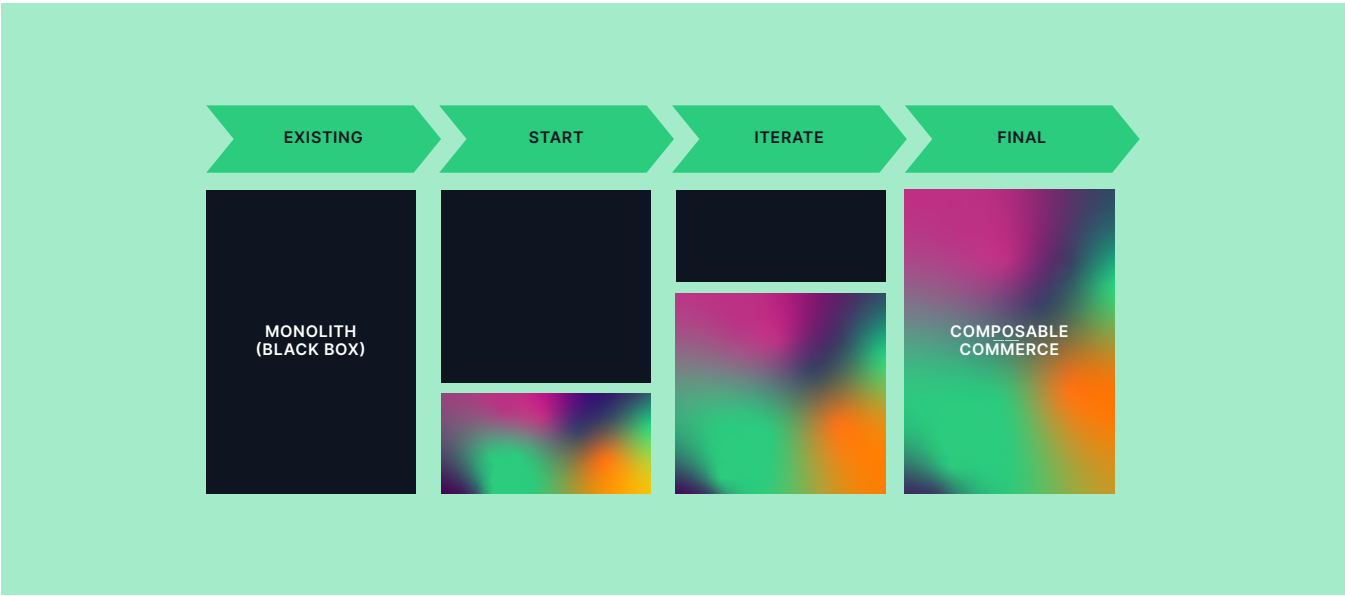
Mapping the Monolith Features To Best-in-Breed Solutions
Replacing a legacy monolith with new modular best-in-breed offerings creates the potential for a more robust, feature-complete, and adaptable system. This modular approach covers all the functionality of older monoliths while ensuring key areas of business focus are identified and optimized.

The new API-economy allows for a wide selection of cloud services providing great building blocks for the final design. For unique business features, custom microservices are developed or SaaS offerings are extended for the use-case.

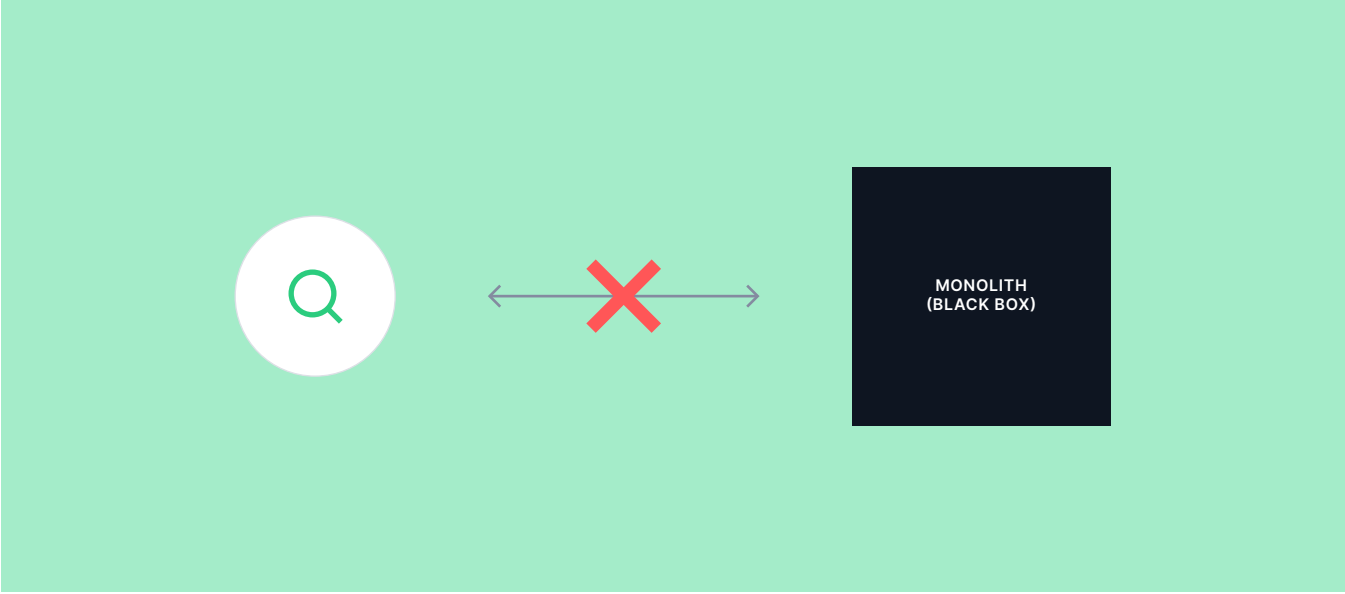
Common items to include:

- Search & Navigation: Algolia, Constructor.io, and Elasticsearch provide robust enterprise-class functionality.
- CMS and DXP: Consider an API-First headless CMS, these include Amplience, Contentful, and Contentstack.
- Storefront: Ensure a modern JavaScript based front-end. A progressive web app (PWA), JAMStack, or transitional app provides the best foundation for eCommerce.
- Composable Commerce: While there are many headless and MACH options available, choosing a Composable Commerce vendor should also include pre-built integrations and full solution support.

With the modular components identified, you can determine which areas will provide the most business value and how to begin the migration.



A traditional monolithic application is developed and packaged in a single program that runs on a standard physical or cloud server. While the monolith contains many different business functions and varied business logic, this is compiled into a final single application, and the distinction between subdomains may be clear or intermingled in the original codebase. Each function may also have dependencies that are often shared and increase as the monolith grows. Any monolith features that are accessible via a network will use protocols, message formats, and API interfaces that are proprietary and built to only interact with the monolith, as such they are difficult to leverage from new client applications. For example, the monolith may leverage a SQL database, but new applications calling the database directly will destabilize the monolith.



It's important to determine if new systems can be integrated with the existing monolith. This can depend on a variety of factors.

- Does the company have developers experienced with the legacy code who are confident in making structural changes?
- Does the monolith allow for code changes, will those changes significantly impact the project timeline and release cycle?
- Does the monolith provide points of integrations including REST APIs where required?

If the monolith can not be adjusted, then the common approach is to wrap the monolith with a proxy or façade to insulate changes to the existing system. This wrapping can be seen as an API gateway for accessing the old system with the new approach, as well as ensuring new system use a modern approach when accessing the legacy system.

Strangling Scenarios
Strangling a complex monolith requires extensive planning with an experienced architect, setting up a microservices foundation for your development team, and determining which functionality should be replaced in which order.

Starting small will ensure the groundwork has been covered while minimizing risk in the initial release. This first project will be focused on building the infrastructure and best practices around a composable microservices design.

When determining the order of functionality, multiple approaches can be leveraged depending on the needs of the business.

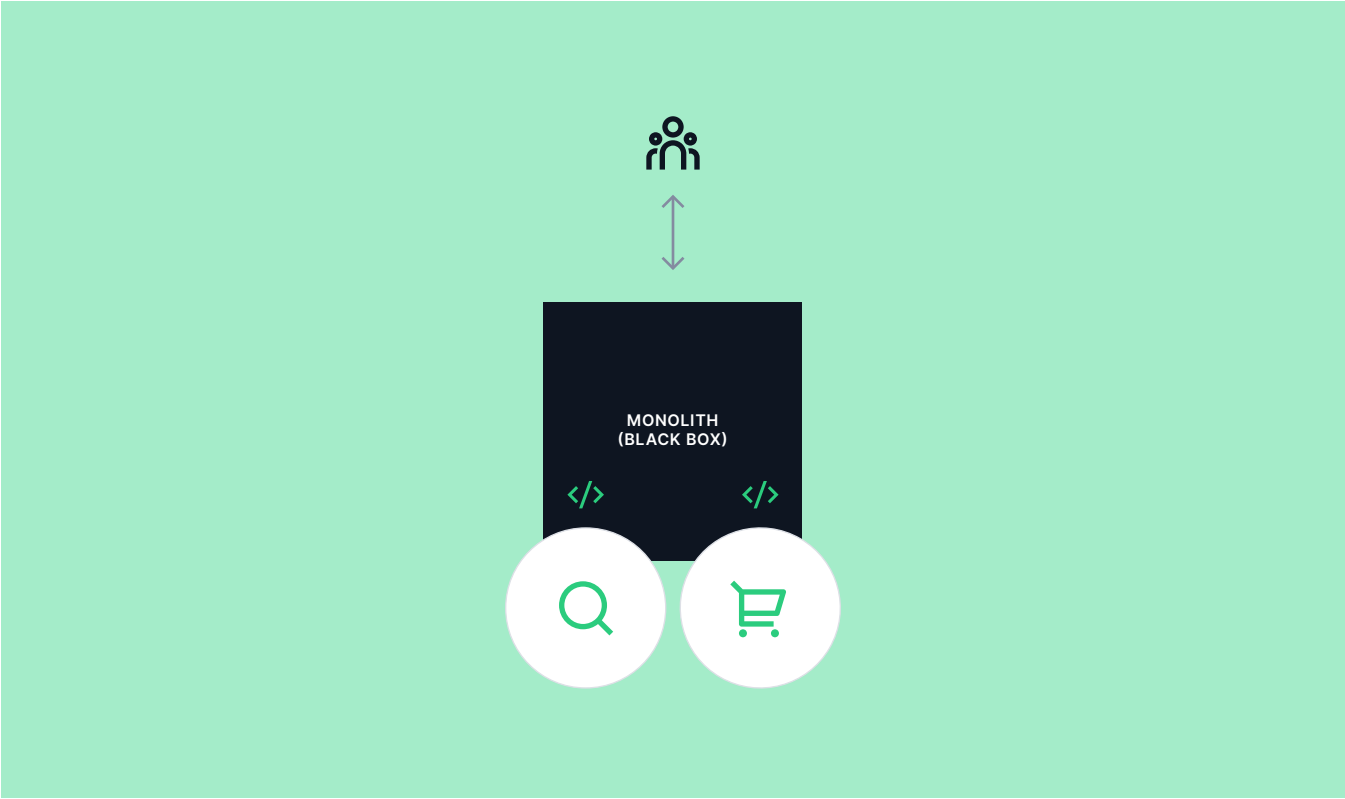
Pain points of the current monolith may need immediate attention. If the system is regularly crashing, throwing errors, or delivering a slow response time those items should be identified and replaced. This typically starts at the product listing pages and product display pages as they handle most of the traffic. If there are process issues, for example fulfillment or managing data, you may need to address those areas first.

New functionality should be considered if the current monolith is performing within expectation, but new features are desired to spur company growth. This may include adding new personalization or integrating a more advanced promotions system.

Strangling Approaches
The approach will be dictated by the current system capabilities and the ability to modify them. If a system must remain untouched the options become very limited.
The first approach is to integrate the new or replaced functionality directly into the existing monolith. This may mean replacing the product discovery functionality with Algolia's advanced search or moving the cart to leverage Elastic Path.

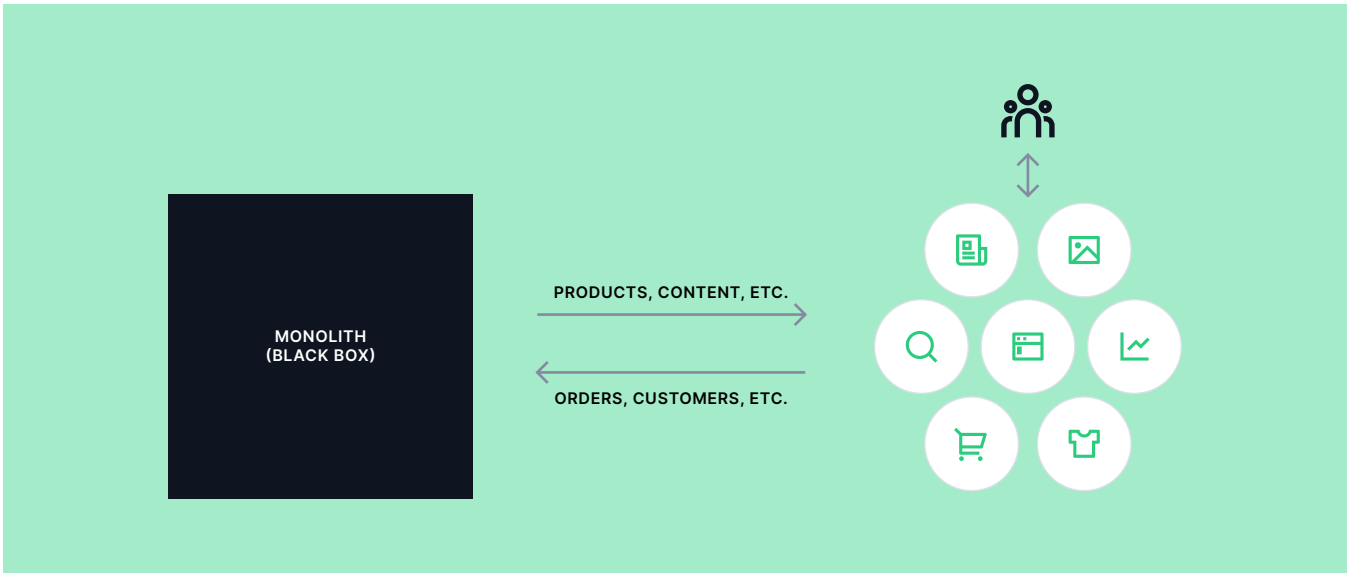
Modern microservice based systems are designed to be used as individual pieces and make this possible. For example, an item can be added to the Elastic Path cart, even if it does not exist within the Elastic Path catalog.

This approach can add value quickly to the business but generates the most code that will ultimately be discarded. It should only be leveraged when looking to add new functionality instead of fixing pain points.

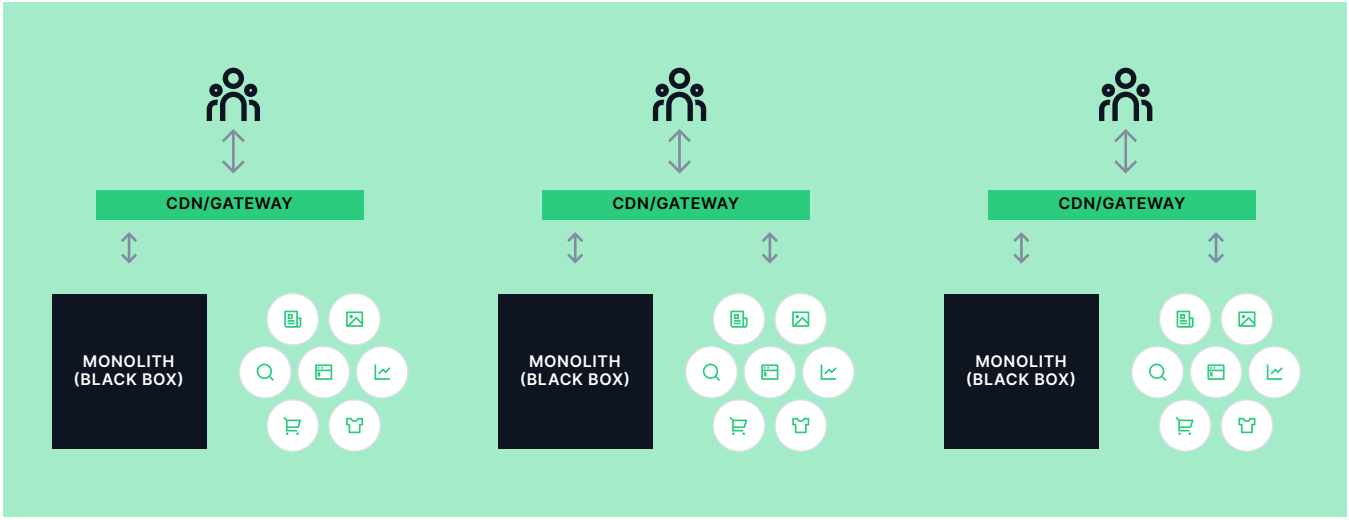


The next approach is to move the existing system into a supporting role. In this scenario the existing system can remain the source of truth for most of the data and will not disrupt the business functions until the project is closer to completion. Here, the data is synced between the existing monolith and new Composable Commerce solution.

The monolith maintains its integrations, but the new system serves customers ensuring they receive the new functionality with the goal of increasing conversion rates and customer satisfaction.



This approach requires that the existing monolith have proper API endpoints for syncing data. It does allow the use of accelerators or reference solutions to speed development time. Any pain points around performance will be alleviated with this solution.
The final approach is to wrap both systems with an API gateway that will serve to route traffic accordingly. As an example, the homepage and product browse path can be directed to the new solution, while cart and account functionality remains with the monolith. Over time each site section will be transferred from the monolith to the new system.



This approach gives extensive control over the transition and provides the easiest test and roll-back to ensure success. This works best with a website presence and mobile applications, other UI touchpoints need to be handled with one of the other two approaches.

Chapter 5:

About Elastic Path

Elastic Path is the company powering mission-critical digital commerce for the world's leading brands, such as Intuit, Pella, Deckers Brands, T-Mobile, and over 250 other leading brands. As relentless innovators, Elastic Path pioneered the Headless Commerce space in 2011 and spearheaded Composable Commerce in 2020. Elastic Path provides industry-leading headless commerce solutions for digitally-driven brands to rapidly build, deploy, and continuously optimize highly differentiated commerce experiences. Elastic Path is a global company with offices in Boston, Newcastle, Reading, Toronto, and Vancouver.