ICREON

Ebook

# THE FUTURE OF SCALABILITY

YOUR GUIDE TO TRANSITIONING FROM MONOLITH TO COMPOSABLE STACKS

# CONTENTS

# BREAK FREE FROM MONOLITHS

EMBRACE THE FUTURE WITH COMPOSABLE CMS INNOVATION

## The Hidden Costs of Traditional CMS: How They Shackle Your Business

Enterprises running on monolith CMS systems encounter a host of challenges that hinder their digital growth and agility. One of the most significant hurdles is the lack of flexibility. Monolith CMS architectures often bundle various functionalities into a single, unwieldy structure, making it challenging to update, scale, or innovate swiftly. This rigidity extends to development teams struggling with collaboration, as changes in one aspect of the system can disrupt the entire application.

Moreover, as enterprise needs evolve, monolith CMS systems become harder to adapt and integrate with modern technologies. They tend to create bottlenecks, leading to slower time-to-market for new features or applications. Maintenance also becomes complex, as any update can inadvertently impact other parts of the monolith, risking system stability.

In the rapidly changing digital landscape, enterprises require agility to stay competitive. Monolith CMS limitations hinder that agility, resulting in missed opportunities, frustrated development teams, and slower innovation cycles.

**52% of business leaders express confidence that using headless CMS would enhance their website performance.[1]**

Transitioning to a composable architecture addresses these challenges, unlocking the potential for modular development, rapid iteration, and seamless integration across diverse technology stacks.

# Monolith Suites vs Composable Stacks: Unraveling the Battle of Software Architecture

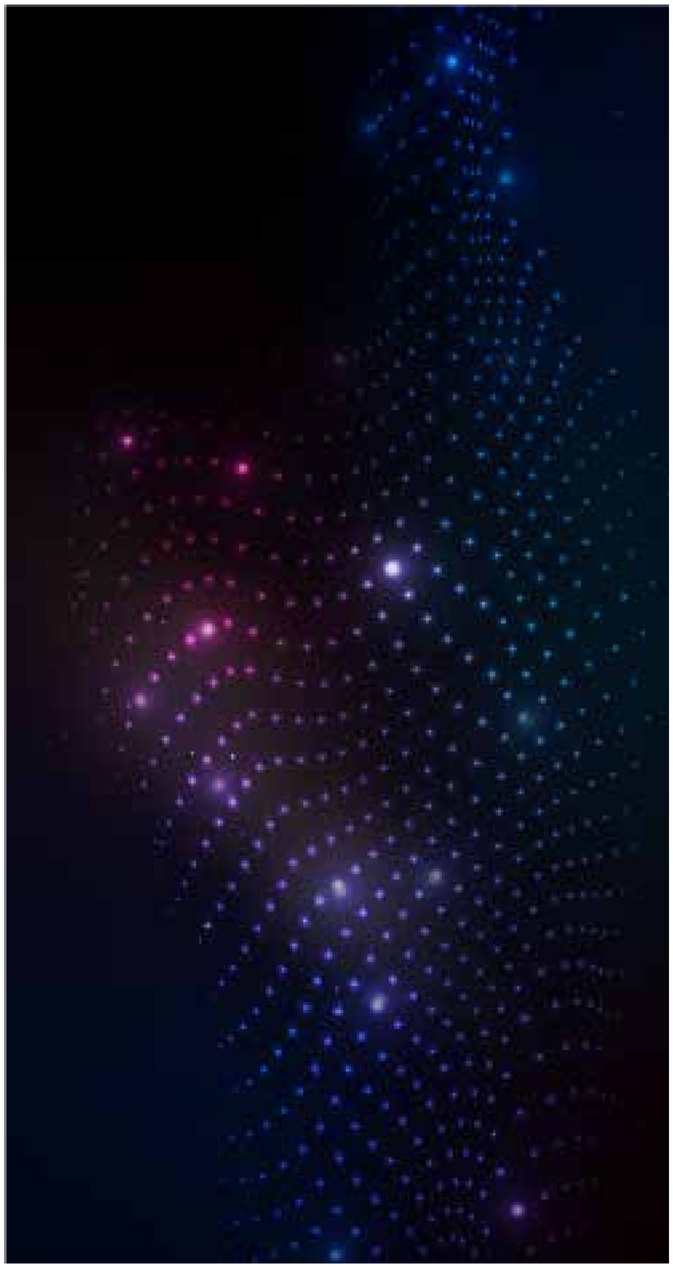
Software architecture is a critical decision that can shape the success and scalability of any application. Traditionally, monolith suites have been the go-to choice, offering a unified codebase encompassing all components. However, as applications have become more complex and the demand for flexibility and scalability has grown, composable stacks have emerged as a compelling alternative.

Let's delve deeper into the concepts of suites and stacks to gain a better understanding.

**Monolith Suites** refers to a software architecture pattern where an entire application's functionalities are tightly integrated and contained within a single, large codebase. In this approach, all components, modules, and services of the application are interconnected and run as a single unit. This type of architecture is called a "monolith" due to its unified and self-contained nature.

**Composable stacks**, also known as composable architectures or composable systems, are a modern software development approach that emphasizes modularity, flexibility, and scalability. In a composable stack, applications are built as a collection of loosely coupled microservices, each responsible for specific functionalities.

These microservices communicate through well-defined APIs, allowing them to work together as a cohesive system.



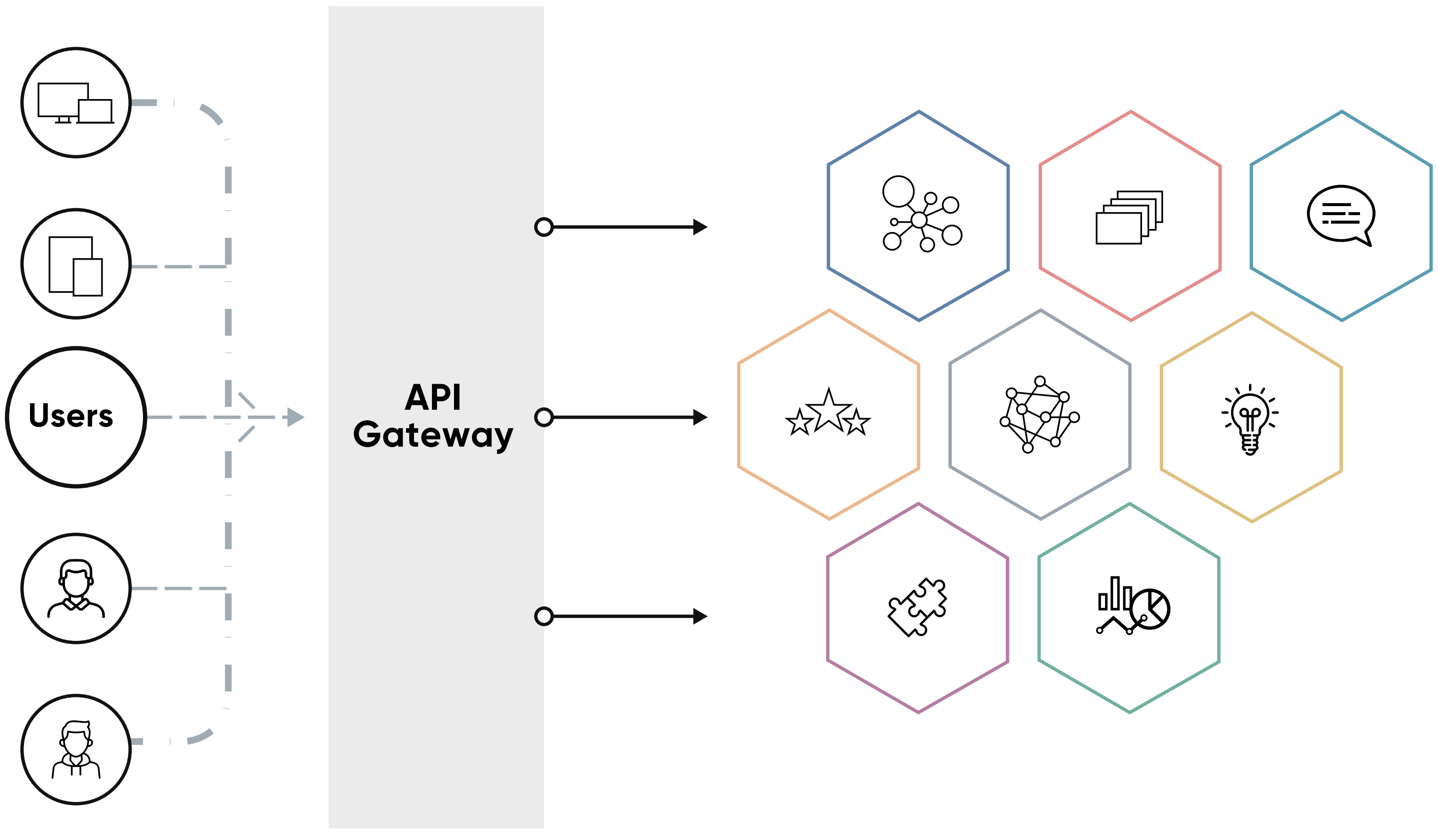
Monolith suites boast the advantage of simplified testing and shared state, making them easier to manage for smaller projects. Yet, as the application expands, maintaining and testing a monolith suite can become a daunting task, with a single change potentially requiring extensive testing.

| | Monolith Suites | Composable Stacks |
|---|---|---|
| Easy to understand & develop | ✓ | ✗ |
| Faster development | ✗ | ✓ |
| Flexible & scalable | ✗ | ✓ |
| Omnichannel support | ✗ | ✓ |
| Ease of testing | ✓ | ✗ |
| MACH architecture | ✗ | ✓ |
| Ease of technology integration | ✓ | ✓ |
| Ease of maintenance & upgrades | ✗ | ✓ |

The global CMS market was valued at around **$36 billion** in 2018 and is expected to generate around **$123.5 billion** by 2026, which is a very healthy compound annual growth rate (CAGR) of nearly 17 percent.

- Contentstack

# Benefits of Composable Architecture



A composable architecture, with its modular and flexible design, provides a robust foundation for building scalable and adaptable systems. This versatility not only enhances efficiency and cost-effectiveness but also empowers organizations to stay ahead in the ever-evolving landscape of technology and innovation. Let's explore how?

1. Composable stacks offer the promise of independent testing, better isolation, and technology flexibility. Each service within the stack can be tested separately, enabling faster iterations and pinpointing the source of issues.
2. By breaking down complex systems into independent and reusable modules, it allows for increased flexibility and agility. Developers can modify and update specific modules without impacting the entire system, enabling faster adaptation to changing requirements.
3. Additionally, modular architecture promotes scalability and optimal performance by allowing individual components to be scaled independently, ensuring efficient resource utilization.
4. The reusability of modules saves development time and enhances maintainability, as changes and updates are isolated to specific modules.
5. Collaboration among teams is facilitated, as parallel development becomes possible, and integration with external components is simplified.

Moreover, modular architecture improves testability, quality assurance, and overall system stability, making it an invaluable approach in building robust and adaptable software systems. However, the architecture understanding, development, 3rd party integration complexity, and the challenges of managing a distributed system introduce new testing considerations. To overcome these challenges, you can go through free composable blueprint on unlocking the true potential of composability.

# Navigating Migration: Choosing the Right Strategy to Migrate from Monolith to Headless CMS

Before you kick-start the migration process, looking for different migration strategies that may align with your end goals is imperative.

### *INCREMENTAL APPROACH*

An incremental approach to migration involves breaking down the process of moving from a legacy system to a composable system into smaller, manageable steps. Instead of attempting a full-scale replacement all at once, an incremental approach focuses on migrating or replacing specific components, features, or modules in a step-by-step manner.

For instance, **Strangler application** pattern is an incremental approach where you build and deploy the individual applications for each function that you've mapped and grouped. This approach allows you to deliver consistent UX during migration by creating a link between old and new CMS via a router. If the user requests a function that is not yet migrated, the router will direct it to the old monolith system and if the function is successfully migrated to the new CMS, the router will direct it to the new one.

This method allows for a more controlled and less disruptive transition, reducing risks and ensuring that the existing functionalities are maintained as the migration progresses.

### *AGILE APPROACH*

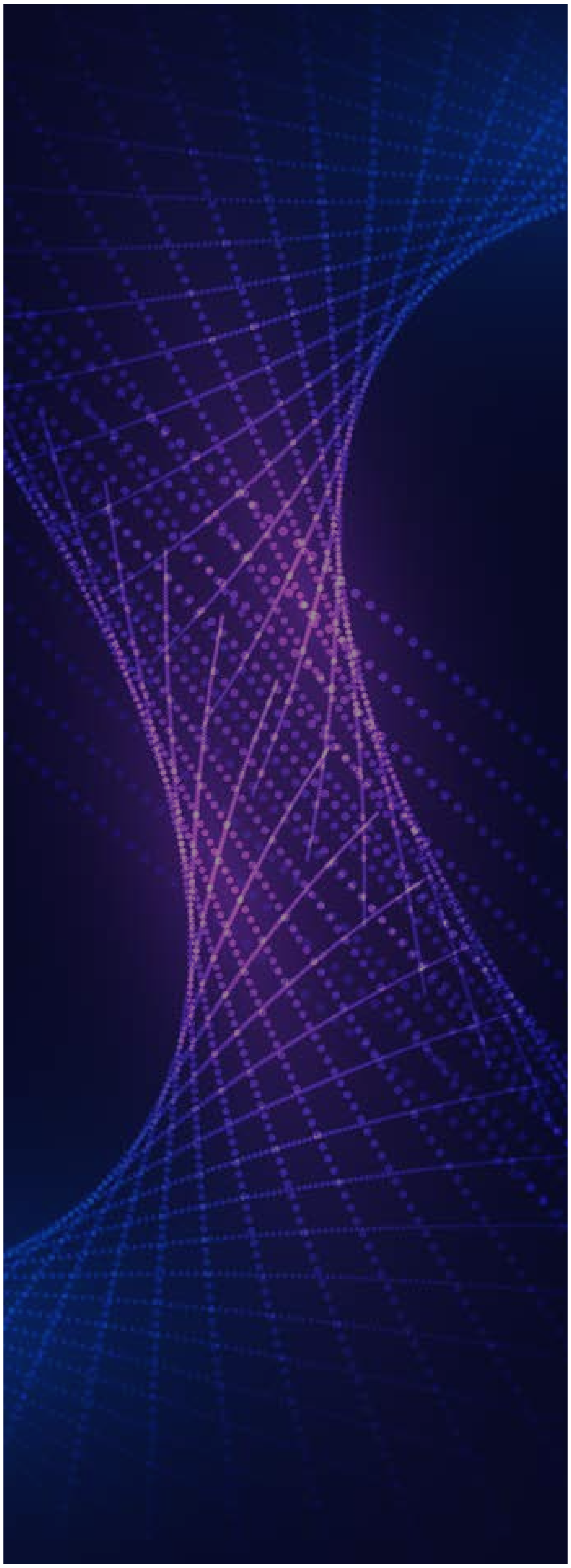
As the name suggests, this approach supports your ongoing and constant needs of adding new features and functionalities. For instance, if you're migrating from monolith to headless CMS and during the migration you bring in additional features to your system, you can simply add it to the standalone application instead of the whole monolith codebase.

For instance, the **DDD or Domain Driven Design approach** is a perfect example of an agile or flexible strategy for your migration from monolith to headless CMS. It emphasizes understanding the core domains of the business, enabling effective decomposition into microservices. This method ensures a seamless transition while aligning technology with business goals and enhancing flexibility in the new architecture.

## *PARALLEL APPROACH*

The parallel approach in migration is different from the above-mentioned approaches as it does not abandon the monolith CMS right away after migration. In fact, new composable components are developed alongside existing systems. This approach allows businesses to compare the performance of each feature and function migrated to the new CMS with the old one.

The major purpose of this approach is to test the success of the migration process, i.e., whether you've achieved your goals or not. Gradually, the new components take over, reducing risk and allowing for a controlled transition while ensuring uninterrupted operations and user experience.



## Points to Consider While Navigating the Path from Monolith Suites to Composable Stacks

Transitioning from a monolith suite to a composable stack can be a challenging but rewarding process. Composable stacks allow for greater flexibility, scalability, and agility in software development. When evaluating existing monolith suites for the transition, consider the following steps:

### *IDENTIFY GOALS*

Start by outlining your goals. Are there essential functions or features that require scaling to accommodate customer growth? Does your digital presence hinge on a specific tech stack incompatible with a traditional CMS? Is reducing development time and hastening your time to market a priority? Precisely identifying the business needs that this transition aims to fulfill is a crucial initial step.

If your existing monolith architecture is incapable of achieving all these goals, you're on the right track to migrate. Now, you need to make sure you've the right team of developers and testers who are well-versed in composable architecture transition and everything that occurs after.

### *MAPPING & GROUPING*

In every monolith architecture, multiple functions are weaved together, or each function depends on other functions. On the contrary, t is well-defined and has functions related to a certain process that are contained together in tech stacks. Therefore, it is important to map the functions in your existing monolith codebase and group them into logical units related to specific areas of your business.

For instance, you need to map and group all the functions of your marketing services into one unit, while functions supporting other areas should be grouped to their respective units. Doing this will help you perform migration easily.

### *START OFF WITH A PILOT TRANSFORMATION*

Once you've mapped and grouped functions into units, you need to start off with a pilot transformation that involves migrating a unit of the monolith into a microservice instead of migrating the whole system. Doing so will help you gain insights into challenges and 5 benefits of transitioning to a composable stack without disrupting the entire system.

### *DEPLOY A SMALL TEAM TOGETHER WITH AGILE METHODOLOGIE*

Next step is to assemble small, cross-functional team consisting of experienced developers, testers, DevOps engineers, and domain experts who understand the intricacies of the existing monolith and have a solid grasp of the target composable stack technologies. By deploying a small, agile team, organizations can quickly respond to challenges and adapt to the complexities of transitioning from monolith suites to composable stacks.

Make sure the team follows agile methodologies during pilot transformation that can accept or reject changes required at any time. Agile methodologies enable incremental progress and frequent feedback, leading to more successful outcomes and reduced risks during the migration process.

### *DEPLOY A SMALL TEAM TOGETHER WITH AGILE METHODOLOGIE*

Next step is to assemble small, cross-functional team consisting of experienced developers, testers, DevOps engineers, and domain experts who understand the intricacies of the existing monolith and have a solid grasp of the target composable stack technologies. By deploying a small, agile team, organizations can quickly respond to challenges and adapt to the complexities of transitioning from monolith suites to composable stacks.

Make sure the team follows agile methodologies during pilot transformation that can accept or reject changes required at any time. Agile methodologies enable incremental progress and frequent feedback, leading to more successful outcomes and reduced risks during the migration process.

### *SELECT THE RIGHT TECHNOLOGY STACK*

This is a crucial step as it decides the success of the technology transition. You need to choose the appropriate technologies and tools for the pilot project. Ensure that the selected technology stack aligns with the organization's long-term vision for the composable stack. However, avoid introducing unnecessary complexities during the pilot phase by keeping the technology choices manageable.

Consider factors like scalability, performance, security, team expertise, and budget constraints. Identify the key pain points in the existing monolith suite that need to be addressed with the new technology stack. Also, you need to look for monitoring and observability tools available for the chosen technology stack. Effective monitoring will help you track the performance, health, and logs of microservices.

### *REPEAT PROCESS UNTIL FINISHED*

Once the pilot transformation is successful, pick another unit and repeat the process until whole migration from monolith suites to composable stacks is completed.

# Overcoming Challenges with Best Migration Practices

The transition from monolith suites to composable stacks represents a fundamental shift in how organizations approach software development and system architecture. However, this transformation is not without its challenges, particularly in terms of cultural, organizational, security, and governance aspects. Below we will explore each of these and provide strategies to address these effectively.

## CULTURAL & ORGANIZATIONAL

### *RESISTANCE TO CHANGE*

Moving away from established monolith suites can be met with skepticism and fear of the unknown. To address this, it's crucial to foster a culture of continuous learning and improvement. Engage employees in open discussions about the benefits of composable stacks and how it aligns with the organization's long-term goals. Encourage feedback and provide training and workshops to help teams develop the necessary skills to work with the new technology effectively.

### *SILOED MINDSET*

In monolith suites, teams often work in silos, focusing on specific modules or components. Transitioning to composable stacks requires breaking down these silos and fostering cross-functional collaboration. Encourage regular team interactions, conduct joint planning sessions, and implement agile methodologies that promote communication and knowledge sharing across departments.

### *LEGACY SYSTEM INTEGRATION*

When transitioning to composable stacks, evaluating legacy system integration within monolith suites is pivotal. It necessitates a strategic approach to seamlessly incorporate existing infrastructure into the evolving ecosystem, optimizing both functionality and efficiency. These include reconciling disparate data formats, ensuring seamless communication between old and new components, and addressing compatibility issues.

### *RESOURCE ALLOCATION*

Resource allocation under Cultural & Organizational aspects is about aligning resources with an organization's values and strategic goals. It involves transparent, inclusive decision-making, smart financial planning, talent management, and technology investments. Adapting to change and measuring performance are key to fostering a culture that drives success.

## SECURITY AND GOVERNANCE

### *DATA VULNERABILITY*

Composable stacks often involve the exchange of data between various microservices and components. This increased communication introduces potential vulnerabilities, such as data breaches, unauthorized access, and data leaks. Implement robust authentication and authorization mechanisms to control data access strictly. Use encryption to protect sensitive data during transmission and storage.

### *API SECURITY*

Application Programming Interfaces (APIs) play a vital role in composable stacks, facilitating communication between different services. Poorly secured APIs can be exploited by malicious actors to gain unauthorized access to the system. Ensure that all APIs are properly secured with authentication, authorization, and rate limiting to prevent API abuse and potential denial-of-service (DoS) attacks.

### *CONTAINER SECURITY*

Containers are commonly used in composable stacks to package applications and their dependencies. However, they can be susceptible to security issues if not adequately protected. Use trusted container images, regularly update dependencies, and implement container isolation mechanisms like Docker security best practices and Kubernetes Pod Security Policies.

### *COMPLIANCE & REGULATORY REQUIREMENTS*

Transitioning to composable stacks doesn't exempt organizations from compliance and regulatory requirements. On the contrary, it may introduce additional complexities. Conduct a thorough assessment of the applicable regulations, such as GDPR, HIPAA, or PCI DSS, and ensure that your composable stack architecture complies with all necessary standards. Implement data anonymization and pseudonymization techniques where required to protect user privacy.

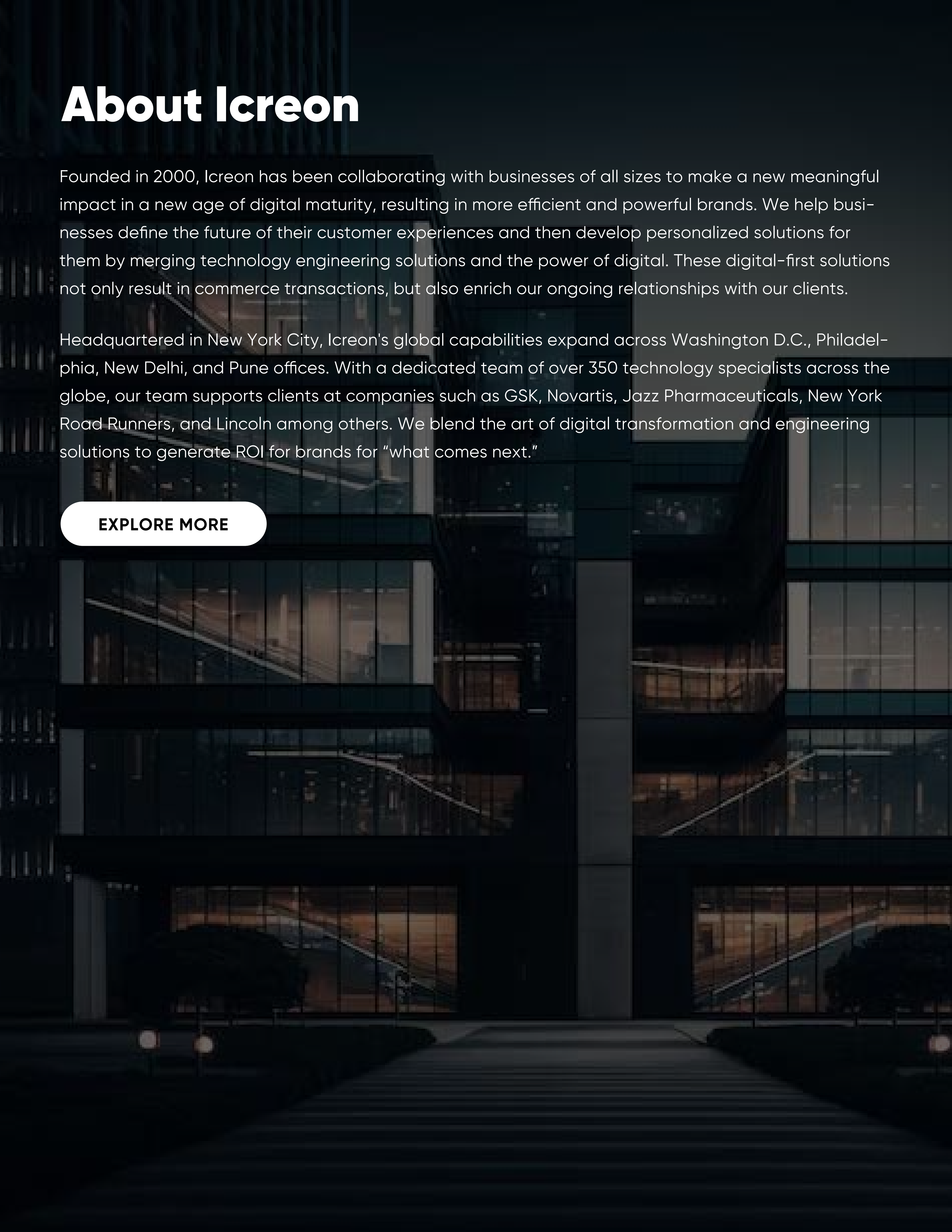
### *CHANGE MANAGEMENT & VERSION CONTROL*

Composable stacks encourage frequent updates and changes to individual services. Managing these changes effectively becomes critical to ensure stability and security. Implement a robust change management process, version control, and deployment strategies to track changes and roll back if necessary. Employ continuous integration and continuous deployment (CI/CD) pipelines to automate these processes and reduce human errors.

## Embark on Your Journey to Composable CMS Excellence Now!

In a world spinning faster than ever, bidding adieu to monolith suites and embracing composable stacks isn't just a change—it's a daring leap into the future of business. This transition holds the key to unlocking a realm of unparalleled flexibility, rocket-like scalability, and streamlined efficiency, lighting the way for innovation and a fierce competitive edge. Yet, we empathize with the thought of stepping into the unknown, where challenges and intricacies abound.

At Icreon, we're your trusted partner in this transformative journey. With our Composable CMS consulting, engineering, and development services, we offer you the guidance and expertise needed to navigate the complexities of composable stacks. Together, we can transform your digital landscape, elevate customer experiences, and drive growth. Talk to our composable experts to start your composable journey.



## About Icreon

Founded in 2000, Icreon has been collaborating with businesses of all sizes to make a new meaningful impact in a new age of digital maturity, resulting in more efficient and powerful brands. We help businesses define the future of their customer experiences and then develop personalized solutions for them by merging technology engineering solutions and the power of digital. These digital-first solutions not only result in commerce transactions, but also enrich our ongoing relationships with our clients.

Headquartered in New York City, Icreon's global capabilities expand across Washington D.C., Philadelphia, New Delhi, and Pune offices. With a dedicated team of over 350 technology specialists across the globe, our team supports clients at companies such as GSK, Novartis, Jazz Pharmaceuticals, New York Road Runners, and Lincoln among others. We blend the art of digital transformation and engineering solutions to generate ROI for brands for “what comes next.”

EXPLORE MORE