uniform

Composable CMS evaluation

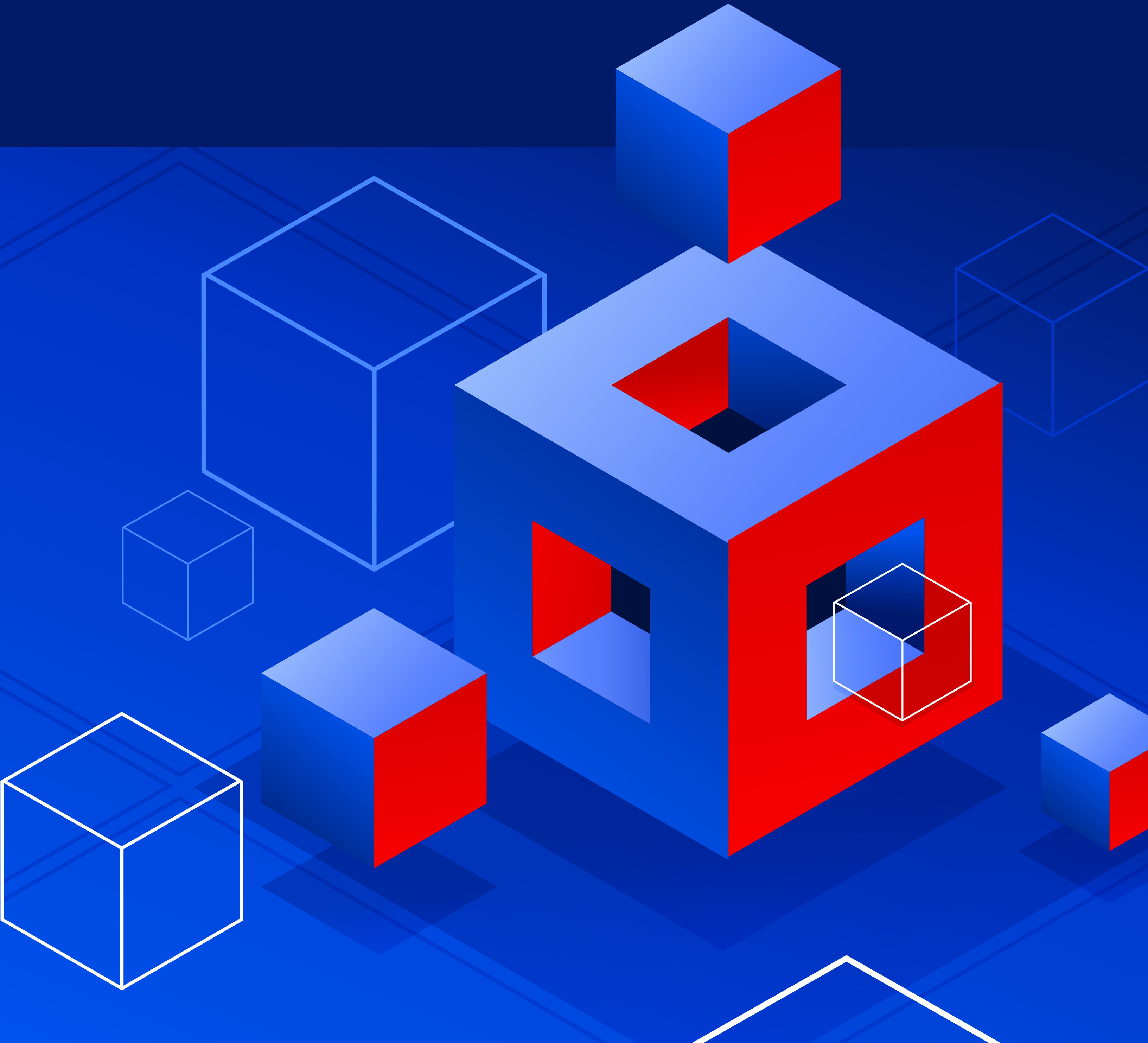


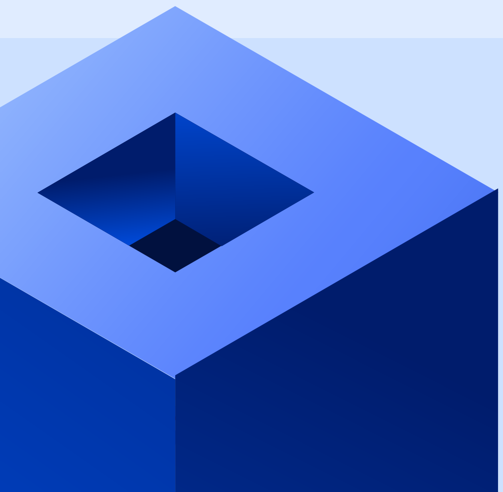
TABLE OF CONTENTS

Introduction

This guide explains the approaches and criteria associated with composable architectures, especially for organizations that are planning a migration. Fact is, even though those who have adopted composable all agree that it's the right thing to do, the process is complicated because, besides changing technologies, it requires a shift in how teams operate and interact.

For simplicity, this guide contains four sections with notes for specific task groups:

- Background analysis and planning, which is useful for the team in charge of planning: marketing, IT, a combined digital group, product owner, etc.
- Advice for meeting strategic goals, which, based on Uniform's extensive experience and CMS knowledge, delves into common success patterns and specifies the pitfalls to avoid while creating a tech stack in today's constantly evolving market.
- Vendor evaluation, which chronicles the major criteria for assessing vendor capabilities.
- Consideration for composable stacks, which, given that CMSes no longer own digital delivery end to end, recounts the categories and functions to take into account.



With a digital experience platform (DXP), you rely on one vendor for most capabilities, including complementary products at times. That's not foolproof, however, due to complex integrations and the need to justify the high expenditure for that larger platform.

An evaluation of composable solutions

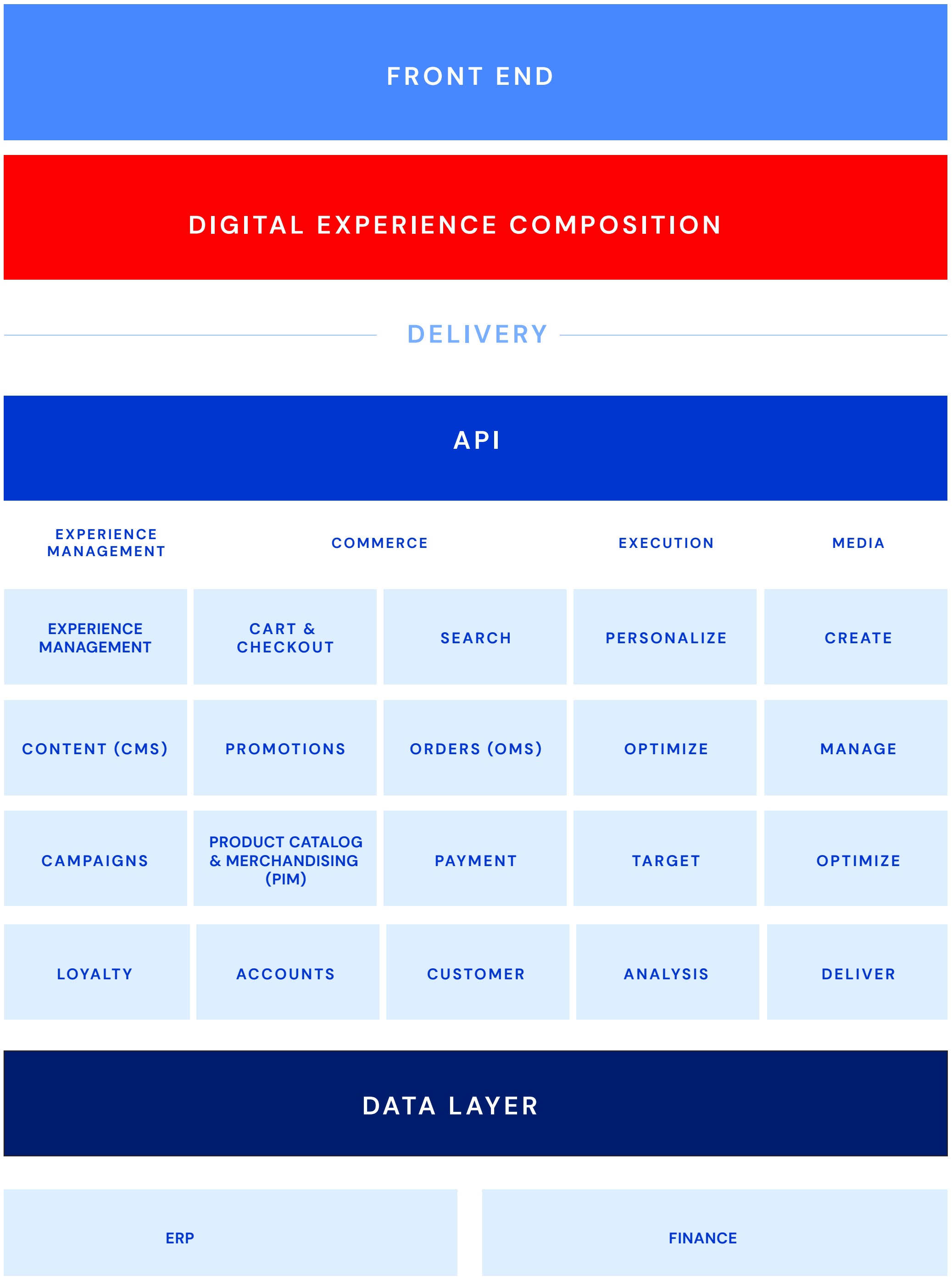
Rather than examining a single vendor's features, picking the right CMS for a composable architecture requires a familiarity with the following:

- The industry landscape and the additional functions and vendors you will need, as well as the way they will fit together.
- The various offers and trade-offs in pricing and cost, especially in case of a significant overlap among vendors.
- The SaaS delivery model, which often offers much faster try-before-you-buy iterations to understand not just the software, but also the speed at which you can onboard, develop, and integrate. As a rule, your PoC efforts move much faster and are more self-led than legacy DXPs.

A focus on architectural openness and flexibility would ultimately result in a more sustainable platform capable of withstanding rapid changes like new functionality, channels, or programming frameworks. The foundational investment and organizational revamp needed to support such a switch promises to serve you well for much longer, if not indefinitely. That's a tremendous transformation from traditional platforms, which proffered a fixed life and amortized cost.

Constantly chasing opportunities without adequately considering stakeholder needs might lead to a disconnect in requirements and frustrations when the new system launches. Similarly, concentrating on only the existing problems could result in a narrow vision, decimating motivations and fracturing efforts.

Not least, the emergence of MACH (Microservices based, API-first, Cloud-native SaaS, and Headless) technology has meant that the modern way of implementing a tech stack requires selecting best-of-need, MACH-oriented vendors for optimal agility and flexibility.



For all of MACH's amazing technological benefits, it can muddle procurement. Be sure to evaluate multiple vendors and observe how they work together. A case in point: For commerce, you could realistically start with at least six criteria:

- Commerce engine (cart and checkout, products, promotional campaigns)
- Search
- Content management
- Digital asset management
- Front-end framework
- Hosting

Before, you might have sought most of the functionality you desire from only one vendor, often based on features, and far less on the effectiveness of the solution's interactions with other systems. A major benefit of going composable is that, instead of launching a massive all-in-one, rip-and-replace migration fraught with risks and steep learning curves for your team, you can move in stages. An exception is that certain old systems lack the APIs to properly implement headlessness.

Key terms and definitions

Application programming interface (API)

A protocol for software components to communicate with one another, with which developers structure communications among platforms.

Composability

The strategy of building a more flexible tech stack with open and headless services and tools.

Content delivery network (CDN)

An origin server that stores and distributes content worldwide through a network of servers, accelerating the loading of websites.

Content management system (CMS)

Software with which creators generate website content.

Content model

The content structures, fields, and relationships that make up the digital experience of your website and other channels.

Digital experience composition platform (DXCP) Learn more

A system that orchestrates solutions through a composable architecture, resulting in fast, robust, and adaptable tools.

Digital experience platform (DXP)

Technologies that started with CMS capabilities but now comprise features for personalization, marketing automation, and other advanced capabilities.

Headless

Back-end content management that is separate from the front-end presentation layer.

MACH

A modern tech standard, coined by the MACH Alliance, with four prerequisites: Microservices based, API-first, Cloud-native SaaS, and Headless.

Rich media

Dynamic forms of digital advertising, such as moving images, video, 3D, AR, and VR.

Query language

A means of requesting APIs for content systems with advanced capabilities that support complex operations like filtering, joining, pagination, and granular calls.

Software development kit (SDK)

A package of programming tools that can be installed on a platform. In the context of CMSes, SDKs are language-dependent API layers offered by the vendor on top of its underlying CMS APIs, facilitating programming in a specific language.

Difference between this guide and RFP templates

Vendors often offer similar CMS RFPs. Search for "CMS RFP template" and you'll see many examples). This guide does not do that because of the different value proposition for composable vendors.

Take the common CMS RFP "Vendor five-year roadmap." Before the evolution of SaaS, that roadmap was key since product implementations often took at least six months; time to value, many years. Aligning a vendor's understanding with your stakeholder needs was paramount.

In SaaS, however, the equivalent due-diligence process is the ability to try out software yourself. Instead of rounds of RFP refinement, interviews, demo pitches, and educated guesses that things will "probably" meet your needs, you can test it. And, if another vendor can better meet your changing needs later on, it's far easier to switch. Accordingly, you must refocus your inquiries on composable.

Previously, the emphasis was on features because it was hard to add or integrate other features. Despite the fewer features per software solution in composable, you can evaluate, integrate, and extend the software more smoothly and rapidly, relying less and less on specific frameworks long term. Indeed, one of the first principles of composable is that no matter that languages and frameworks come and go, your underlying investment is your **approach** to composability and the resulting ability to readily swap out vendors and languages.

That's why this guide focuses less on describing features and much more on ensuring that you can aptly evaluate vendors based on composable's best practices. For the same reasons, antiquated CMS RFP questions on browser-based interfaces, vendor versioning, upgrades, and end-of-life policies no longer apply and are not table stakes anymore. Conversely, since the architecture is more important in composable, your RFP process must include—and lend more weight to—input from developers and system architects.

NOTE:
Unlike in traditional enterprise-technology sales, many MACH vendors adopt a land-and-expand philosophy for growing their business, greatly reducing the number of RFP cycles in which it participates. That scenario particularly holds true at the lower end of cost, where most customers are expected to directly evaluate the technology, especially with the try-before-you-buy option, which we strongly endorse.

Invariably, vendors would answer yes to all RFP criteria because, technically, software can do anything. What counts is how effectively and quickly you can capitalize on it to perform the tasks. You can now see that ease of use and development for yourself.

Background analysis and planning

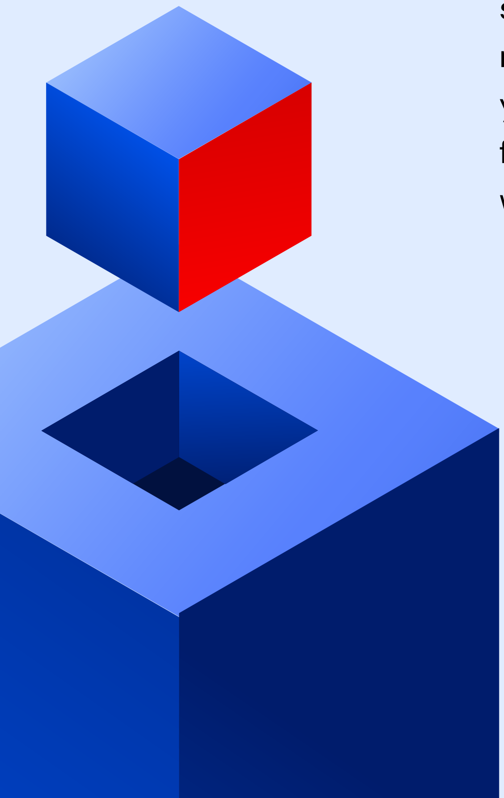
To ensure cross-team cooperation and allocate resources, management buy-in is key. The group that manages the overall project is responsible for the digital property, including the creation of a background analysis. You should involve as many stakeholders as feasible for endorsement and feedback, but remember that those who've been working with the old tools could contribute valuable input on the current content processes and strategy.

Key takeaways

Note these two basic rules:

- Successful project execution requires a clear understanding of your resources, stakeholders, and goals.
- Agility in evaluation does not mean no planning: It actually requires preplanning of goals while giving you optimal flexibility and freedom in experimenting with how to meet them.

For a foundation, conduct a deep analysis of your aspirations, stakeholders, requirements, and current capabilities. As mentioned, even while modernizing systems and technologies, you'll learn a lot about your current systems and processes. Thus, first aim at properly evaluating and leveraging what already exists wherever feasible.



Technology audit

An audit of your current technology must cover the systems, processes, content, as well as their pros and cons.

Inventory of systems and processes

By adopting a format similar to that previous slide on MACH architecture, examine your technology and consider the API-filled features that you might be able to harness. The necessary but API-lacking features are often priority candidates for replacement or augmentation through composable.

The common composable "first wins" tend to occur in the B2C space, particularly in social and payment providers. Firmly embedded systems are usually CMSes, product catalogs, and other content sources. The good news is that they're often the easiest to leave in place while still modernizing the process and other tools around them.

Content audit

A content audit would reveal the following:

- The size of—i.e., the number of items in—your content footprint.
- The value of your content: SEO, ability to generate inbound marketing traffic, etc.
- The effectiveness of the content in meeting the needs of external stakeholders.
- An insight into whether your content is well structured and suitable for meeting future channel requirements.
- The content sources with which you might need to integrate.

An audit of your current technology must cover the systems, processes, content, as well as their pros and cons.

The cost of doing nothing

Quantifying what happens if you **don't** modernize your tech stack is a worthwhile exercise. Here are the risks:

- You might be running on unsupported hardware or operating systems.
- You might incur additional costs, such as difficulties in hiring or retaining talent, from using outdated technologies.
- The longer you wait, the harder it might be to switch due to lack of skilled resources, support, or migration tools from the vendor.

Also, consider the likelihood and impact of the risks. Running on old and potentially insecure software might generate low yet high-impact risks. Difficulties in recruiting developers to code in legacy programming languages might be more likely but manageable.

In some cases, it makes sense to retain the old system but modernize the process. Such an approach, called a strangler pattern, is not new to enterprise software. What's new are the layers of digital experience composition that can more smoothly abstract—and often cache—those requests.

SaaS versus on-premise systems

Whether to opt for SaaS or on-premise has long been a crucial decision while selecting new technology. An on-premise (or hosted) system often owns the entire stack from creation to delivery. If you are removing certain aspects of your system (e.g., transitioning delivery to a cloud solution), you might need to determine how to access and work with the existing system through APIs.

In particular, answer these four questions:

1. Do you need to comply with firewall rules to grant access to SaaS systems?
2. Is the API's performance acceptable?
3. Does the existing system return data in a popular format, e.g., JSON instead of older standards like XML?
4. Do you need to go on-premise so as to comply with certain rules and regulations?

Audit of skills and roles

When examining your organizational skills, be sure to include those of your implementation partner, if any, who could be a valuable asset with a host of expertise, e.g., knowledge of your vertical or experience with applicable technologies or architectures. In addition to assisting with your front-end designs and CMS setup, a partner can often bring specific product expertise around understanding what technologies are used in your domain and common approaches for connecting those systems.

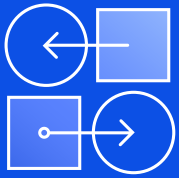
But of course, your relationship with your partner is key. If you are in a specialized field in which your partner is a domain expert, you're likely to have a closer working relationship and are potentially more inclined to defer to that partner's technical preferences and dependencies rather than relying on a technology vendor.

Be sure to map out the parts of the project to assign to your partner and specify the related role by answering these questions:

1. Will you consult your partner for major decisions?
2. Will you augment your team for some of the partner's assignments?
3. Will your partner own the end-to-end implementation?

Finally, project elements, such as evaluation, hand-off, and regular enhancements, might vary in SaaS. You might be working with actual, live software or nonproduction environments rather than developer instances and copies of the software.

Build versus run teams:



Given that digital transformation projects often require a massive amount of time, effort, and expenditure, it is extremely common—even among the most technologically savvy organizations—to do the initial build through vendors. However, for long-term development and maintenance, various models exist, ranging from a partner being continually involved and almost "owning" the architecture—to the exact opposite, whereby a partner hands off to the client, who continues to run the system.

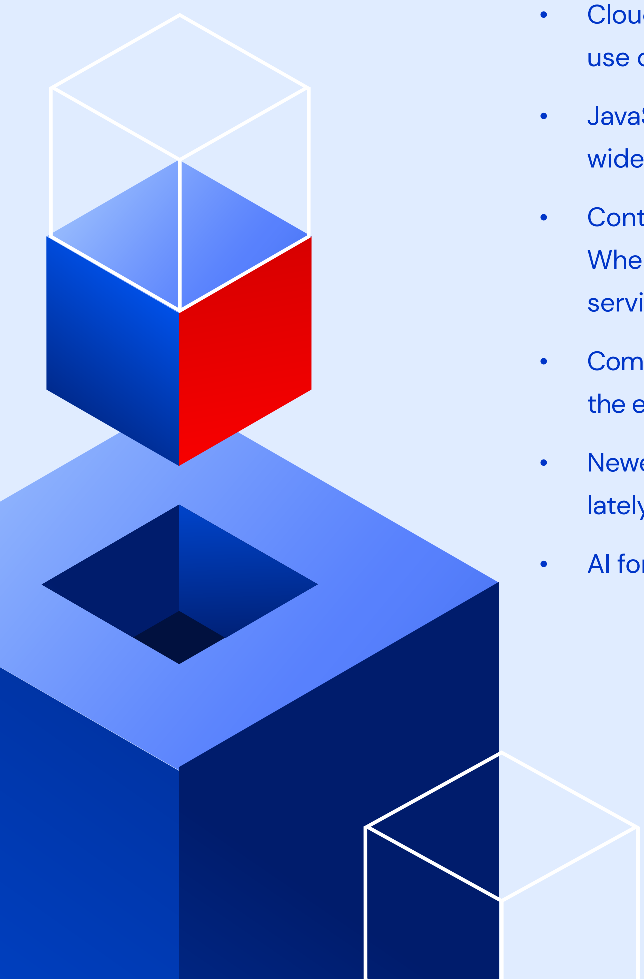
Advice for meeting strategic goals

Preferably, while embarking on a digital transformation project, you balance the pursuit of new opportunities (adding channels, customer segments, capabilities, etc.) with your current tooling's challenges stakeholders must grapple with. Constantly chasing opportunities without adequately considering stakeholder needs might lead to a disconnect in requirements and frustrations when the new system launches. Similarly, concentrating on only the existing problems could result in a narrow vision, decimating motivations and fracturing efforts.

New market opportunities

Thanks to the players described below, the last few years have seen tremendous advancements in the ways in which to deliver software and work with it in a composable fashion.

- Cloud providers for scalable SaaS solutions that can fulfill the needs of all use cases and industries, triggering a move from large, generic platforms.
- JavaScript-based frameworks and JavaScript Object Notation (JSON) for wide interoperability of native data.
- Content delivery networks (CDNs) that offer major performance gains. When was the last time you heard of a successful distributed denial-of-service (DDOS) attack on a modern platform?
- Computation for complex and dynamic tasks, such as personalization, on the edge or even on client hardware, e.g., a mobile device running your app.
- Newer channels in addition to the web: mobile, text, kiosks, social, and, lately, metaverse and augmented reality.
- AI for augmenting, categorizing, and generating content.



Organizational structure

A common failure in shifting to composable emerges from not adapting your team to work with the technology and to take advantage of new channels as well as accelerated campaigns and publishing.

Most large organizations boast teams tasked with the functions spelled out in the diagram below. Find out which functions and roles apply to your business and add them to the digital-transformation and technology-evaluation process.

FUNCTION	TITLES	TASKS
Discover	CXOs, enterprise architects, information architects, product owners	Gather requirements User journey mapping SEO/accessibility Content strategy, design and analysis Architecture audit/planning
Content design	Content designers, content engineers, content strategists	Content architecture Content modeling Taxonomies & ontologies Content governance Author workflows
Visual design	Head of UX, creative directors, visual designers	Visual & architecture design Component library Sitemap & wireframing
Create	Content teams, product managers	Content authoring & localization CMS input Content approval & workflows Experience assembly
Deploy	Information technology, DevOps	Content delivery Cloud infrastructure
Engineer	Product managers, engineers, front-end devs	Data integration Front-end coding
Analyze & optimize	Growth marketers, performance marketers	Content personalization A/B testing

Additionally, the Create functions might have the following stakeholders who're assigned an outsized role or forgotten during evaluation:

- **Skilled superusers**, who support other content creators and who require more robust tools for such tasks as bulk editing and scheduling.
- **Casual users**, who rarely work in the CMS but would serve as a good proxy for determining the intuitiveness of the system and its implementation.
- **Nonusers**, who might be content stakeholders relying on superusers, interacting with those folks outside the CMS, for example, by email.

External subject-matter experts: legal counsel, content-creation agencies, etc. Determine how to grant and revoke their access as warranted.

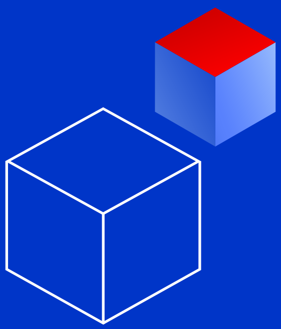
Most of all, ruthlessly prioritize the needs of those stakeholders. Since chances are that your first iterations will not completely satisfy them all, be vigilant about the implementation phases and update them regularly on the progress.

Heed stakeholder pains

A usual reason for updating enterprise software is to address ongoing user frustration. Identify, categorize, and prioritize the issues, ensuring that the new system satisfactorily and promptly removes the majority of the roadblocks.

Take note of what works

Avoid being caught up in problems and opportunities. Rather, in consultation with stakeholders, catalog what works well with your current system: customizations, features, and processes that appeal to your business users, who would feel lost without them. Vigorously managing changes here often makes for a successful rollout instead of a stress-filled one.



In principle, involve as many stakeholder teams as possible to secure buy-in and feedback on all aspects of the digital production cycle.

Common pitfalls

Avoid the four common pitfalls, as described below.

1. Missing out on an organizational change to an agile approach

Migrating to composable requires organizational change to support the new architecture and its goals for an agile approach that accelerates iterations and tests on the underlying systems, visual designs, and campaigns. Collaborative teams are a must for success.

Similarly, SaaS and composable platforms shift many responsibilities to the vendor. Hence, you and your partner must make many decisions for the architecture. If you previously worked with a CMS or DXP, your choices for hosting, programming language, design philosophy, etc. were made for you. Going composable means that you must understand and own those decisions.

2. Reimplementing old, ineffectual patterns and content models

A common mistake is simply copying your old content structure or, worse, embedding visual layouts into a content mode in the hope of speeding up the migration process. In fact, it does the opposite by codifying and compounding bad practices.

During replatforming, scrutinize your goals, content strategy, and team structure. A sound approach for minimizing change is through a DXCP, i.e., architect new content models in the CMS and combine them with the old content and models from your other platforms.

During replatforming, scrutinize your goals, content strategy, and team structure.

3. Planning for a big-bang migration

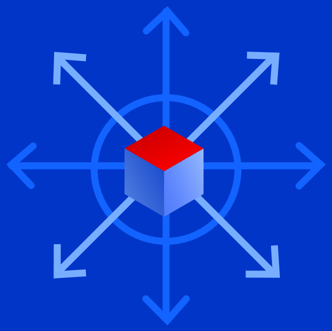
Oftentimes, failure occurs because you assume that you must implement composable projects in the same way in which you chose architectures before. That is, you would implement the new system, migrate the content, and retire the old system—usually as quickly as possible to avoid business disruptions—primarily because not only the software did not interoperate, but also you'd already licensed the entire package.

However, composable spells "pay as you go," i.e., you can implement it in small steps at lower cost. Given the goal of minimizing risk, changing systems in miniature increments makes a lot of sense, assuming that licensing of the underlying, to-be-retired systems costs only a minimal amount or less with reduced usage.

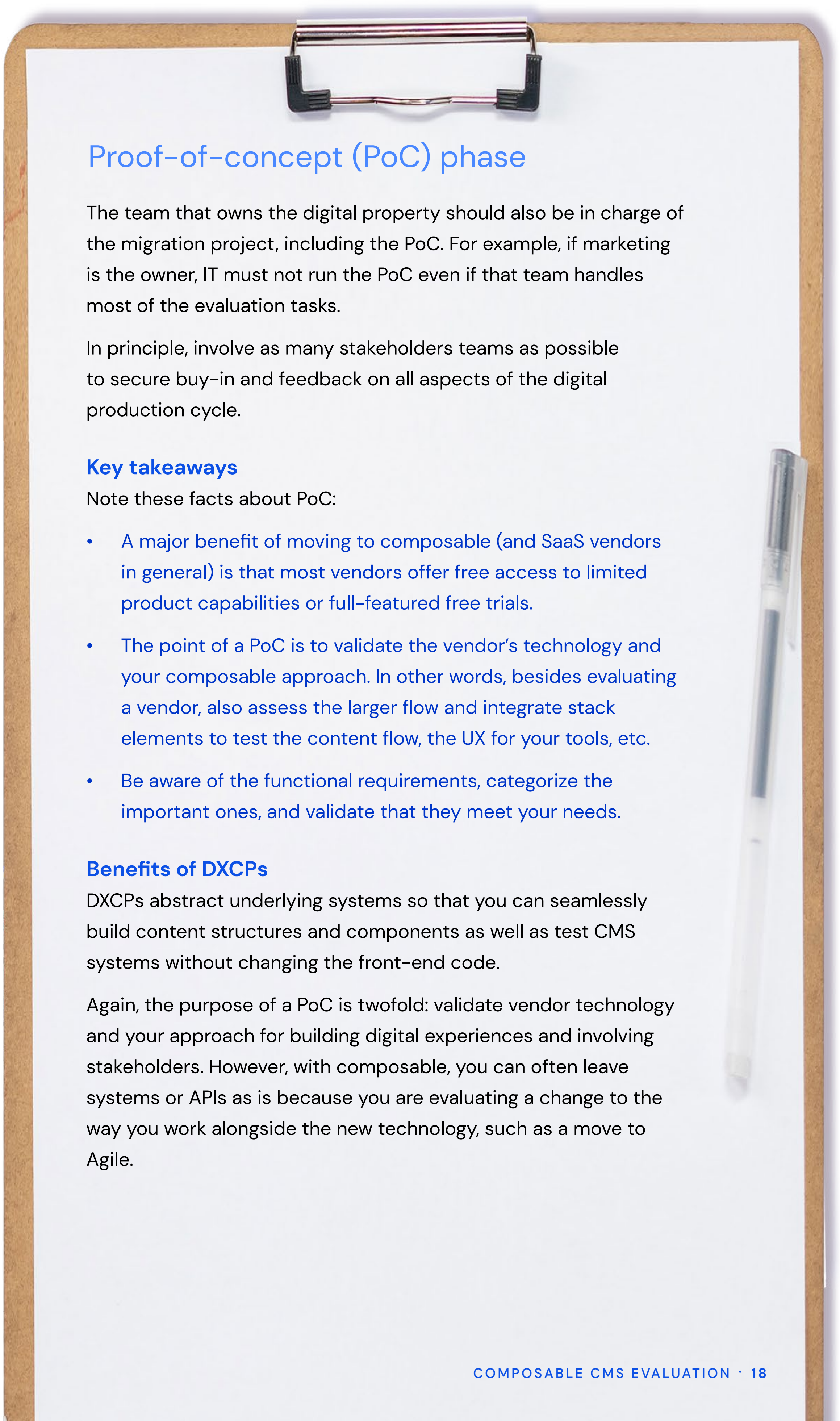
4. Losing sight of long-term sustainability and flexibility

Rather than treating it as an ongoing process, organizations often reimplement software as a one-and-done activity, plunging into other pitfalls as a result.

Composable encourages trying out new technologies or frameworks. That leads to a focus on long-term improvement and sustainability instead of the one-and-done process, which usually requires labor-intensive, painful migrations and disrupts the business. You can then always adapt to changing circumstances or technologies with negligible impact. Remember: The more you can do to keep your organization agile and your systems loosely coupled, the less the risk and cost for future projects.



Remember: The more you can do to keep your organization agile and your systems loosely coupled, the less the risk and cost for future projects.



Proof-of-concept (PoC) phase

The team that owns the digital property should also be in charge of the migration project, including the PoC. For example, if marketing is the owner, IT must not run the PoC even if that team handles most of the evaluation tasks.

In principle, involve as many stakeholders teams as possible to secure buy-in and feedback on all aspects of the digital production cycle.

Key takeaways

Note these facts about PoC:

- A major benefit of moving to composable (and SaaS vendors in general) is that most vendors offer free access to limited product capabilities or full-featured free trials.
- The point of a PoC is to validate the vendor's technology and your composable approach. In other words, besides evaluating a vendor, also assess the larger flow and integrate stack elements to test the content flow, the UX for your tools, etc.
- Be aware of the functional requirements, categorize the important ones, and validate that they meet your needs.

Benefits of DXCPs

DXCPs abstract underlying systems so that you can seamlessly build content structures and components as well as test CMS systems without changing the front-end code.

Again, the purpose of a PoC is twofold: validate vendor technology and your approach for building digital experiences and involving stakeholders. However, with composable, you can often leave systems or APIs as is because you are evaluating a change to the way you work alongside the new technology, such as a move to Agile.

Vendor evaluation

Once you've prioritized what you'd like to replace, including stakeholder needs and technological and resource constraints, list the composable technologies you want to add.

Remember that, since the goal is to minimize risk, updating systems in small increments is the way to go, assuming that licensing of the underlying, to-be-removed systems costs minimally or you can reduce their usage.

Of help is a digital experience composition layer with its SaaS-performance benefits, e.g., caching and CDN, on top of legacy APIs and abstractions against older APIs. As a result, you need not update the front-end development when ultimately swapping out content to a new system. After all, agile approaches are based on the concept of building a Minimum Viable Product (MVP) for validating key questions and iterating from there.

NOTE:

Be sure to evaluate the nonfunctional requirements that any solution must satisfy. That's a key but often overlooked step.

Non functional criteria

This section covers the nonfunctional requirements in detail.

Key takeaways

Intangibles are an important criterion for vendors. Even though, with composable, you can readily change vendors from a technical perspective, you must still invest heavily in training and connecting systems, as well as take into account the longer-term implications of working with a vendor.

Equally important are nonfunctional requirements that permeate the project, from evaluation to implementation to ongoing support.

Each criterion contains two categories:

Baseline capability	This is the minimum **commoditized** capability to **ideally** expect from a vendor. Absent this capability, critically evaluate if that's acceptable for your use case.
Advanced capabilities	These are examples of some—but not all—additional or **differentiating** capabilities. Carefully assess if you need them since you often pay more for them because they represent where the majority of differences in vendor capabilities lies.

Evaluation of onboarding and operational process

FUNCTION: DISCOVER CONTENT DESIGN ENGINEER

Find out how quickly you can start evaluating and doing business with a vendor. During the evaluation process, developers usually opt for trying out the software as quickly as possible. Most composable vendors do offer free trial accounts for limited functionality or time period, with some requiring a sales presentation or credit card at the start, or taking longer to respond.

Baseline capability	• The ability to open a trial account with minimal vendor intervention: demos, sales pitches, etc.
Advanced capabilities	• The ability to get a trial account immediately. • The ability to get a free account with no time limits for individual developers. • Onboarding material: training videos, walk-throughs, example code, etc.

Partner network

FUNCTION: DISCOVER CONTENT DESIGN ENGINEER

Understanding the depth and experience of your vendor's partner network is vital. Even if you already have partners, who might prefer a certain vendor, being familiar with the nuances of their skills—e.g., industry, application type, location, working cadence, agile versus waterfall across the partner network—helps in scoping vendors that can fulfill your architectural and domain expertise needs.

A major shift in a composable world in deference to agility is that the brand, not the partner, chooses the stack components to gain the competitive differentiation.

Baseline capability	Partners within your region with an understanding of the vendor's capabilities and experience in running a small number of projects.
Advanced capabilities	• Same as above, but with domain expertise specific to your business. • The ability to work within your team as agile resources. • Completion of a greater number of projects with multiple vendor tools for comparison and contrast during evaluation.

Geographical footprint

FUNCTION: DISCOVER

Examine the support model, including that from a partner network, and gauge the vendor's knowledge of the local laws and regulations. The best "functional" vendor that doesn't overlap with your development team or understand regional privacy rules would be suboptimal long term.

Baseline capability	Support in your preferred regions and time zones.
Advanced capabilities	• The regional infrastructure for complying with regulatory requirements, e.g., hosted in Europe for meeting German or EU laws. • The vendor's capabilities for regional compliance, e.g., GDPR data processing and retention.

Community

FUNCTION: DISCOVER CONTENT DESIGN **ENGINEER**

The developer and practitioner communities can be key information sources and starter code to assist not just your initial efforts, but also down the road with advanced concepts. So, find out the community size and available resources.

Baseline capability	The active community on social channels: GitHub, Discord, Slack, etc.
Advanced capabilities	• Sponsored community events, e.g., meetups and user groups. • Large amounts of open-source contributions to codebase or add-ons. • Tutorials in the form of blog posts, videos, etc.

Documentation

FUNCTION: CONTENT DESIGN CREATE ENGINEER

Browsing vendor documentation is an often-overlooked lifehack for software evaluation. Documentation lends insight into how the vendor works by answering questions like these:

- Does the vendor release products without documentation? If so, the vendor might lack sound release processes or thoroughness.
- Does the documentation clearly address the needs of all your stakeholders? Is it solely developer focused or does it also take business users into account?
- Does the documentation contain useful examples, including those posted on source-code repositories like GitHub? Do the examples address real-world use or merely demonstrate abstract concepts?
- Does the documentation include clearly and concisely written tutorials?

Baseline capability	Clear and up-to-date product documentation.
Advanced capabilities	• Conceptual and practitioner documentation. • Example code and tutorials.

Reputation

FUNCTION: DISCOVER CREATE ENGINEER

Determine if the vendor has published customer case studies that you can review and relate to your use case. In addition, ask other authoritative sources, such as analysts and implementation partners, about the vendor's reputation, viability, and relative scoring for the above criteria.

Baseline capability	Common usage from technical and implementation partners you already work with through webinars, integrations, etc.
Advanced capabilities	• Case studies of the vendor's industrial clients or use cases similar to yours. • Recognition from industry analysts. **NOTE:** Don't just look at rankings because vendors have strengths and weaknesses relevant to use cases. If you have a niche use case, it's worth vetting a niche vendor instead of a leading one.

Functional criteria

The criteria described in this section relate to the product itself. Since costs can vary considerably based on features, accurately assess your true needs in advance. For example, single sign-on (SSO) is often priced at an enterprise tier, an order of magnitude higher than the lowest tier.

Key takeaways

Do the following:

- Discuss your needs with stakeholders beforehand and weigh your criteria appropriately.
- Refrain from being distracted by the shiny attractions in a vendor demo. They might look impressive but distract from the underlying—yet ultimately more important—requirements.

The criteria below are ranked, starting with the simplest and most common needs. We recommend that you establish your own prioritization and weighting.

Delivery model: SaaS versus on-premise	The way in which the vendor delivers the software to you and maintains it.
Pricing model	The price options of the software. Pricing models vary from complexity, e.g., more content types, roles, etc.; to consumption, e.g., pay for API calls. Pick one that fits your needs.
Content modeling capabilities	The ability to create and link content types, which is a foundational capability of headless CMSes.
Delivery APIs	The ability to deliver content through APIs and query granularly and performantly.
Authoring interface	The maturity and usability of the interface on which authors create and reuse content, especially for applications and sites that are updated regularly.

Preview	The ability to double-check page content before publication, which is necessary even though headless and composable systems require that content be portable—and not locked into a channel-specific or proprietary format, hence separate from the content's presentation.
DevOps and application-management capabilities (including management APIs)	The ability to interact with the application through management APIs, which is a differentiator for advanced use cases for maintaining applications, especially at an enterprise level.
Software development kits (SDKs)	The ability to more easily program in a specific language through language-dependent, vendor-offered API layers on top of their underlying CMS APIs, called SDKs.
Internal search	The ability to reference and sort content internally by content creators and through APIs.
Content delivery network CDN	The ability to deliver content fast, particularly for globally distributed consumers.
Workflow	The ability to move content through various stages of editing and review, which is important to organizations with stringent governance requirements.
Governance (roles and permissions)	The ability to define granular roles and permissions so as to restrict content viewing and editing privileges to the appropriate groups.

Internationalization (“i18n”)	The ability to work with multilingual content and integrate with translation providers, in some cases through an editing interface in other languages.
Integration and plug-in framework	The ability to build applications—along with other elements of a composable stack—on top of the platform.
UX customization framework	The ability to extend or customize the UX for content creators.

Delivery model: SaaS versus on-premise

FUNCTION: DEPLOY ENGINEER

Most modern headless CMSes are SaaS, i.e., the vendor offers you a web-based login, after which you interact directly with the vendor-maintained software. The advantage of this approach is that, since the vendor is responsible for the software's performance, scalability, upgrades, and security, you spend less operational effort on running the underlying systems.

Note the terms of the Service Level Agreement (SLA) for uptime, performance, and, in some cases, throttling of API requests. If you are using other delivery systems for caching or static generation of content, that might be less of an issue. But, if you are working with the APIs "live" regularly, treat the SLA as a top priority.

Some headless and open-source vendors fit in a composable stack because they share many of the same characteristics like content modeling and delivery APIs. Be aware, however, that you might have to tackle legacy problems: hosting, security patches, upgrades.

Pricing model

FUNCTION: DISCOVER CONTENT DESIGN **DEPLOY**

Typical pricing levers for most CMSes belong to these categories:

- Organizational complexity
- Content complexity and scale
- Enterprise features
- Consumption

Why are they important? Because some organizations have advanced complexity but low consumption, and vice versa. Ensure that the vendor's pricing model is aligned to your needs. The exact same requirements might result in vastly different quotes from the vendors you're considering.

In general, small use cases boast pricing models that are clearly detailed on vendor websites. If, however, you need enterprise features, such as CDNs and SSO, pricing is still opaque: You go from simple pay-as-you-go, usually with a credit card, to enterprise sales processes, which usually involve more discovery, needs-based negotiation, longer contract agreements, etc.

See the criteria below.

Organizational complexity

This criterion encompasses the numbers of users and roles, called workflow types. Also think about the ability to customize roles in depth, e.g., "Our German translation provider can read and update the German content in these branches only."

Content complexity and scale

This criterion includes the limits of the number of repositories, content types, and content items.

Enterprise features

This criterion comprises these advanced features: auditing, granting access to external or guest users, and SSO. More advanced SLAs for application performance or support responsiveness might also be a factor.

Consumption

This criterion includes metrics, such as the number of API calls, storage, and bandwidth.

Important: Weigh your solution-specific needs across the entire composable stack. If you are on a DXCP like Uniform, you can often limit the number of roles and content types within your CMS, and run with a lower tier. Similarly, if you are integrating a digital asset management (DAM) system, you likely need not pay for extra storage in your CMS.

Content modeling capabilities

FUNCTION: CONTENT DESIGN

Content modeling, a foundational capability of omnichannel CMSes, typically includes field types like text, images, rich text, checkboxes, etc. If you need specific features, such as addresses and geolocation data, identify them and verify that they exist. If they don't, find out if you could extend the system to accommodate your needs and how easy it is to do so.

Baseline capability	A web-based interface for business users to create content models. **NOTE:** Some vendors lack this ability and offer a code/configuration-only function, which is acceptable if your content-management practice is largely developer led.
Advanced capabilities	• More advanced field options: • Granular reference control: one-way, two-way, etc. • Raw JSON storage along with the ability to customize UIs. This is a major area of extensibility for complex use cases. • Management API with export, enabling developers to make updates and take charge of source control. • Custom field types and interfaces. • Permissions for field values, e.g., the number of references to other items. • Data validation capabilities: • Basic: out-of-the-box rules or regular expressions. • Advanced: the ability to call custom code.

Channel-specific content modeling

Some headless CMS vendors emphasize the ability to smoothly create content for visual display through a capability like modules or blocks that can be used on a page. Though excellent for smaller use cases, those components are often problematic at scale, limiting where content can reside in omnichannel scenarios. For example, you must adapt the front-end code to work with the blocks, which might be nested, generating complexity for developers. Not to mention that the blocks might not be searchable or usable as content outside of that page context.

Internal search and findability

FUNCTION: CONTENT DESIGN CREATE

The quality of an internal search engine within the CMS becomes more important as your content multiplies and your governance becomes more complex. Being able to easily pinpoint existing content is key to its reuse through references and avoidance of duplication.

Baseline capability	The search capability for creators to easily filter content to granularly identify what you want to find based on criteria like content field types and values.
Advanced capabilities	• The ability to save and share configured searches. • Content tagging and taxonomies. • The ability to apply search criteria to reference fields. • Advanced search capabilities that are also exposed to the content delivery API, not just the authoring UI.

Since internal search capabilities are rarely external (front-end) components for your visitors to search your content, a common practice is to also evaluate external Search and Recommendations and Search Merchandising capabilities, where appropriate.

Authoring interface

FUNCTION: **CREATE**

Doubtless, the quality, performance, and extensibility of the day-to-day authoring interface is a crucial aspect of CMSes. However, those needs are too often overlooked in favor of architectural suitability, support of the programming framework, and other developer requirements, leading to frustration and dissatisfaction among teams and operational inefficiencies that translate to lost revenue.

Therefore, when evaluating solutions with your content creators, ask these important questions:

- Is the interface well designed and intuitive?
- How well does the UI match the content creators' current or desired workflows?
- How fast and responsive is the interface?
- Do content authors feel comfortable entering or creating content on the interface?
- Would common tasks be easy to do, i.e., with only minimal clicks?
- How easy is it to create and manage compound content if a single unit of work involves updating multiple content items?
- Can we customize the interface? If so, at what of these three levels?
    - Field level
    - Item level
    - High-level dashboards, i.e., the start screen that's displayed upon login
- If required, how well does the interface accommodate bulk actions like editing or updating content or metadata?
- How easy is it to add semantic information, e.g., tags and taxonomies?
- How accessible is the interface? This question is particularly important for organizations in regulated industries, which require a high degree of accessibility compliance.

NOTE:
When running your RFP and PoC phases, be sure to ask content creators to validate their scenarios and ways of working against the CMS. Oftentimes, customizing the interface is among the first project deliverables to be dropped in case of time or cost constraints. Hence, prioritize those resources in the build phase.

Preview and visual editing

FUNCTION: VISUAL DESIGN CREATE

Validate the preview capabilities and ensure that they match your requirements. No matter that, with most systems, you can build a preview URL that points to your application outside the headless system, you might require advanced capabilities, such as the ability to preview content before publishing.

Some headless CMSes also offer visual-editing capabilities, which often require a bridge API and scripts such that your output HTML contains references back to the underlying content items in your repository. Do the following:

- Confirm with your creative stakeholders if they need that type of inline editing for their tasks. If they regularly work with web content, they might prefer that to the layout-agnostic, form-based editing approach.
- Exercise real use cases. Ask authors to report how quickly they can create content in the interface, how easy the interface is to use, and how well it aligns with their workflows.
- Assess these yardsticks:
  - How concise the common workflows are by counting the number of clicks required.
  - How clear and helpful is the microcopy, e.g., field labels and error messages.
  - How easy it is to move around and find things, aka the intuitiveness of the information architecture.
  - How intuitive is the interface without the content creators having to refer to documentation or training material.
- Ensure that you can correctly attribute back to the source content, especially in larger, distributed content scenarios. This capability might limit how you can structure web properties.

Baseline capability	The ability to initiate a render with a webhook and build a preview URL for an examination of published changes by content creators.
Advanced capabilities	• Inline editing. • Preview of work-in-progress content. • Preview of changes across content items in a scheduled multiitem deployment.

DevOps and application-management capabilities

FUNCTION: **DEPLOY** **ENGINEER**

As organizations grow their content-management footprints, being able to perform more tasks programmatically and automate processes becomes increasingly critical. Even though, as a rule, those are advanced features, understanding them enables you to identify the ones your organization might require.

Webhooks

Webhooks are a critical component for headless systems to communicate events to other systems in a composable architecture. A typical use case is to initiate a search-indexing operation for new content, rerendering changes or initiating a deploy function.

Baseline capability	Webhooks for typical content operations: create, read, update, delete, publish, etc.
Advanced capabilities	• Webhooks for content-model changes or system events, such as publishing and adding users or roles. • Logging of webhooks. • The ability to customize the request, e.g., headers, format, etc.

API coverage for content modeling

Some CMSes support content modeling by business users within the application; others afford programmatic access to that functionality. The latter is key for advanced use cases, which require, for example, simultaneous deployment of code and content changes—or the ability to backup and restore or easily build projects from existing configurations

.

Baseline capability	N/A. Only certain vendors offer this capability.
Advanced capabilities	• The ability to update content models through APIs. • Webhooks for content-model changes. • Controlled and addressable development environments for testing content-model changes with rollback and deployment.

DevOps and application-management capabilities

API coverage for administration tasks

Since some systems support modeling through the UI only, some applications might not offer full API coverage for all administration tasks.

Baseline capability	N/A. Only certain vendors offer this capability.
Advanced capabilities	• The ability to update content models through APIs. • Webhooks for content-model changes. • Controlled and addressable development environments for testing content-model changes with rollback and deployment.

Bulk import and export of content

Baseline capability	N/A. Only certain vendors offer this capability.
Advanced capabilities	The ability to upload or export large amounts of content to facilitate backup or migration through a single package or another prescribed method.

Command-line interfaces (CLIs)

CLIs are key for organizations that prioritize the **repeatability** of their digital footprints. System implementers with reusable patterns and multiple clients or shared-services teams within larger organizations would consider this functionality a must-have, but smaller teams or less complex use cases might attribute less importance to it.

Baseline capability	N/A. Only certain vendors offer this capability.
Advanced capabilities	• The ability to perform common tasks, e.g., creating projects and uploading content-model changes on the command line. • Full API coverage related to management APIs and examples for advanced tasks. Also, the ability to script processes for building repeatable structures from start to finish.

Enterprise features

FUNCTION: DISCOVER **DEPLOY** **ENGINEER**

Large organizations tend to require more functions vis-à-vis compliance, governance, and logging, for example:

- Account-usage statistics and warnings, particularly if you are nearing an overage based on your license and consumption.
- SSO integration.
- A system for cross-domain identity management (SCIM) to automatically share groups or roles with other enterprise systems and provision users.
- Audit logs for access and content updates.
- Bulk backup and restore.
- Compliance certifications, e.g. GDPR, CCPA, SOC2, and ISO27001.

Software development kits (SDKs)

FUNCTION: DEPLOY ENGINEER

Technically, you can use any headless system with any programming language or framework. In practice, however, you'd likely want to align your team's efforts around single frameworks that work across applications. Similarly, if you have teams in older server-side languages, you must ensure that those languages are supported through SDKs.

In general, SDKs support headless systems in three groups:

- **JavaScript-based frameworks**
  These frameworks offer the most common support for development on headless CMSes. Examples are React, Vue, Angular, and static-site generation systems, such as Gatsby and 11ty, which are based on core JavaScript frameworks.
- **Server-side languages**
  If your development team specializes in languages like Java, PHP, and C# (.Net), see if SDK support is available for a slick integration of the headless CMS into your existing teams and applications. However, just because your organization might use server-side languages doesn't mean that your team does. Don't list this item as a requirement unless it's actually aligned to relevant use cases.
- **Mobile languages**
  If you're developing native mobile applications, analyze the SDK support for the use case on the related frameworks: React Native, Flutter, Swift (iOS), Android.

Delivery APIs

FUNCTION: **ENGINEER**

Most headless applications and front-end frameworks now exchange data through JSON. Legacy RFPs might reference XML, but, unless your use cases must interoperate with specific XML types like DITA for documentation, just review the JSON capabilities.

If your CMS cannot deliver JSON over REST, the most popular delivery method, most people probably wouldn't consider it headless. Plus, most modern development practices work with JSON output.

Baseline capability	JSON delivered over REST-based APIs.
Advanced capabilities	JSON delivered over GraphQL.

JSON and REST

JSON is now the most commonly used data-interchange format for most composable applications. Despite its name's reference to JavaScript, JSON is actually language agnostic as a way to set up data already in the correct JavaScript array format. The upshot is, unlike XML, which requires transformation, a JSON response can be referenced immediately and natively by all JavaScript frameworks.

Also, while XML understands data formats, aka schemas for applications, JSON does not. When looking at your CMS's JSON and REST output, seek the answers to these questions:

- How verbose is the response? Are there unnecessary system fields in the output? If so, can I minimize them?
- Can I pull referenced content into a single query?
- Does the response require nesting, i.e., traversing many parent-children levels in the array?

Delivery APIs

GraphQL

GraphQL was invented by Meta, formerly Facebook, for generating more granular queries to underlying data sources. Previously, an API would provide all request-related information. With GraphQL, you can not only select specific fields, but also retrieve data from the related items through, for example, references or child items. Doing so minimizes the number and size of API requests, which counts in scenarios with latency and performance issues.

Despite its being an advanced capability, not all scenarios require GraphQL. Conversely, the GraphQL API exposed by a CMS frequently has restrictions that limit the complexity. For example, you might have a query size limit or might not be able to nest deeply enough to retrieve all the necessary elements—a common occurrence for atomic content models. Therefore, before adopting GraphQL, validate its capabilities against your use case.

Since GraphQL is no magic bullet, determine if it costs extra and if your use case does need it. Scenarios like Static Site Generation (SSG) and DXCPs do not require it because they act as a filtering and linking layer, similar to GraphQL. In the case of DXCPs, GraphQL could be a business-user capability rather than solely a developer one.

Rich text editing (RTE)

FUNCTION: CONTENT DESIGN CREATE

Currently, three common ways are available for storing and working with rich text on a headless CMS: HTML, Markdown, and JSON, each with its own pros and cons.

HTML is difficult to transform for nonweb uses, e.g., a mobile application. However, you must **convert** Markdown or JSON, both agnostic formats, for use in HTML output. In either case, find out how complex and proprietary the JSON schema structure is and how robust the conversion and extension tools are within the system.

Baseline capability	Storage of rich text in a format that fits your use case, e.g., web, omnichannel, etc.
Advanced capabilities	• Channel-agnostic storage, ideally JSON, and a robust transformation framework. • The ability to extend RTE with custom formats or tags.

CDN and delivery performance

FUNCTION: **DEPLOY**

If you are directly reading from a content-delivery API, that is, not with static generation or on a DXCP, evaluate the capabilities of the API and CDN against your requirements. Answer these questions:

- Do I require multiple, redundant CDNs?
- Do I require content delivery for multiple locations worldwide?
- Are there API restrictions and limits?
- Does the vendor provide an SLA and uptime statistics?

Workflows

FUNCTION: CONTENT DESIGN | CREATE

Workflow needs and capabilities vary vastly among vendors and implementations. The complexity of workflow requirements tends to scale based on the number of approvals and the degree of compliance with regulations.

For example, healthcare providers have the most complex workflows, typically involving many approvals—legal, marketing, product teams, regional compliance, etc.—and the timing of marketing and regulatory announcements. Conversely, for organizations with a high degree of trust and little oversight, people mostly just make updates and publish them as necessary.

Baseline capability	The ability to have configured approval steps for different roles, enabling typical draft ▸ review ▸ publish processes
Advanced capabilities	• Support for multiple workflows. • The ability to preassign on specific content types or through rules. • The ability to route rules-based tasks. • Extensibility, such as webhooks, for enabling external processes like translation. • External review and approval. • Advanced collaboration features, such as inline commenting. • The ability to launch multiple items in a single release.

Governance roles and permissions

FUNCTION: CONTENT DESIGN CREATE

Mostly, you desire the minimum level of governance that meets the most critical organizational needs. Many organizations spend a great deal of time defining complex rules and requirements up front, only to discover that those regulations are an ongoing obstacle for their content teams. Nonetheless, in larger and more sophisticated organizations, e.g., multinational corporations, sound governance raises agility and efficiency by making it unambiguously clear what content can be used, shared, or published among teams.

Baseline capability	The ability to define roles and permissions for common tasks, such as creating, editing, publishing, and personas.
Advanced capabilities	Granular or custom roles and permissions based on content characteristics like language and locale.

Internationalization (i18n)

FUNCTION: CONTENT DESIGN CREATE

Multilingual support is key for most sizable organizations. Obviously, some of them are large multinationals, but significant populations in local markets prefer the option of using another language they're more familiar with. Thus, clearly define your requirements in multilingual support, which addresses concerns across multiple categories:

- **Application**
  Must the CMS be available in other languages? If so, does the vendor support them? Otherwise, can I set up my own support? What does that process entail?
- **Translation**
  Can I create and manage content in multiple languages for those who will consume it? Are there integrations that enable translation with other providers?
- **Localization**
  Can I customize both the translation and the content for various locales to meet different needs? For example, if I translate a regulatory block in English to French, can I also enable content changes for the Quebec version versus the Belgium one if the regulatory language is only slightly different?

Baseline capability	The ability to work with multiple languages and locales per content item.
Advanced capabilities	• The ability to define fields as being translatable or not, such as content that remains the same for all languages, e.g., metadata and imagery. • A validation process through which to audit and ensure translation coverage. • Configuration of fallback to another language in case of missing content in a language. • Language-based roles and permissions. • Integration with translation providers.

NOTE:
Internationalization can get extremely complex, let alone that vendor support is often limited. If you do require internationalization, be sure to add it to your PoC, including, if applicable, workaround governance and permissions to enable internationalization correctly.

Integration and plug-in framework

FUNCTION: CREATE ENGINEER

A significant advantage of composable is that integrating systems is mostly a smooth sail. Most systems contain clear APIs and methods, such as iframes, for embedding external functionalities into the systems' application. Given that a digital experience in composable requires integration as a rule, a robust solution is a key differentiator.

Contrast that approach with legacy systems, which usually mandate that you integrate on top of their application, code in their specific programming language, and update version references regularly—a process aptly called the expressway to upgrade hell.

Nonetheless, some vendors enable integration more effectively than others through concrete APIs and means for connecting applications. Similarly, if your underlying content systems lack a reliable approach, consider a DXCP as a way of connecting to other systems.

Baseline capability	The ability to integrate with third-party services within the application, e.g., connecting to a DAM system for image or document assets.
Advanced capabilities	• The ability to run customizations as background processes. • The ability on the vendor's part to host integration code. Absent that ability, you must do that yourself—usually alongside other applications. • Well-documented design systems with which partners can build integrations with a similar look and feel of the rest of the application.

UI customization framework

FUNCTION: CREATE ENGINEER

The ability to customize UIs is a default requirement for all but the simplest scenarios. Here are two related tasks:

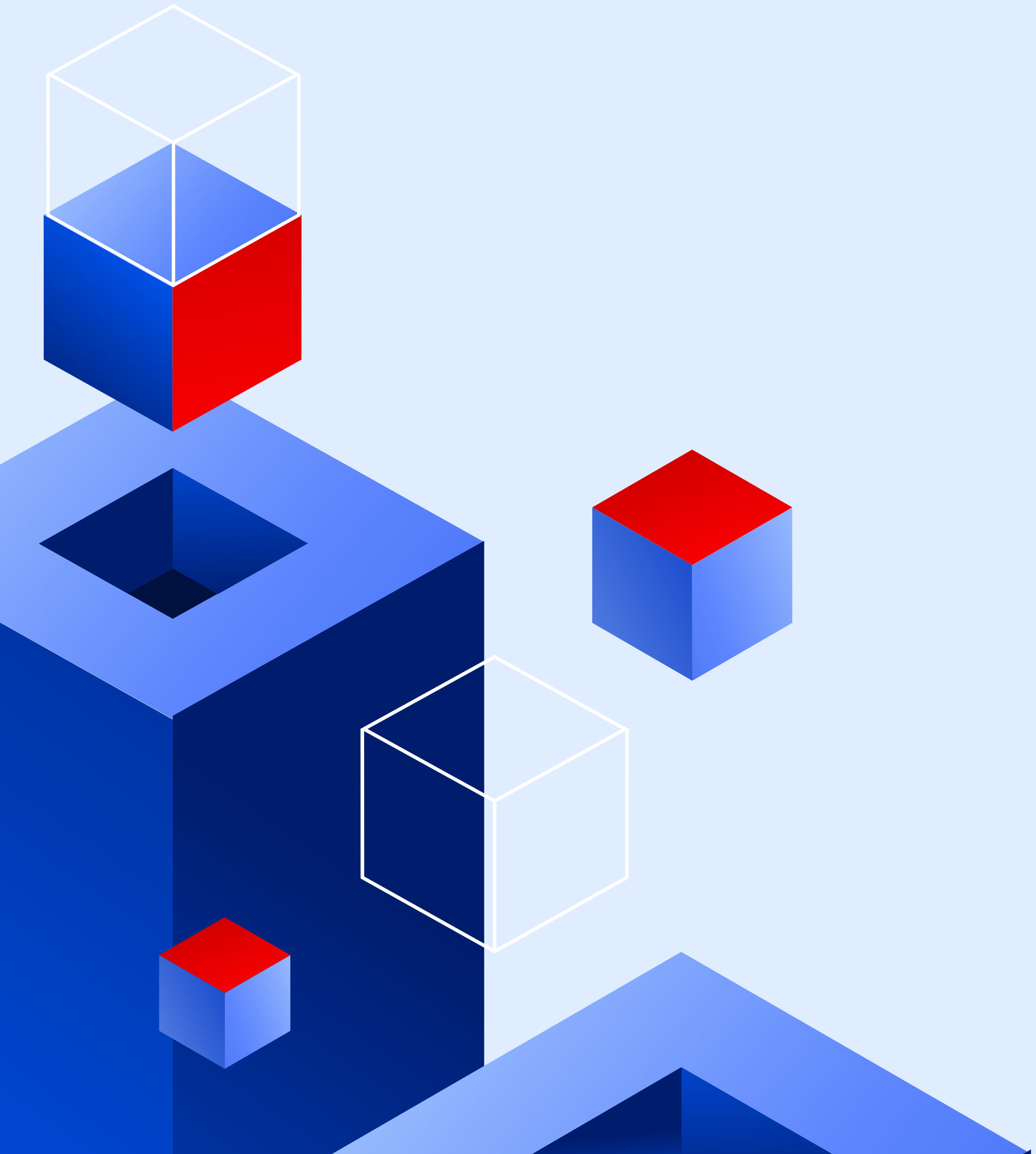
1. Adding custom fields for your domain and extending the interface as necessary, which significantly fosters adoption and satisfaction among content creators.
2. Extending the CMS to the level required.

Baseline capability	The ability to customize content-field interfaces, e.g., create a brand-compliant color picker.
Advanced capabilities	• The ability to customize the UI for editing the entire item. • The ability to customize dashboards, i.e., the start screens. • The ability for the vendor to host the code required for customization.

Considerations for composable stacks

To determine how much the success of your digital project depends on composable, you must have a clear grasp of the larger ecosystem and your requirements. The greater number of dependent systems, the more important interoperability is, and the more likely you'll switch the elements within the stack.

Headless CMSes customarily offer the features and functions for content modeling and storage along with all the other capabilities described above. However, a modern digital experience typically requires more functions, including those in the pages that follow.



Rendering of the front end and templating

FUNCTION: VISUAL DESIGN CREATE ENGINEER

Even though programming languages and frameworks were among the mission-critical choices for early resolution in an RFP process in the past, they have been moderately upended by composable architectures.

Given the lightning-fast evolution of language frameworks and channels, you can, hands down, try out—and discard—new frameworks with composable, opening up new opportunities with minimal risk. Before, your vendor choices and programming languages were inexorably linked. Not in a composable world.

Nevertheless, if you prefer a particular framework, verify that all the related vendors offer SDKs that support that framework.

Ask these three questions while gauging front-end rendering approaches:

1. What frameworks does my run team prefer?
2. Do my main composable applications furnish SDKs for those frameworks?
3. Will my use case be accessing the CMS APIs live, through static-site rendering, or both?

In addition, be aware that—

- A performance tradeoff is the norm. SSG sites, though loading much faster, often require longer build times, limiting preview and causing frustration for content creators.
- A common complaint against headless systems is that they deprive business users of control when those folks are building pages because tasks like page layout or personalization require developer help. A DXCP can help here so that developers can continue to work on top of headless APIs while business users manage the visual and structural elements.

Hosting and delivery

FUNCTION: **DEPLOY** **ENGINEER**

Unless you are serving mobile devices that run your application as an app and accessing underlying APIs directly, hosting must be in place to provide a domain to route requests and host your site code. Some platforms offer additional benefits, such as edge-caching and computation or rendering as a built-in feature, e.g., Vercel Incremental Static Regeneration. You then get the best of both worlds: fast front-end performance and minimized buildtimes.

Digital experience composition

FUNCTION: CONTENT DESIGN CREATE ENGINEER

Gartner first defined the digital experience composition category in a 2022 article *Hype Cycle for Digital Commerce, 2022* and then in a later post *Innovation Insight for Digital Experience Composition.* The related technology realizes the potential of headless systems in a larger composable approach.

Gartner's definition calls for a product that satisfies three requirements:

1. **Prebuilt connectors f**or content sources, with which to reduce the amount of data-access logic to be built by developers.
2. **No-code or low-code tools,** e.g., webpages, mobile-app screens, and email campaigns, with which business users can create and manage digital experiences. Those tasks involve leveraging data from the prebuilt connectors and UX components from a design system or component library.
3. **Front-end orchestration,** which encompasses the capabilities that make the digital experiences composed by business users available to end-users. Examples are front-end components and connections to CDNs.

NOTE:

Uniform is an example of a DXCP with additional capabilities for personalization and caching of underlying systems for improved performance.

Customer data

FUNCTION: ANALYZE & OPTIMIZE

Customer data is important for delivering relevant personal experiences. The sources of customer data described below are in wide use in composable architectures.

Customer data platforms (CDPs)

As first-party information sources that work across channels, CDPs come with functions for cleaning data; unifying profiles, e.g., someone logging in to the same account on multiple devices; and building audiences for personalization. For composable, the APIs' ease of use and ability to work with those segments in real time are important features. Verify that you can consume those segments into your downstream usage, either directly on your site and your custom code, or through a personalization vendor.

B2B or account-based marketing (ABM)

ABM platforms lend an understanding of visitor-segmentation information based upon network identifiers, affording marketers an insight into the industry, revenue, company name, and other information. If you need a high degree of segmentation and must qualify at scale, integrate ABM platforms into your stack. Take into account the answers to these questions:

- Can I query for the information in real time for a fast consumer experience?
- Can I readily ascertain that my segments and content sources are synchronized between applications and delivery to channels to avoid duplication and accelerate content creation and deployment?

Data management platforms (DMPs)

DMPs are third-party data sources that are coordinated with advertisers and resold. Those ads that pop up that relate to a product you browsed on a site you visited before are likely thanks to a DMP.

DMPs are falling out of favor, however, because privacy regulations in jurisdictions are limiting the data-sharing and metadata arrangements that make those targeted ads effective and economical for sellers. Instead, first-party data captured and enabled by CDPs is gaining popularity.

Personalization

FUNCTION: ANALYZE & OPTIMIZE

While evaluating personalization in the context of composable vendors, answer this question on how to manage content variations: Can I consume these context variations—and create personalization rules—from within my CMS? If the answer is no, a management issue might ensue unless solid governance is available to prevent old content from being used in those contexts.

Another critical aspect to evaluate is front-end performance. Many personalization solutions perform DOM replacement on the client side, i.e., they deliver a page to your end-user and then swap out content. Negative consequences emerge, including the following:

- Content must be copied to your personalization system, limiting agility and introducing errors in case of out-of-date content.
- The swap step adds time and request round trips. The resulting performance lag might negate any uptick in content success from effective targeting. Similarly, personalizing across systems on top of a CMS, commerce, and search provider would be difficult, if not impossible, without a DXCP.

Uniform was founded to address the challenges of content duplication and usability in headless scenarios with a focus on personalization and speedy rendering through edge technologies. Ultimately, we built a full DXCP.

Commerce

FUNCTION: CREATE ANALYZE & OPTIMIZE

Commerce requirements include functions related to product catalogs, product recommendations, customer profiles, pricing, cart and checkout, offers, and promotions, some of which constantly bleed into each other. Some vendors offer more functions; others offer fewer. The requirements described below actually contain categories.

Product information management (PIM) system

All commerce implementations require a product catalog, and stand-alone PIM systems usually boast capabilities that go beyond commerce, including the ability to associate imagery and comply with regional labeling requirements. Since some DAM systems have PIM features as well, allow more time here to validate the options and approaches for complex product-catalog needs.

Recommendations and search merchandising

Since larger retailers customarily recommend products through external systems, knowledge of your datasets, the way in which to leverage them in recommendations, and even the process of incorporating them into navigation and product-display pages is essential. Remember that a recommendation algorithm is only as good as the data sources behind it. So, make it a priority to learn the input requirements of that system and the output requirements of your data sources.

Search

FUNCTION: CONTENT DESIGN CREATE

Although most CMSes comprise internal search capabilities for authoring findability and reference creation, those features are not on the front end, rendering an external search index a popular element. To meet commerce- and recommendations-oriented requirements, a specialized Recommendations and Search Merchandising tool might be the answer.

Email and marketing automation

FUNCTION: CONTENT DESIGN CREATE ANALYZE & OPTIMIZE

An effective composable platform should share content and customer data among all your channels, including email. By sharing those same foundational data sources, you can ensure consistent customer experiences across channels. Be sure to validate that you can repurpose the audience segments and content from your other systems without difficulty.

Content repositories

FUNCTION: CONTENT DESIGN CREATE

Although content repositories are a resource for assessing a CMS, organizations tend to have multiple CMSes, e.g., a DXP like Adobe Experience Manager for the marketing site, WordPress for the corporate blog, and Contentful for the mobile application.

To build a cohesive experience across those platforms, see that they can integrate seamlessly or adopt a DXCP for an out-of-the-box experience.

Media-asset repositories

FUNCTION: VISUAL DESIGN CREATE

DAM is a core technology for many sites filled with media: images, video, documents, even 3D rendering assets. Even though you can handle media-management tasks with most CMSes, you might have additional media-creation workflow or transcoding needs, which your CMS does not support.

If you are leveraging a DAM system, which is ordinarily deeply embedded into a content-creation process, verify that the DAM system and CMS work together. It would be ideal if the DAM vendor supplies a plug-in for your CMS. Incorporate the plug-in into the PoC you're evaluating. Otherwise, see if you can effectively assemble content and assets together on a DXCP.

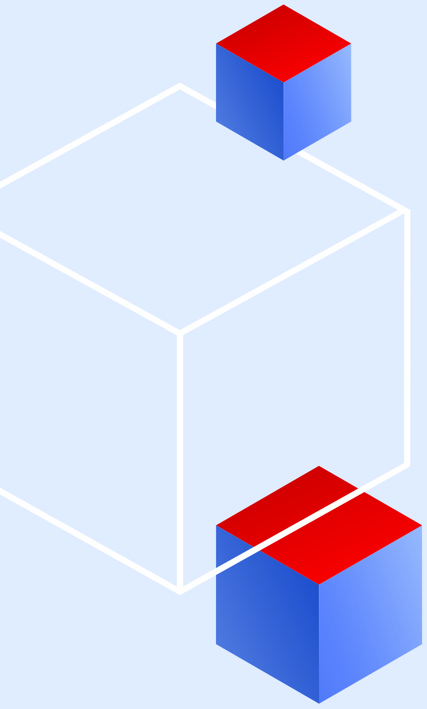
NOTE:

While evaluating DAM systems, verify that they are true SaaS applications and that they deliver assets. Older DAM systems stored assets locally and, even if they have moved to the cloud, their delivery capabilities might be subpar.

Form handling

FUNCTION: CREATE ANALYZE & OPTIMIZE

A routine requirement for most organizations is collecting form submissions related to sales and marketing activities. That task could become extremely complex given GDPR and CCPA rules and the process of leveraging the data for marketing and channel campaigns.



Conclusions

Before choosing a CMS for composable, become familiar with the digital landscape and requirements for functions and vendors. Since SaaS delivery spells faster iterations and PoC efforts, shifting toward architectural openness creates a more sustainable platform, necessitating foundational investment and organizational change yet yielding long-lasting benefits.

Transition to composable calls for a lot of organizational change and advance planning. The simple comparison to LEGO bricks is commonplace, but a more accurate metaphor would be erecting a building with a pile of random bricks without a manual. Besides infinite flexibility, you must also build the muscle of knowing what pieces fit together to achieve various aims. We hope that this guide is a useful resource, equipping you with the knowledge of the pieces you need and the process of assembling them.

We also hope that this guide highlights the fact that you must invest more around the organizational mindset of how to efficiently plan, learn, and adapt well, and that a composable approach accelerates and complements that working style. In contrast, attempting to implement composable without changing the underlying organizational mindset results in solution gaps and team bottlenecks rather than technology winners.

All told, as noted at the top of Chapter 1, notwithstanding all the effort, “those who have adopted composable all agree that it’s the right thing to do.”

Acknowledgments

This guide was written by experts with hands-on experience and expertise in many projects with numerous vendors in all the categories mentioned. We hope our explanations, suggestions, and prompts will help you succeed in your composable journey.

Much of the experience behind this guide went into building a DXCP at Uniform, specifically to speedily propel composable projects by solving many of the conventional challenges.

Special thanks go to these industry luminaries: Irina Guseva of Gartner (https://gartner.com), Karen McGrane of Autogram (https://autogram.is), and Cathy McKnight of The Content Advisory (https://contentadvisory.net), for their external perspective and valuable feedback.

About Uniform

With Uniform Digital Experience Composition Platform (DXCP), you can quickly combine headless services with their legacy tools in an intuitive, visual interface on which to create web and app experiences at a speed beyond your competitors' belief.

While doing that, you can eliminate huge amounts of tech debt—the most boring tasks—for developers, simultaneously freeing marketers from waits in the developer backlog and arming those professionals with the tools they need to efficiently and agilely meet their KPIs.

Customers that have adopted Uniform include Cobham Satcom, Sunweb, and Triumph.

Learn more at uniform.dev and follow us on LinkedIn and Twitter.