
CONTENTSTACK®
THE BEST OF
JOHN WEST
Headless CMS Advocate
Former Sitecore CTO
A collection of his top blogs
that answer top CMS questions.

WHO IS JOHN WEST?

His interest in computers began in the mid-1980s on the Commodore VIC-20. Regardless of the different jobs John West has held throughout the years, he considers his role to be Chief Customer Advocate — professionally consulting on platform selection and solution implementation, supporting developers in implementing maintainable, scalable, usable and high-performance CMS solutions.

West has been thinking about, blogging about, and debating the finer points of CMS since the early 2000s. Here are some highlights of his full work history:

- In the late 1990s, using mainly Solaris, Oracle, and C++, West implemented systems that automated diverse business processes responsible for hundreds of millions of dollars in transactions annually.
- In 2000, West began consulting using the TeamSite Content Management System (CMS) and soon began using their C# and .NET before joining Sitecore in 2004.
- At Sitecore, West assisted with sales, provided tech support, taught training, and eventually wrote over 500 blog posts and a book about Sitecore.
- After Sitecore, he worked with various CMS vendors, several system integrators and for multiple customers, implementing complex solutions with Content Management Systems.

West's vast experience integrating countless upstream and downstream systems across all verticals has given him a broad understanding of customer requirements and application architectures. He experienced an aha moment while evaluating vendors for a systems integrator in 2019. Because he wasn't a JavaScript developer, he had largely ignored the trend towards headless content management — until he encountered Contentstack and its key differentiators that set it apart from the competition.

Headless CMS and Contentstack got West excited about technology again. In Contentstack, he has found a platform that he considers superior to .NET or any other SaaS headless CMS.

The following are his blogs, his words, exactly as written.

TABLE OF CONTENTS

WHAT YOU NEED TO KNOW ABOUT CMS

THE BASICS

Blogs

- Deliverystack Author: John West (commodore73) | page 5
- What Does It Mean for a CMS to Be Headless? | page 6
- Approaching SaaS Headless CMS Projects | page 8
- Differentiating Headless CMS Products | page 10

LET'S GET TECHNICAL

A DEEPER DIVE INTO THE WORLD OF CMS

Blogs

- SaaS Headless CMS Fundamental Concepts | page 13
- .NET Headless Content Management Considerations | page 15
- Making the Transition to SaaS Headless CMS and Service Architectures | page 18
- SaaS Headless CMS and Service Oriented Application Objectives and Advantages | page 20
- Contentstack SaaS Headless CMS Content Delivery Differentiating Factors | page 22

WHAT YOU NEED TO KNOW ABOUT CMS

THE BASICS

Blogs

- Deliverystack Author: John West (commodore73) | page 5
- What Does It Mean for a CMS to Be Headless? | page 6
- Approaching SaaS Headless CMS Projects | page 8
- Differentiating Headless CMS Products | page 10

DELIVERYSTACK AUTHOR: JOHN WEST (COMMODORE73)

I have been playing with computers since I gained access to a Commodore VIC-20 in the mid-1980s. In the late 1990s, using mainly Solaris, Oracle, and C++, I implemented systems that automated diverse business processes responsible for hundreds of millions of dollars in transactions annually. In 2000 I started consulting with the (now defunct) Interwoven product line including its TeamSite Content Management System (CMS). In 2002 I began using C# and .NET with TeamSite before joining Sitecore in 2004. As Sitecore expanded from an ASP.NET web content management system into omnichannel content delivery, experience management (analytics, segmentation/personalization/individualization, email campaign management, marketing/engagement automation, testing/optimization, and so forth), commerce, and other categories, I assisted with sales, provided technical support, wrote documentation, taught training, and eventually wrote over 500 blog posts and a book about Sitecore. The book was admittedly never much good. Both time and multiple blogging platform migrations have destroyed the blog (remember when everything was HTTP instead of HTTPS?). Maybe this is fitting because much of the content is obsolete or lacks sufficient context for use today.

After Sitecore, I spent some time as a digital experience management technical lead for an agency. In addition to multiple roles with CMS vendors, I have worked for a digital advertising firm, with several system integrators, and directly for multiple customers, all implementing complex solutions, generally with a Content Management System as a sort of hub or at least a critical component. My experience integrating countless upstream and downstream systems across all verticals have given me a broad understanding of customer requirements and application architectures.

Not being a JavaScript developer, I had largely ignored the trend towards headless content management. While evaluating CMS vendors for a systems integrator in 2019, I encountered Contentstack and determined what I consider to be some of its differentiating factors relative to its competitors. My experiences with Contentstack reminded me of early days with Sitecore that prompted me to write one of my first blog posts (Three Factors to Sitecore's Success). Headless CMS and Contentstack specifically have gotten me excited about technology again. I try to evaluate things impartially, but I have not found a platform that I consider preferable to .NET or a SaaS headless CMS that I find preferable to Contentstack. For reasons having nothing to do with content management, I consider WordPress to be a better blogging platform than Contentstack (I would host my blog on Contentstack if I was already hosting a site with Contentstack), but I would not use WordPress as a CMS. I will use C# whenever I can, and my coverage of Contentstack will be more complete than the alternatives. Regardless of my title, when working for a CMS vendor, I consider my role to be that of Chief Customer Advocate, responsible for professional consulting on platform selection and solution implementation with focus on supporting developers and improving their satisfaction and productivity in implementing maintainable, scalable, usable, high-performance CMS solutions.

I will try to answer any questions that appear in comments on this blog. I will likely redirect some questions to https://stackoverflow.com/ or elsewhere. You can also connect with me at https://www.linkedin.com/in/johnwest3/.

P.S. I have not taken the time to configure navigation in WordPress; I'm just dumping content here. I recommend that you start from the first/front/home page at https://deliverystack.net and use the links in the footer to navigate forwards through the posts.

Here are my other tech blogs, though I generally do not maintain the posts about Sitecore anymore.

- https://sitecorejohn.wordpress.com/
- https://community.sitecore.net/technical_blogs/b/sitecorejohn_blog/
- wslguy.net – Using Windows Subsystem for Linux
- rustapopoulos.com – Yet another blog about programming with rust

P.P.S. I make a lot of mistakes and do not believe in final drafts, so anything on this blog could change at any time.

July 20, 2020

WHAT DOES IT MEAN FOR A CMS TO BE HEADLESS?

This blog post provides perspectives on what it means for a content management system to be considered headless. A content management system structures and separates data from its use, allowing domain experts to maintain content used by delivery systems such as websites and mobile applications. Rather than providing HTML generation engines, headless CMS expose content as JSON over HTTPS services. If you have perspectives on what it means for a CMS to be headless, please comment on this blog post.

DECOUPLED

Headless content management is said to be decoupled from content delivery, meaning that the customer has greater control of the tools and phase(s) in the solution lifecycle at which to apply different levels and forms of coupling.

- Coupling in [Headless] CMS – Deliverystack.net

THE HEADLESS CMS METAPHOR

In the headless CMS metaphor, the body is the content management system, where CMS users maintain content. The head is the content delivery system used to make that content available to the visiting public. Headless CMS provides bodies, but not heads.

- The Headless [CMS] Metaphor – Deliverystack.net

Traditionally, content management systems included platform-native content delivery software that customers would install as a CMS and to generate static files or to access data from the CMS. Headless CMS solutions do not include any software for generating files and expose only HTTPS APIs with platform-native software development kits (SDKs) that simplify their use. Reducing direct coupling in the solution architecture has numerous advantages, most of which derive from using SaaS.

- SaaS Headless CMS Vendor and Customer Responsibilities – Deliverystack.net

SAAS

Using Software-as-a-Service (SaaS) applications reduces Total Cost of Ownership (TCO) and improves ROI for CMS most implementations. SaaS headless CMS customers can use the CMS vendor's content delivery network (CDN) to scale global caches of content and media used by other applications without the need to understand and manage applications or application servers.

SaaS CMS vendors host content management environments and content delivery services for their customers, as well as SDKs for implementing content delivery solutions. SaaS headless CMS vendors are responsible for content management environments and content delivery services including media.

Content delivery solutions implemented by customers access content delivery APIs in the build process, while generating HTML, or using end-client logic. Headless CMS customers are responsible for implementing, hosting, and deploying content delivery solutions that consume data from the CMS, often using CDNs to host static files, optionally including server-side logic and often using JavaScript to implement applications on the client. Headless CMS customers are responsible for content delivery solutions including implementation, hosting, and deployment.

- Three Major Phases for Data Access in Headless Content Management Solutions – Deliverystack.net

Not all headless CMS software is SaaS. Customers that choose not to use SaaS can provision and manage instances of some headless CMS software on their own infrastructure. This may be especially relevant for open source CMS projects where customers want to factor their custom enhancements into the CMS solution. Alternatively, some vendors can provision and possibly co-manage instances of their CMS software in the customer's cloud.

NO GENERATION

A headless CMS does not include any facility to generate HTML or any other output format. A headless CMS provides a user interface for managing content, HTTPS APIs for accessing that content, and webhooks that signal other applications about events in the CMS.

While CMS is multichannel, the heads in most headless CMS solutions typically generate HTML or any other output format. To facilitate developers building websites, mobile applications, and other solutions, headless CMS vendors provide numerous SDKs, frameworks, component libraries, examples, and other elements for different technologies in use by content delivery solutions.

SDKS

Rather than content delivery engines, headless content delivery vendors provide SDKs that wrap content delivery services. To access content delivery services, headless CMS vendors provide software development kits (SDKs) for the various technologies used by their customers.

An SDK differs from an installation of the software in that it allows use of a product without installing that product separately. When custom code invokes APIs in headless CMS SDKs, the SDK invokes HTTPS services that communicate with the CMS.

TECHNOLOGY INDEPENDENT

Headless CMS is omnichannel, meaning it intends to service content delivery points other than websites. Headless CMS is used to power mobile devices, kiosks, in-store displays, menus, video game content, print, and almost any other medium imaginable.

Headless CMS vendors support as many technologies as customers require without absorbing the cost of implementing and maintaining low-use SDKs. Platform technologies generally start with JavaScript and JavaScript libraries but can also include at least .NET, apple, android, Java, PHP, and Ruby. Customers can always call CMS APIs directly, avoiding any dependency on a vendor SDK.

MICROSERVICE ARCHITECTED

Any CMS can expose services, but a headless CMS is typically service-oriented, API-first, and built upon its own services. The CMS user interface is an application separate from the CMS and uses the same APIs exposed to external developers using the product.

Microservice-architected applications expose every possible function as an API and every system event as a webhook. Working with microservice-architected applications has its own consistency, as all applications conform to parallel patterns.

Just as a headless CMS can include components used for content delivery, not every service-oriented application must be microservice-architected. Simple, custom applications developed by small teams may not benefit from microservice architectures, especially where the application exposes no external API or a simple external API.

VENDOR INDEPENDENT

One goal of headless CMS is vendor independence. Especially in a market where vendors can come and go, CMS customers must be able to migrate between vendors relatively easily, and headless is a step in that direction. As mentioned in the blog post linked in the section about coupling, headless CMS maintains some coupling between content management and content delivery systems, and hence some of the relevant risks.

November 19, 2021

APPROACHING SAAS HEADLESS CMS PROJECTS

This blog post lists some questions to start conversations while evaluating architectural requirements for SaaS headless content management projects, which typically include one or more websites. This post is not specific to any individual CMS and much of it could apply to any modern connected system, but I work for Contentstack and so emphasize relevant terminology in some places. If you have any suggestions to add to these lists, please comment on this blog post.

A successful CMS implementation prioritizes and balances customer requirements against resource constraints, of which development time is often primary. Before implementation, reviewing questions such as the following with anyone involved in the CMS can help to gather potential requirements. Help the customer understand the need to plan for potential future needs while minimizing initial expectations, especially anything that involves custom development, and then architect the solution appropriately.

What are the core goals and requirements of the solution, including expectations based on sales demonstrations, POCs, training, and experiences with other systems?

- What is the chosen CMS, and for what primary reasons?
- If there is a legacy CMS, what are the main reasons for its replacement?
- What is the existing development process, and will it change with the new system?
- Who are the parties involved in the implementation?
- What are the main websites, user interfaces, applications, and other solution deliverables?
- How often are new sites or other significant components of the solution created?
- What are the lifespans of the various components of the solution?
- What are the requirements for media?
- What are the organization's preferred development, scripting, source code, ticket, build, and other tools and technologies?
- What hosting and programming technologies will the solution use for content delivery?
- What is the size of the solution?
- What are the requirements for internationalization and localization?
- What is the information architecture of the solution?
- What is the search solution, and what are the intentions for its use?
- What is the strategy for defining and managing URLs, including redirects and vanity URLs?
- What additional solutions, such as internal systems, media, streaming, commerce, social, experience management, and otherwise need to integrate with content management and content delivery, and in what ways?
- What are the known project deliverables, when are they due, who is responsible for each, and what is their state now?
- How should preview, workflow, staging, and publishing work?
- Is there an existing volume of semi-structured or better content worth attempting to migrate automatically?
- Should visitor traffic interact with CMS, search, and other vendor platforms directly, or through services managed by the organization?
- Will this solution involve an enterprise message bus?
- In what cases would the organization use the vendor's SDKs and in which would it call the HTTPS APIs directly?

In CMS, data modeling may be more fundamental to success than development. Mapping the information architecture and other known requirements to optimize content types used by the organization's users to enter data is a skill and an art that takes some experience. To improve usability, try to minimize the number of content types and references between entries. Content types are flexible and easy to create and define. If possible, have the eventual CMS end-users evaluate prototype content types before committing to a specific strategy.

Apply vendor-specific content modeling features such as reusable structure definitions (Contentstack global fields), repeating fields (Contentstack multiple fields) and groups of fields (Contentstack groups), and nested flexible data structures (Contentstack modular blocks). Try to balance the data model to facilitate both CMS users and developers. Focus development resources on the content delivery environment (the published website) rather than the content management environment (customizations to the CMS).

While determining the content types, required fields, and field properties, identify where each CMS facility is most appropriate, specifically for integration with other systems.

- Content Management API: Import, update, workflow, publish, and otherwise manage content.
- Content Delivery API: JSON CDN of published content with query facilities.
- GraphQL API: JSON CDN of published content with query facilities.
- Synchronization: Initialize and then retrieve batches of updates.
- Media Delivery API: Including server-side image manipulation and optimization.
- Custom User Interfaces: Consider needs at the field, entry, and dashboard levels of the CMS UI.
- Webhooks: External systems respond to event notification from the CMS.

Certain patterns emerge:

- Custom field types can select data from external systems, such as a value from a selection list, the identifier of a product, or a more complex user interface consisting of multiple nested fields.
- Webhooks and/or polling and/or custom UI components can trigger synchronization APIs such as to build or update a search index, clear caches, and otherwise manage data replicated into other systems.
- A webhook listener can publish CMS events to an enterprise message bus with subscribers that trigger additional functionality.

Links

https://deliverystack.net/2020/09/08/considerations-for-migration-to-headless-cms/

https://deliverystack.net/2021/05/07/making-the-transition-to-saas-and-headless-cms/

https://deliverystack.net/2020/08/26/hard-questions-for-headless-cms-vendors/

https://deliverystack.net/2021/03/31/saas-headless-cms-architectural-considerations-part-ii/

September 10, 2021

DIFFERENTIATING HEADLESS CMS PRODUCTS

The headless CMS software space is crowded with competitors, which can make it difficult for prospective customers to evaluate available offerings. This blog post presents some criteria that can help to differentiate headless CMS products. If you have additional significant factors by which to evaluate headless CMS products, please comment on this blog post.

When evaluating vendors, consider prioritizing the following factors. Then, eliminate vendors that are not aligned with key priorities. These are ranked subjectively by my general perceived order of priority, where true prioritization would vary by organization.

COST

The fastest way to eliminate a vendor is often by cost. In many cases, you get what you pay for, and you may want to get more than some of the inexpensive options offer. Conversely, your budget may not support various offerings. Still, it is important to evaluate vendors outside of your price range to know what features are available and what might be coming.

COMPANY

For both parties to benefit most from the relationship, the CMS vendor should be a good fit for its customers. While technology is important, people are what make a company, and what make it successful. If you cannot get accurate information that you need from a vendor easily, you may want to consider alternatives. Try to evaluate the vendor's capabilities at managing relationships at all levels, including both the sales process and technical support for users and implementers. Consider whether the CMS vendor is well-led, well-funded, experiencing steady business, and prepared to manage growth. Ask the CMS vendor about recent customer attrition.

DIFFERENTIATING FACTORS

I always ask vendors to describe their differentiating factors relative to their competition and their competitions relative to theirs. If they don't answer, I ask the same question about specific competitors. Always do some research before contacting any vendor.

DATA MODELING CAPABILITIES

Data modeling refers to the definitions of the types of records to manage as well as potential relationships between records. Good data modeling can simplify implementation, improve ease of use, increase developer productivity, boost performance, simplify maintenance, extend longevity, and otherwise benefit solutions. CMS data modeling characteristics to evaluate include the range of field types provided by the system, the ability to group fields, the ability for fields and groups of fields to repeat, the ability for multiple content types to share common data structure definitions, flexible nested structures within field values, how to implement custom field types, and any other features for representing data in the system. For me, data modeling capabilities are more important than SaaS, which is extremely important.

SAAS

Some headless CMS products, especially open-source solutions, allow installation by customers. I prefer SaaS hosting models, which I feel assign appropriate responsibility to the vendor and deliver the greatest value from the cloud, allowing implementation without vendor-specific installation, infrastructure, and administration knowledge. Some organizations may prioritize SaaS differently or prefer to avoid it.

USER INTERFACE

CMS usability is critical solution acceptance and long-term satisfaction. The user interface should be intuitive enough to meet most CMS user expectations with minimal or no training. Depending on requirements, the user interface must be extensible in various ways, such as custom field types, custom tools at the entry level, and the ability to embed user interfaces elsewhere in the CMS application.

SEARCH

All CMS solutions should allow structured queries, but typically do not support stemming, synonyms, keyword highlighting, and other features used to search. Evaluate any search features available for use in content delivery, whether the vendor offers a search solution, or what the vendor recommends for search.

RULES

Some CMS provide browser-based user interfaces to define rules that control various features of the CMS, such as when and how to invoke web hooks. Rules can provide a user interface that allows relatively non-technical CMS users to define logic to apply various circumstances and when given conditions apply. If the CMS stores rule definitions in the repository, then you can publish them, allowing application of rules for content delivery visitors to the published website including support for personalization. Changes to corresponding logic without a rules engine can require deployment or a custom solution with similar functionality.

PERFORMANCE

Confirm sufficient CMS performance for users at expected distance from the infrastructure. Evaluate content delivery service performance from locations that may consume content at runtime. To meet performance requirements in certain geographies, it may be necessary to augment vendor content delivery services with your own.

HEADLESS

Some CMS that provide content delivery engines also expose services used to decouple solutions. Using JSON over HTTPS services to separate the content delivery engine from the content management system may loosen but does not remove the coupling if solution code relies on content delivery APIs that read directly from the CMS, and especially if code access content management APIs to write data back to the CMS. To approach the decoupling objectives of headless, content delivery solutions should minimize dependence on CMS vendor APIs and data structures, especially for anything other than data access. Typical coding should have little to no awareness of the CMS, only the JSON data structures defined by the content types implemented for the solution. The content delivery solution should never use content management (write) APIs and should minimize the use of content delivery APIs and vendor-specific data structures.

HIERARCHICAL

Some CMS organize records into one or more hierarchies, like a subdirectory structure. Others function more like document databases, as if all the records are in a single folder. This difference affects how CMS users navigate content and availability of the hierarchical capabilities for content delivery.

FEATURES

Depending on project requirements, evaluate CMS features such as workflow to require approval or other processes before publication, how publication works, locking or other features to prevent concurrent changes by multiple users, support for internationalization, features for managing releases.

PLATFORM

While a headless CMS should look like a black box to developers, there are places where the light leaks out – the decoupling is not complete. One such place is custom user interfaces, which typically load CMS JavaScript and CSS. Resources such as code samples from the vendor are likely to be most available for the technologies used by the vendor, including build tools, coding languages, JavaScript libraries, and otherwise. Technology may be even more important when using open source or when not using SaaS.

Update 21.September.2021:

RATE LIMITING

To avoid overwhelming network traffic, SaaS systems restrict the number of interactions per unit time per client. For example, any SaaS CMS customer may be limited to a certain number of content delivery API calls per second. Rate limiting can impact the performance of very busy content delivery systems, increase startup time for applications that cache the entire CMS repository, reduce the throughput of data import processes, and otherwise negatively impact performance of the solution. For a cost, some SaaS CMS vendors allow tiering of rate limitations, and some can host their software in the customer's private cloud.

OPEN-SOURCE

While I believe in open-source software, it's at the end of this list because I have not evaluated any vendors offering open-source headless CMS on SaaS, which is one of my top priorities. Organizations that have a commitment to open source, want to minimize costs, would like to customize the CMS or contribute to its source code, and especially those that want to administer their own installations may place open-source at a higher priority.

September 20, 2021

LET'S GET TECHNICAL

A DEEPER DIVE INTO THE WORLD OF CMS

Blogs

- SaaS Headless CMS Fundamental Concepts | page 13
- .NET Headless Content Management Considerations | page 15
- Making the Transition to SaaS Headless CMS and Service Architectures | page 18
- SaaS Headless CMS and Service Oriented Application Objectives and Advantages | page 20
- Contentstack SaaS Headless CMS Content Delivery Differentiating Factors | page 22

SAAS HEADLESS CMS FUNDAMENTAL CONCEPTS

This blog post attempts to provide an overview of some fundamental concepts for working with SaaS headless content management systems.

CONTENT MANAGEMENT SYSTEM (CMS)

A content management system separates text and media content from presentation and logic, allowing re-use of content managed by non-technical subject matter experts. Developers write code that retrieves data from the CMS and render that content as HTML or otherwise.

While each product has different features, consider all CMS from a single high-level architectural perspective. Each vendor should allow you to configure webhooks to signal external systems, provide HTTPS APIs for accessing content, and support customizations to the CMS user interface.

CONTENT DELIVERY

Using various technical architectures, content delivery systems retrieve data from the CMS and render it elsewhere, such as on a website, digital display, print, or otherwise. Where only authorized users can access content management, content delivery is generally accessible to anonymous visitors.

HEADLESS CONTENT MANAGEMENT

In the headless CMS metaphor, the body is the content management system that supports editorial activity, and the head is the content delivery system that services the visiting public. Headless content management systems are headless to the SaaS CMS vendor, who has no responsibility for content delivery, but bodiless to the customer, who has no responsibility for content management.

Headless content management systems do not include an engine for generating HTML. Customers choose from a range of available technologies that access CMS services to retrieve data as JSON over HTTPS.

JAMSTACK AND HEADLESS CMS

While any technology can work with services and CMS can serve any channel, most headless CMS solutions use Jamstack to implement websites. Such solutions use various tools to embed content from the CMS in optimized HTML files that simplify deployment and benefit performance and hence SEO. While variations are possible, HTML in Jamstack sites typically references JavaScript for dynamic features, instructing browser clients to invoke content delivery and other services directly to retrieve and render additional content.

Organizations invested in Jamstack should find headless CMS to be relatively straightforward, and organizations using headless CMS may be able to achieve greater agility and productivity with Jamstack than with other technologies. Periodically research and evaluate Jamstack technologies and JavaScript frameworks such as React and Vue.

SAAS

SaaS, for Software-as-a-Service, means that instead of selling the software to the customer to install, administer, upgrade, and otherwise maintain, the vendor hosts its own software for its customers, exposing all features as HTTPS endpoints. Compared to on-premises, Infrastructure-as-a-Service, Platform-as-a-Service, and even managed PaaS, SaaS places greater responsibility on the software vendor and reduces cost of ownership for the customer, such as the need for expertise in system internals.

Other than the fact that you are not responsible for managing it, you don't need to know anything about SaaS to work with a SaaS headless CMS. The vendor provisions a CMS environment for you, and then you log in through a browser and start modifying data, or invoke APIs from your code, without ever considering the CMS installation and upgrade process.

HTTPS APIS AND SDKS

Historically, an API meant a native application programming interface such as those available in C and Java that interact directly with local machine resources. In distributed, service-oriented architectures, an API is an HTTP endpoint with a contract for passing and returning data. Headless CMS vendors provide SDKs that abstract their HTTPS APIs as local application programming interfaces.

If you have not used services extensively in the past, the architectural shift from native to network APIs may be the most significant in the headless CMS learning process.

WEBHOOKS

Where HTTPS service APIs allow external applications to call into the CMS, webhooks allow the CMS to call out to external applications over HTTPS. A webhook simply passes system event data to the URL configured to receive it.

JSON

JavaScript Object Notation, or JSON, has become a sort of de facto format for exchanging text data between systems. HTTPS APIs exposed by headless CMS products typically use JSON by default.

While you should have a general understanding of JSON, data structures generally vary by CMS vendor and customer project, so you will have to learn them as you progress.

CDNS AND SERVERLESS

For performance and scalability, Jamstack solutions typically deploy static files to a CDN rather than to a specific farm of web servers or collection of application servers.

For server-side logic, serverless models, in which cloud infrastructure manages applications implemented by smaller independent services, minimize administrative costs and optimize scalability, Serverless really means less server, as there must still be an application server, even if the organization does not manage it directly.

Some independent vendors provide end-to-end content delivery hosting services. Customers check builds into source control for the vendor to deploy to their hosting infrastructure.

DATA MODELING

CMS implementations require data modeling, which is the definition of content structures to meet requirements for the solution. Data modeling includes the types of entries (records) and their fields, data validation requirements, relationships between entries, and the information architecture of the website as a collection of entries. Data modeling is a critical factor that can affect the usability, efficiency, and even success of a CMS implementation. A good data model simplifies initial and ongoing development.

TOOLING

Although SaaS systems are more agnostic about technology and customers are unrestricted in their choice of technologies, there are benefits to an alignment between technologies used by vendors and the organization. When choosing CMS, service and content delivery hosting, search, and other vendors, consider their expertise in technologies preferred by the organization.

October 11, 2021

.NET HEADLESS CONTENT MANAGEMENT CONSIDERATIONS

I intend to use this blog post to track technical concerns to consider before implementing a .NET solution that leverages a headless content management system. Some of these considerations apply to content delivery solutions on any platform and some may apply to service-oriented solutions beyond content management. Linked posts tend to focus on .NET and I work for Contentstack, so concrete examples tend to use that product. If you have additional considerations for headless content management with .NET, please comment on this blog post.

— Certain products attract different developer communities. A .NET CMS may attract .NET developers. While vendor selection is important, the technology behind the CMS product itself need not be ASP.NET, as customers should invoke service APIs or SDKs that wrap service APIs rather than involving platform-native APIs directly. Beware that if the CMS is not ASP.NET, existing examples of customizations and code made available from the vendor will likely use a different server-side platform such as NodeJS. You may be able to use ASP.NET Core to host their UIs and services on which they depend, CMS custom user interface components generally interact with the CMS via JavaScript. You may choose to use Blazor (C# run as WebAssembly on browsers) to invoke the CMS vendor's JavaScript or to invoke the service endpoints directly, or possibly to use the vendor's SDK.

- [Hard] Questions for [Headless] [CMS] Vendors – Deliverystack.net
- Considerations for Headless CMS Architectures – Deliverystack.net
- SaaS Headless CMS Architectural Considerations, Part II – Deliverystack.net

— Determine whether and when to use the vendor's SDK, to access vendor services directly, or whether to access data exported from the CMS rather than accessing the CMS directly.

- Whether to Use Vendor SDKs to Access Vendor Services – Deliverystack.net
- Working with [Headless] [CMS] Vendor [.NET [Core]] SDKs – Deliverystack.net
- Why Isn't This In the .NET SDK? – Deliverystack.net

— Determine which data the solution will access at each point in the application lifecycle (build, serve, render).

- Three Major Phases for Data Access in Headless Content Management Solutions – Deliverystack.net

— Implement search indexing and usage strategies. My suggestion is to index everything and access search rather than CMS whenever possible.

— Consider whether and how to employ an event system, for example to translate CMS webhooks to events published to other applications that can in turn invoke CMS APIs. Id, serve, render).

- Advantages of Event Systems with Headless Content Management Systems – Deliverystack.net
- Headless CMS with Event System Diagram – Deliverystack.net
- Integrating Event Systems with Headless CMS – Deliverystack.net

— Consider whether and how you will use ASP.NET MVC, Razor Pages, Blazor, static files, and any other technologies to serve HTML and other content types. Services typically use MVC, but Razor Pages can have advantages over the underlying MVC framework.

- ASP.NET Core Razor Pages and SaaS Headless Content Management Systems – Deliverystack.net

— Determine if and how to implement Jamstack.

- Jamstack and Alternative Architectures for Headless CMS with ASP.NET Core – Deliverystack.net

— As with any ASP.NET application, the URL structure and routing are critical. Determine how you will implement routing to support the URL rules for the solution. Determine if and how what URLs will map to entries. Determine how you will generate links including navigation.

- Data-Driven Navigation with Headless Content Management Systems – Deliverystack.net

— Whether and how to use GraphQL, which constructs queries using proprietary JSON formats rather than proprietary APIs. I prefer to query a search index.

— Implement one or more typed clients to access the CMS.

- Prototype Content Delivery .NET Typed Clients for the Contentstack SaaS Headless CMS – Deliverystack.net

— Go serverless.

- Implications and Considerations for Serverless ASP.NET – Deliverystack.net

— Consider your toolset.

- Cool Tools for [Contentstack] [SaaS] [Headless] [CMS] [.NET] Developers – Deliverystack.net

— Do not forget about robots.txt and sitemap.xml.

— When working with large numbers of entries, beware of paging.

- Paging Through HTTP API Results with .NET – Deliverystack.net
- .NET Core Paging and Threading with Headless (CMS) – Deliverystack.net

— Focus on data modeling.

- CMS Content Modeling Considerations and Resources – Deliverystack.net

— Determine whether to implement entry models or access entry JSON directly. For entry models, determine how field types in the CMS map to .NET types and implement any required infrastructure. Consider exposing the raw JSON as part of the entry model.

- .NET Core Headless CMS Entry Models and Entry Model Classes – Deliverystack.net
- Mapping Field Types to .NET Types in Headless CMS – Deliverystack.net
- Entry Model structs for Markup Fields – Deliverystack.net
- Expose Entry JSON from Entry Model – Deliverystack.net

— Consider a controller, UI customizations, and other techniques to implement conveniences for developers, such as to rebuild search indexes.

- Contentstack Developer Convenience Controller – Deliverystack.net
- Contentstack Developer Custom Widget – Deliverystack.net

— Determine architectures for integrations such as search, commerce, digital experience management, and otherwise.

- SHCDAS: SaaS Headless CMS Integrations and Extensions – Deliverystack.net

— Determine an exception management strategy.

- Contentstack SaaS Headless CMS .NET SDK Exception Management – Deliverystack.net

— Consider a repository abstraction pattern and in any caching situation beware of publication.

- .NET Repository Pattern for Headless (CMS) – Deliverystack.net

April 29, 2021

MAKING THE TRANSITION TO SAAS HEADLESS CMS AND SERVICE ARCHITECTURES

While the concepts are simple, my experience with other systems and architectures presented obstacles to comprehending and adopting SaaS headless content management philosophies and techniques. This blog post intends to provide guidance for architects, developers, and organizations migrating to headless content management systems from other web experience management platforms, though some suggestions are more broad and may apply to any service-oriented architecture. If you have additional perspectives on these topics, please comment on this blog post.

- Use your experience but forget most of what you know about existing content management solutions; envision a completely new CMS architecture intended to truly separate data from presentation and logic.
- Implement applications as aggregations of independent functions (services) rather than monolithic platforms.
- The content management system is not the foundation of the application; the data is the foundation of the application.
- The content management vendor is responsible for managing the content management application and environments.
- The CMS is a UI for managing data.
- All text is JSON; dates, numbers, and other simple values are text.
- All APIs are JSON HTTPS endpoints.
- Focus on data modeling.
- Use the CMS for its purpose: content (generic data) management.
- Build your solutions next to the CMS rather than on top of it.
- The CMS is the body managed by the vendor; customers manage content delivery heads.
- You own the content delivery build processes, application services, web servers, and any content delivery networks beyond those provided for content delivery services.
- All systems are completely unaware of the internals of all other systems, interacting only through webhooks and services.
- SaaS uses webhooks, services, and UI customizations rather than back-end configuration and customization.
- All interactions, including UIs and UI extensions, involve only HTTPS services and webhooks.
- You are responsible for applications that orchestrate services and process webhooks for applications that expose services and process webhooks.
- Use the most standard and common technologies possible.
- Minimize and abstract vendor technologies, use of vendor software development kits, JSON formats, UI extensions, and other proprietary technologies.
- Use only hosted services, SaaS, static files, CDNs, serverless, and other modern technologies.
- Strongly consider implementing an event management system.
- Focus on key objectives such as rapid content creation, deployment, and re-use.
- Reduce expectations and simplify solutions.

- Use independent, purpose-built systems for search, authentication, event management, commerce, stalking/experience management, and otherwise.
- Be prepared for increasing use of JavaScript, including both front-end application integration and server-side frameworks.
- Make some time to invest in understanding and prototyping solutions based on what you consider to be the most common requirements for web solutions.

ADDITIONAL RESOURCES

- CMS and Web Industry Trends, May 2021 – Deliverystack.net
- My Headless CMS Philosophy – Deliverystack.net
- Considerations for Headless CMS Architectures – Deliverystack.net
- SaaS Headless CMS Architectural Considerations, Part II – Deliverystack.net
- [Hard] Questions for [Headless] [CMS] Vendors – Deliverystack.net

May 7, 2021

SAAS HEADLESS CMS AND SERVICE ORIENTED APPLICATION OBJECTIVES AND ADVANTAGES

This blog post describes some of the objectives and advantages of SaaS headless content management systems and service-oriented architectures in general.

Of course anything can be implemented well or poorly, and it is possible to build any solution on any platform. SaaS headless CMS systems endeavor to provide the following advantages relative to other content management systems:

ORIENTATION

While vendors focus on APIs, beyond vendor-specific URL paths and semantics including URL paths, query parameters, as response formats, because all service endpoints represent all data as JSON, customer applications focus on data rather than vendor APIs.

FOCUS

Service-oriented architectures allow each vendor to focus on their core competency, such as content management, search, analytics, experience management, engagement automation, commerce, or otherwise. Customers focus on their solutions that integrate the required services from their various vendors of preference.

ABSTRACTION

Separating concerns requires abstracting interfaces, which allows a component model for application assembly rather than an integrated, monolithic stack that may expose services around it.

CONSISTENCY

Integrating with applications that expose JSON HTTPS APIs and webhooks with simple UI extension frameworks where needed reduces vendor-specific developer and administrator knowledge required to implement and maintain the solution.

CHOICE:

Customers determine the SaaS software vendors, implementation technologies, infrastructure platforms, hosting partners, and other infrastructure components appropriate for their solutions. I prefer ASP.NET Core with WebAssembly, while many developers prefer Jamstack.

SIMPLIFICATION

Simpler solutions present lower risks. Separating a web solution into functions allows services that follow Single Responsibility Principle. Merging subsystems properly reduces complexity and risk.

CLEANLINESS

Service-oriented systems have modern implementations. Clean architectures lead to higher quality solutions that are more efficient to implement, maintain, extend, and replace when needed.

MAINTAINABILITY

SaaS reduces customer responsibility by requiring the CMS vendor to meet service level agreements for hosted content management environments and content delivery services. Rather than building against or on top of shifting platforms and SDKs, customers implement applications next to the vendors applications accessing HTTPS services with stable JSON formats, avoiding the need to synchronize content delivery builds with content management infrastructure.

SCALABILITY

Separating responsibilities allows tuning and scaling each component of the architecture individually.

INDEPENDENCE

Each component of the solution operates independently from the others, retrieving, processing, and returning data as needed.

PERFORMANCE

By separating services from presentation, headless allows independent management including provisioning, deployment, caching, and scaling of services including static files, HTTPS content delivery services, and media content, sometimes including application services such as NodeJS or ASP.NET for server-side logic.

AGILITY

By decoupling presentation (HTML) from both data (services) and logic (JavaScript, HTTPS services), headless improves developer productivity while supporting continuous integration with frequent delivery cycles. Headless can significantly reduce the time it takes for solutions and new features to reach the web.

ADMINISTRATION

Headless CMS vendors have responsibility for provisioning, scaling, upgrading, and otherwise administering and hosting infrastructure and software including a browser-based content management user interface, content (text) CDN, and media (binary) CDN.

FEATURES

Because headless CMS vendors typically upgrade client solutions without downtime, new CMS features can appear after vendor testing cycles are complete, without investment by CMS customers.

TECHNOLOGIES

Customers can choose their preferred platforms, technologies, application platforms, and providers for each service.

INTEGRATION

Headless CMS applies modern web architecture for application integration. CMS vendors provide solutions for integrating with major service providers and other partners. External applications use HTTPS content management and content delivery services. Extensible CMS user interfaces allow custom applications that retrieve data from external systems to present, manage, and reference from the CMS. Headless CMS can trigger webhooks that pass event data to other systems.

TESTING AND TROUBLESHOOTING

Separating concerns simplifies testing and troubleshooting.

RESPONSIBILITY

SaaS places responsibility with the appropriate parties and allows replacing components that serve specific needs.

ACCOUNTABILITY

In addition to reducing customer responsibility, service-oriented architectures provide clear accountability in problem scenarios.

TECHNICAL CREDIT

Moving to SaaS can help not just to eliminate technical debt, but to build technical credit towards future solutions.

ROI AND TCO

Software solutions should always work to increase return on investment and reduce total cost of ownership.

OMNICHANNEL

All CMS can be omnichannel, and most CMS implementations include web solutions. Headless focuses on data structures and APIs rather than page generation, increasing the focus on data as separate from logic, presentation, and use.

CAPITALIZATION

Service-oriented architectures allow customers to leverage technologies such as Jamstack and the tremendous value available in the cloud by assembling applications that orchestrate services rather than building applications on a shifting platform.

RELIABILITY

Many of the factors outlined here, most specifically separation of concerns, leads to greater reliability.

May 20, 2021

CONTENTSTACK SAAS HEADLESS CMS CONTENT DELIVERY DIFFERENTIATING FACTORS

This blog post describes factors that I believe differentiate Contentstack from competing SaaS headless CMS products considering only features that affect content delivery. I tried to focus on features that are not present in almost every SaaS headless CMS. Contentstack is not the only vendor that provides each of the features listed here, although I have not seen another product with anything like Contentstack's modular blocks. This is my perspective based on my experience. If you are aware of other vendors with comparable or additional features, please comment on this blog post.

FIELD TYPES

Contentstack supports a good balance of field types (at least asset, Boolean, checklist, datetime, dropdown, reference, HTML, link, markdown, multiline, number, radio, and single line) relative to other CMS that I have encountered, and it is extremely easy to implement custom field types.

MULTIPLE VALUES

Developers can specify that any field can contain multiple values, for example to associate several dates with an event.

GROUPS

Contentstack allows grouping fields to separate them visually in the user interface and structurally in JSON representations. Groups can also repeat.

GLOBALS

Contentstack content types can use groups of fields defined at a global level, such as the fields to capture basic metadata for every type of page in a website. These globals reduce developer effort while making the CMS UI and JSON representations of different types of entries more consistent.

MODULAR BLOCKS

I think that modular blocks are Contentstack's most significant differentiating factor when it comes to data modeling, which is the critical element of any CMS implementation, including for content delivery. Rather than modeling each individual collection of data elements as a separate entry, modular blocks allow developers to implement flexible structures as fields nested within entries, balancing CMS usability and developer productivity against excessive data type normalization and allowing page modeling rather than page composition. Modular blocks can reduce the overhead of merging JSON from multiple sources, especially when each entry requires a separate HTTP call.

SDKS

Some CMS that may market themselves as decoupled or even headless are relatively tightly coupled by requiring extensive use of vendor SDKs that may be available for only a subset of technology platforms. Contentstack provides SDKs that wrap its HTTPS APIs for developers to use with common mainstream technologies such as .NET Core, Java, PHP, NodeJS, and others, and you can call the APIs directly with any of these or with any technology that can speak JSON over HTTPS.

June 17, 2021