Michał Mitas Oct 17, 2025 259 (1 votes)

Going Headless: Making the Right Architectural Choices

Earlier this year, I began a series of articles about headless architecture:

- Going Headless: Optimizely Graph vs Content Delivery API
- Going Headless: On-Page Editing with Optimizely Graph and Next.js

Today's post continues that series. Having just completed the launch of a new website built with a headless architecture, I'd like to share key observations, strategic considerations, and my own perspective on the strengths and challenges of the headless model. My goal is to help you make confident, informed choices for your own future project architecture.

What This Article Covers

In today's post, we'll start with the fundamentals:

- What “headless” really means in the Optimizely context
- Why some choose it (and why some shouldn't)
- Choosing the right architecture for your headless Optimizely setup
- A high-level architecture overview to ground future discussions

What “Headless” Means in the Optimizely World

In a headless Optimizely setup, the CMS focuses on storing and structuring content and exposing it via APIs. Whether you use a Content Delivery API or Optimizely Graph, the “head” (the UI) is a separate app that fetches the content and renders it.

When Headless Might Not Be the Right Choice

The headless separation lets the front-end team ship without touching the CMS, and enables you to serve multiple channels from a single content source. While this flexibility is appealing, headless isn't always the right fit for every scenario.

When headless may not be necessary:

- **Simple site needs**: If you only require a basic site, headless can introduce unnecessary complexity.
- **Limited developer capacity**: Headless architectures require a skilled front-end team comfortable with API-driven development.
- **Budget constraints**: Having two applications means you need to host two applications. This can impact your infrastructure or require additional licences and potentially higher costs.

Key challenges of going headless:

- **Routing and URL management**: You are responsible for route generation and resolution (slugs, dynamic routes, canonical URLs, redirects, and localization-aware paths). With Optimizely Graph, there's no built-in URL resolver, so you must design and maintain this logic on the front end.
- **On-page editing and preview**: Traditional on-page editing from MVC is not available by default. You'll need to implement preview rendering of drafts, signed preview tokens, and live refresh hooks. If editors require a visual page preview, you'll have to build or integrate that experience. As a starting point you can follow my guide - Going Headless: On-Page Editing with Optimizely Graph and Next.js

- **CI/CD complexity**: Headless usually means separate pipelines for the CMS and front end, so you need to coordinate deployments and keep everything in sync. Additionally, managing schema changes in Optimizely Graph introduces further challenges, as updates must be reflected and tested across both applications.
- **Authentication and authorization**: You'll need to handle user logins, editor previews, and protect your APIs. This means managing secrets and reviewing security.

As you can see, adopting a headless approach involves trade-offs and added complexity. Make sure your decision is guided by genuine business requirements, rather than simply following industry trends.

When Headless Is a Way To Go

Opting for a headless architecture can be a game changer for organizations that need agility, omnichannel delivery, and modern development workflows. If your content needs to be reused across multiple front ends—such as websites, mobile apps, IoT devices, or digital kiosks — headless streamlines the process and reduces duplication. It enables independent development and deployment of the frontend and backend, letting teams iterate quickly and select best-in-class technologies for each layer. For companies with demanding editorial workloads, headless can provide enhanced flexibility, customizable authoring experiences, and the ability to scale globally by leveraging CDN-backed static sites or API-first delivery. If you need powerful integrations, advanced personalization, or the latest in frontend performance tooling, headless often opens more doors than traditional approaches. Ultimately, headless is especially valuable when your project requires future-proofing, composability, and the freedom to evolve both your frontend and backend at their own pace.

Choosing the Right Architecture for Your Headless Optimizely

When picking your setup, balance short-term delivery with long-term flexibility. Consider who you're building for, which channels you must support, your team's skills, latency/scale needs, security/compliance, and total cost of ownership. Write these down up front-they'll keep trade-offs honest.

Technical decisions

These days, the variety of tools and architectural options can feel overwhelming, so it's impossible to cover every scenario here. Instead, I'll highlight some key decisions I've faced recently and briefly explain each. Hopefully, these insights will be helpful.

CMS PaaS vs SaaS

PaaS offers greater control over your codebase and infrastructure, but comes with increased operational responsibilities. SaaS minimizes maintenance and accelerates upgrades, though it restricts server-side customization and hosting flexibility. Your choice should depend on your need for custom server logic and regional deployment control. For our project, we selected PaaS to retain full control over our source code and to support future integrations.

Optimizely DXP vs self-hosted

DXP gives you managed hosting, a global CDN, built-in security, and guaranteed SLAs right from the start. So a lot of the heavy lifting is taken care of for you. On the other hand, self-hosting might be the way to go if you have strict data residency requirements or unique networking needs, but just keep in mind that you'll be responsible for monitoring, scaling, patching, and handling any incidents yourself. Be sure to realistically weigh the operational costs of each option before making your choice.

Graph (GraphQL) vs CDA (Content Delivery API)

Optimizely Graph shifts content delivery to a managed service, enabling precise GraphQL queries-ideal for retrieving deeply nested or specific content-and offers advanced search capabilities. In contrast, the Content Delivery API (CDA) is simpler, REST-based, and provides built-in URL resolution, but may require additional effort for complex content structures or scaling needs. For a detailed comparison, see my article: Going Headless: Optimizely Graph vs Content Delivery API.

For our project, we chosed Optimizely Graph because its data residency met our requirements and it provided greater flexibility for content delivery.

Next.js vs other front-end frameworks

Next.js offers SSR, SSG, and ISR out of the box, making it a strong choice for content-rich sites. Alternatives like React with Vite or Angular are also solid. Choose the framework your team is most comfortable with and that aligns with your hosting environment (Node, Edge, or serverless).

For our project, rapid time to market was essential. We selected Next.js together with Vercel for its seamless support of SSR/SSG/ISR, integrated Turbo monorepo tooling, and a deployment workflow that enabled us to iterate quickly, gather feedback, and launch efficiently.

If you're using the SaaS version of Optimizely CMS, you may want to explore the newly introduced DXP hosting for front-end applications. Learn more in the official documentation: https://docs.developers.optimizely.com/content-management-system/v1.0.0-CMS-SaaS/docs/host-a-front-end-with-optimizely

GitHub vs Azure DevOps

Pick the platform your team knows and that fits your cloud targets. GitHub pairs nicely with Actions and Vercel; Azure DevOps integrates tightly with Azure and enterprise governance. Both support trunk-based development, protected branches, and good CI/CD.

We chose GitHub because we didn't need the additional features offered by Azure DevOps, and hosting our app in DXP meant Azure integrations were minimal. GitHub's seamless integration with Vercel was also a significant advantage for our workflow.

Opti ID vs other SSO providers

Use Opti ID where it makes sense within the Optimizely ecosystem. For your site/app users and internal editors, standard OpenID Connect/SAML providers (Azure AD, Okta, Auth0, etc.) work well. Confirm MFA, provisioning, and claim mapping early.

We selected Opti ID because it met all our requirements and enabled us to implement authentication quickly and efficiently. For more details, refer to the official documentation: https://support.optimizely.com/hc/en-us/articles/12613241464461-Get-started-with-Opti-ID

Common Repository vs Separate Repositories

In a headless setup, the frontend and the CMS/backend are independent. You can keep them in one repository (monorepo) or split them into separate repositories.

- Monorepo (one repository):
  - Pros: single place for code and tooling, atomic changes across front end and CMS, easier end-to-end testing and shared packages
  - Cons: more complex pipelines, risk of unnecessary deploys without proper path filters, larger repo to maintain
- Separate repositories:
  - Pros: clear ownership and access control, simpler pipelines per app, easier to replace one part without touching the other
  - Cons: coordination overhead (shared contracts, schemas, SDK versions), more cross-repo communication

Opt for a single repository when teams need to collaborate closely, share code, or coordinate changes across frontend and backend. Separate repositories are preferable when independent deployments and isolated responsibilities are a priority.

We chose a monorepo approach because our team size makes close collaboration easier, and it gives us greater control and visibility over the entire CI/CD process.

Summary

In summary, successful modern web architecture is about making the right choices for your team, business needs, and long-term maintainability. Each architectural decision, whether it's PaaS vs SaaS, managed services vs custom hosting, or selecting the front-end framework should reflect your unique requirements in performance, scalability, flexibility, and collaboration. By evaluating trade-offs at each level and prioritizing seamless integration, you can build solutions that are robust, adaptable, and future-proof.

Oct 17, 2025

Comments

Mark Stott Oct 20, 2025 01:36 PM

Hello Michal,

This is a great post on when/why you should go Headed or Headless and PAAS or SAAS. I have a couple of thoughts off the back of this which are worth considering:

Hosting Options wise there is also Hybrid, this is where you potentially host your main channel in a traditional headed fashion while allowing secondary content channels to consume the same content in a headless fashion. This could be an option that gives you the best of both worlds.

Another benefit of PAAS (irrespective of headed/headless) is that you can leverage extensions for how your index your content. You could for example do the following:

- Create a custom implementation of "Full Text" search where you strategically omit some parts of your content in terms of full text search.
- Create a custom implementation of IContentApiModelProperty to strategically add or transform data within the CMS before it arrives in Graph to be more useful to the end consumers.
- Use inheritance in your content models, this can result in smaller less complex graph queries that don't have to rely on as many fragments.

Michał Mitas Oct 21, 2025 02:38 PM

Hi Mark,
It's great to see you here! I really appreciate your comment and thoughtful input! I completely agree with your points.
Thanks for contributing to the discussion!

Please login to comment.

Latest blogs

[How to Add Multiple Authentication Providers to an Optimizely CMS 12 Site (Entra ID, Google, Facebook, and Local Identity)]
Modern websites often need to let users sign in with their corporate account (Entra ID), their social identity (Google, Facebook), or a simple...
Francisco Quintanilla | Oct 22, 2025 |

[Connecting the Dots Between Research and Specification to Implementation using NotebookLM]
Overview As part of my day to day role as a solution architect I overlap with many clients, partners, solutions and technologies. I am often...
Scott Reed | Oct 22, 2025

[MimeKit Vulnerability and EPiServer.CMS.Core Dependency Update]
Hi everyone, We want to inform you about a critical security vulnerability affecting older versions of the EPiServer.CMS.Core package due to its...
Bien Nguyen | Oct 21, 2025

[Speeding Up Local Development with a Fake OpenID Authentication Handler]
When working with OpenID authentication, local development often grinds to a halt waiting for identity servers, clients, and users to be configured...
Eric Herlitz | Oct 20, 2025 |

See all blogs

Learning resources

Partner portal
Developer guides
User guide
Video tutorials
GitHub
Optimizely Academy

Contact

Optimizely support

See the support page for contact information.

Feedback on Optimizely World

If you'd like to give feedback about this site, please **click here**. Thank you very much!