netlify + Jamstack

CASE STUDY

Monolith to Microservices: How to Incrementally Migrate Your Web Apps

A step-by-step guide to
enterprise Jamstack migrations

INTRODUCTION

Migrating to a new web architecture can be a daunting task. And if you're migrating off a monolith, it can seem near impossible. But your path to the modern web might be closer than you think.

Our team at Netlify helps all kinds of companies migrate to the Jamstack, empowering developers with streamlined workflows and performant sites. In this guide, we'll walk you through how we approach these migrations, and arm you with the questions, checklists, and common gotchas you need to be prepared for at each step of the way. We'll cover the barriers to migrating as well as the benefits of the Jamstack model, and how to select your tools and plan your migration.

Ready? Let's get started.

SECTION 0

Understanding Jamstack Architecture

Understand Jamstack Architecture

A successful migration needs to ensure that any new site architecture fulfills your team's requirements. That's why, before we dive into the steps necessary to undergo a migration to the Jamstack, it's important to understand core Jamstack principles, from the mechanics of the Jamstack, the workflows it enables, and the ways of building things that it unlocks.

What is Jamstack?

Jamstack is a way of thinking about how to build for the web. The UI is compiled, the frontend is decoupled, and data is pulled in as needed.

Decoupling the frontend from the backend and serving infrastructure breaks that direct relationship between the number of requests being made for your site, and the work that your infrastructure needs to do. We'll see that this brings all sorts of benefits, from easier scaling to more confident workflows.

Why choose Jamstack?

Better Performance

The significant work done by frontend developers to deliver great performance can be undermined when it is either inefficiently rendered on servers at request time, or compromised by a tightly coupled, opinionated backend. Instead of waiting for pages to build on the fly, Jamstack sites generate at deploy time. When it comes to minimizing the time to first byte, nothing beats pre-built files served over a CDN.

Higher Security

With server-side processes abstracted into microservice APIs, surface areas for attacks are reduced. You can also leverage the domain expertise of specialist third-party services.

Cheaper, Easier Scaling

When your deployment amounts to a stack of files that can be served anywhere, scaling is a matter of serving those files in more places. CDNs are perfect for this, and often include scaling in all of their plans.

Better Developer Experience

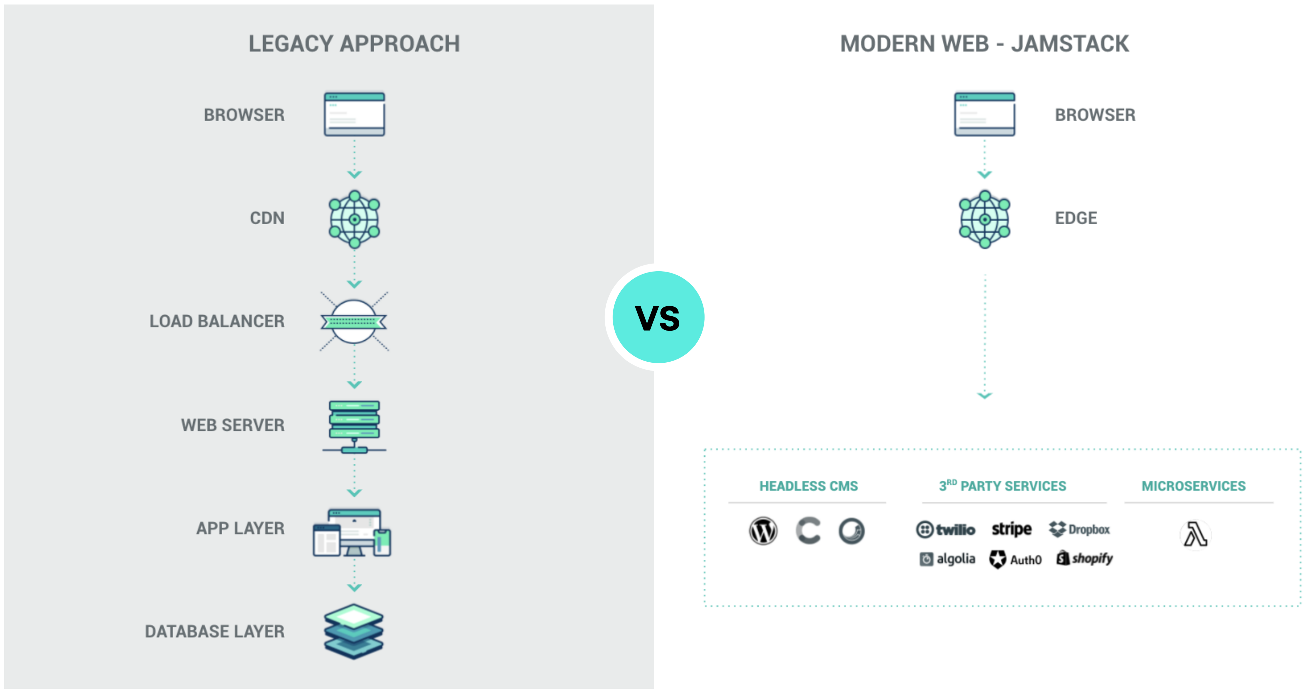
Loose coupling and separation of concerns allow for more targeted development and debugging.

Collaborative Workflows

Web development is a team sport. Jamstack is git-based, and with collaborative tools and integrations with popular project management tools, Jamstack teams can create a tight feedback loop that works with developers' workflows. Important details are less likely to get lost, and everybody wins.

How does the Jamstack work?

The "Jam" in Jamstack is derived from its three components: JavaScript, APIs, and markup. Web pages for a Jamstack site consist of standard markup language, so they can be built and tested anywhere, without request-time dependencies on app servers or server-side technologies like Node.js. With browser APIs and JavaScript runtimes maturing, it has become viable to deliver rich interactions with client-side JavaScript which enhance experiences with custom logic and secure calls to APIs.



Innovations that Power the Jamstack

The concept of static websites is an old one. In fact, the traditional Unix model of a client and server made up the first days of the pre-web. In the early web around the 1990s, web developers built static sites. As the usage of the internet blossomed from educational to personal to business use, the web evolved into meeting more dynamic needs, and the next phase of the web was born. Some call this web 2.0—a more dynamic web (a client, web server, app server, and database) in the early days of the internet in the 2000s that we see through today.

However, with more dynamic demands on websites, also came a variety of complexities, which had impacts on everything from costs to security to speed. Demand grew for a more simple and powerful way to build for the web, independent of the complexities created by monolithic applications and managing infrastructure.

Key changes in the way we all began to work aligned to spark change that set us up as an industry for a more efficient, modern web.

Cheaper, Easier Scaling

- **Git:** The popularization of git–a distributed version-control system for tracking changes in source code during software development–led to significantly improving the way we develop collaboratively.

- **Continuous Integration/Continuous Development:** The universe of software build tools arrived for automation and CI/CD, helping improve the integration, testing, delivery and deployment of an application throughout its lifecycle.

- **Static site generators:** SSGs started being used for efficiency–a script that takes in data, content and templates, processes them, and outputs a folder full of all the resultant pages and assets.

- **Expanding browser capabilities:** The browser evolved into a full-fledged operating system.

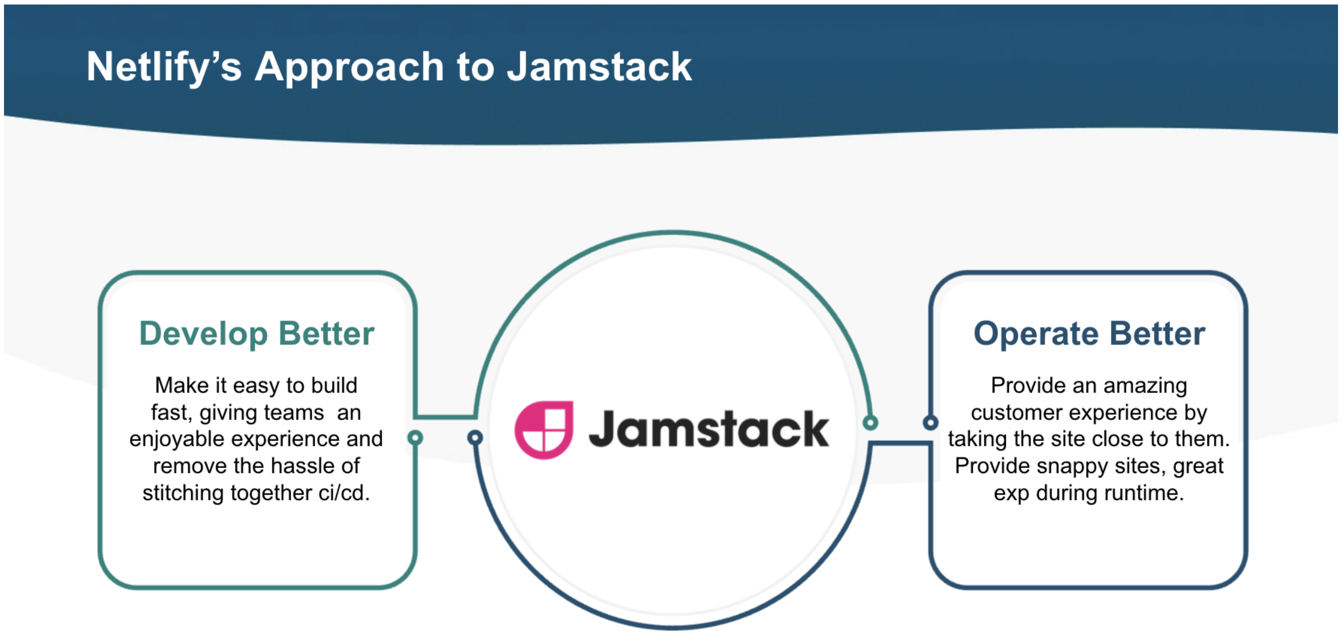
- **API economy:** The growing API economy across clouds unlocked powerful capabilities for frontend developers, who could integrate third-party services and plugins like Stripe, Algolia, and more.

The Jamstack takes advantage of these new practices to create performant sites with simpler workflows. This web architecture prebuilds the site and serves it on a CDN. When set up the right way, will bring all the benefits of speed and security for the web apps.

Netlify and the Jamstack

There are a lot of moving parts to consider while building a Jamstack application, like the edge network and cache invalidation, reliable infrastructure to build the site, coordinating domains and DNS records for go live.

The process of manually stitching together deployment pipelines, build tools, edge networks, object stores, API runtimes and so on, was an inherent barrier that we had to remove to make the Jamstack a truly viable approach. That's where Netlify comes in.


Netlify's Approach to Jamstack
Develop Better
Make it easy to build fast, giving teams an enjoyable experience and remove the hassle of stitching together ci/cd.
Jamstack
Operate Better
Provide an amazing customer experience by taking the site close to them. Provide snappy sites, great exp during runtime.

Netlify aims to provide a first class developer experience for teams building Jamstack apps, while providing the infrastructure needed to build and host these web apps.

Contact Sales

SECTION 1

Audit Your Web Apps

Audit Your Web Apps

Before touching any part of the codebase, the next step of a Jamstack migration is to audit your existing web applications. If you want to migrate to a new stack, you need to know exactly what you are migrating from: what functionalities of the toolstack your team is using now, and what functionalities they need for the new site.

When auditing your web apps, we recommend thinking about them in three separate buckets:

- **Architecture and Web Tooling Audit**
- **Site Stakeholder Audit**
- **Pain Points Audit**

Architecture and Web Tooling

Let's start with the technology. Monolithic applications package a lot of functionality together, which can make it less apparent to understand how you currently build your site, where it is hosted, and what tools you use to build it. Jamstack sites, on the other hand, decouple the frontend from the backend. This allows for more modular development where teams can take advantage of the huge ecosystem of third party tools to provide website functionality. But it also means that teams migrating off of a monolith need to proactively select tools to replace the functionality that their monolith ran out-of-the-box.

Note: Scope your solution to your needs. Many platforms advertise huge lists of features, which is why they get bought in the first place. But seldom or they actually all successfully used. Migrating to a microservice architecture means you have the opportunity to assess which of these features and capabilities your sites and applications need.

Say you have an ecommerce app built on a monolith, then you will have to account for the fact that the individual services like payments, checkout flow, storefront, fulfillment will potentially have to be broken into standalone services while your frontend is also decoupled and built statically. While this is one of the main reasons people migrate to the Jamstack (more flexibility in their tooling), it also means you need more upfront planning.

In terms of reevaluating, some things to consider are listed here. Once you have answers to these questions, you can determine which parts to keep and which ones to change up.

- **Is the entire app a monolith or is it decoupled?**
- **What languages and does your architecture demand of your team?**
- **Is the code version controlled?**
- **Where is code executed?**
- **Is a CMS like Wordpress/Drupal used to build the site?**
- **Where is it hosted? Are you using a CDN?**
- **Do you have a firewall?**
- **What are the key features in use with the current platform?**
  Does the current platform have any gaps or opportunities to streamline?

Note that a Jamstack migration doesn’t always require swapping out tools! Lots of people like Sitecore as their CMS, and find ways to “Jamstack-ify” the tool, still using it for content editing, but decoupling it and serving as a pre-generated site.

How European shopping mall leader embraced lightning-fast Headless Sitecore (with predictable cloud spend) on Netlify

Faced with the challenges of the digital revolution and transformative shifts to the shopping experience, European mall operator Klépierre decided to undertake a massive overhaul of their web presence. Their existing Sitecore architecture worked well for content authoring, but Sitecore's heavy infrastructure didn't allow for the lightning-fast experience their customers expected.

Klépierre engaged the creative and technology agency Proximity to help undergo a global platform redesign, and develop 98 sites in just six months. Sites needed to be fast while still image-heavy, easy to use for Klépierre's many stakeholders and content authors, and the entire project needed to happen within the agreed budget and timeframe.

After encountering performance and price barriers with their planned Sitecore redesign, Proximity made a discerning shift to their planned technical approach. With only 3 months left in the project, they rearchitected on the Jamstack model to dramatically cut their Azure costs. With a headless Sitecore CMS and Netlify as the distributed edge network (an advanced CDN), they massively reduced cloud spend while still delivering performant sites on the client's timeline.

While the initial impetus for the move to the Jamstack was cost-savings, the Proximity team soon realized additional benefits of this decoupled Sitecore architecture. Moving to Netlify also meant improved workflows, simpler cloud management as well as great website performance.

Site Stakeholders

The next thing you need to understand for a successful migration are the people involved. Web development is a team sport. There are people who build the site, design the site, create new content, and those who maintain it and help scale. and identifying the players will help ensure buy-in and a smooth transition. the people who are involved in the site - be it building, maintaining or creating content, helping scale.

Typically different teams care about different aspects of the site. Understanding how the different teams work together and each one's workflows helps in maintaining their needs even on the Jamstack.

Questions to ask:

- **Who builds the site? Who deploys the site? Who maintains the site?**
- **Are multiple engineering teams involved, like frontend, backend, and DevOps involved? If so, how do they all work together?**
- **Who is the business owner of the site?**
- **What do the different teams care about?**

Understand Your Current Friction Points

Now that we know what tools and people are involved, let's do a complete analysis of what is working and what isn't in the current site.
Here are some questions to ask yourself:

- **Site speed**: How performant is your site?
    - Is the site fast in some regions, but slow in others?
    - Is this because you are serving all your customers from a server in a single region?
    - Are the frontend engineers impaired by the backend technology or can they exercise all of their skills without compromise?

- **Time to market: How long does it take to ship new features, or changes to the site?**
    - Are the devs not able to ship features fast enough because of the dependency on multiple teams?
    - Frontend teams will build out the app or pages, but have to depend on Devops to actually spin up environments for testing, staging and qa. This can lead to a huge delay in shipping product updates to the site, sometimes even up to 3 months.

- **Development workflow deficiencies:** What isn't working, and why were these processes set up like this in the first place?
    - If multiple teams are involved in building a site including frontend, and devops, what does the handoff look like? Is that ideal?
    - Who is paged when the site breaks? If there is no devops team, how much time is spent by the frontend team on cache invalidation, scaling CDN, servers etc?

Migrating to a new stack might be a chance to address these pain points, but only if you highlight them to begin with.

SECTION 2

Technology and People

Technology and People

Now that you know why you want to migrate, the next step is to determine which web property you will start with, finding the right people to build it as well as the infrastructure to build and host it.

Which property will you move first?

As part of our Enterprise engagements, we work with teams to understand the different properties in play for the migration, and categorize them appropriately:

- **Content-heavy sites**
  Examples: This includes campaigns, landing pages and .com sites that are not primarily delivering deep interactions and transactional functionality.

  Sometimes, content-focused sites provide a low risk project to prove out ROI of the Jamstack is the way to go. This way you can use this to prove the value to stakeholders and gain their trust to keep moving forward.

- **Web apps**
  Examples: Anything that requires functionality like login, a database, user management, and has dynamic functionality at runtime.

  The key to migrating an app with functionality will be to decouple the frontend from the backend, but also break different services into their own standalone applications. You'll need to use APIs, either your own or 3rd party services like Algolia for search, or Auth0 for authentication and user management. Here, auditing the 3rd party APIs you use will come in handy.

- **Large sites with very high volumes of pages**
  Examples: Publications; Ecommerce sites with 100k + products and pages

  Migrating a large site in one go might seem daunting. Here, what we have seen work is to break the site down into sections. These could be logical sections based on who owns it, like docs, community, product etc. or functional sections based on how often they get updated. Say, the press release section is its own microsite so that content updates can go live instantly.

  This is a whole other topic that we;ve have written about in a blog post: How to build large sites on Netlify.

Based on your team's familiarity with the Jamstack, proficiency with building decoupled web apps and the goals for your site, we determine which of your web properties to start with. For example, it might make sense to migrate your docs before you migrate your web apps.

Determine who will build it

The main team players in a Jamstack site might look a bit different than in your previous architecture.

That's because the Jamstack empowers frontend developers. With the Jamstack, front-end code is no longer limited to being a product of a back-end system.

With the Jamstack, your developers are required to have less exotic proprietary skills, and can draw from the larger, more portable, and more available general modern front-end developer community. This makes their skillsets transferable and also very relevant in today's modern web development scene. It also means you should confirm that you have a development team that can support the migration and is comfortable with JavaScript, APIs, and markup.

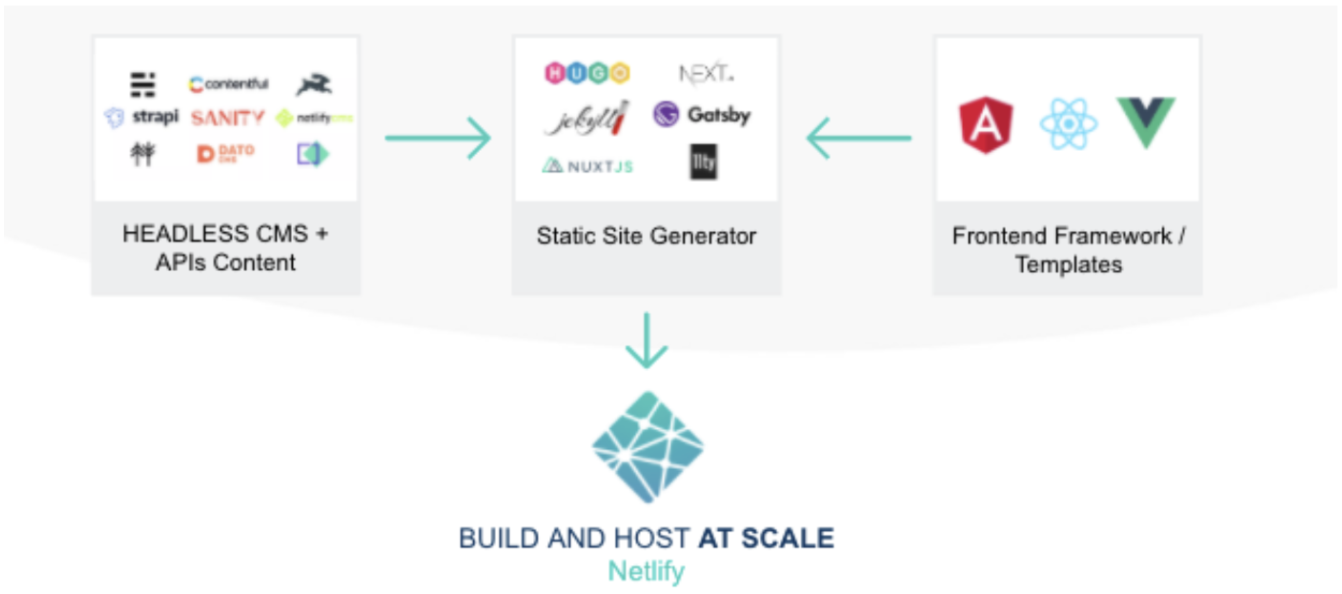
Determine your tech stack

What you need to build a Jamstack app is pretty different from what you will need for a monolithic app. Decoupling is the name of the game, so everything from code, to content and building/hosting are separate

The main ingredients of a Jamstack site are:

1. **The code that lives in a Git provider**
2. **An abstracted source of content, often a headless CMS**
3. **A static site generator that will pre-build the site into static assets**
4. **A platform/infrastructure that will help you build and host the site**

While these are the broad areas, there are a lot of options when it comes to choosing each of these. Let's quickly walk through some key points to consider for each.



How to Choose a Static Site Generator (SSG)

Next, you'll need to choose how you generate your site. Choosing a static site generator goes back to the kind of app you are building, and there are a couple different types of sites we think about here (same as on page XX):

- **Content-heavy sites:** Finding a tool which prioritizes generating pages and URLs might be your best choice. Jekyll, Hugo, and Nuxt are examples here
- **Complex, app style functionality:** Gatsby and Next.js are good choices because they come with more advanced, client-side features ready-to-go
- **Large Application:** If you have larger sites, then you will have longer builds. Hugo and 11ty have incredibly impressive generation times, and Netlify's On-Demand Builders functionality can also assist with reducing build times.

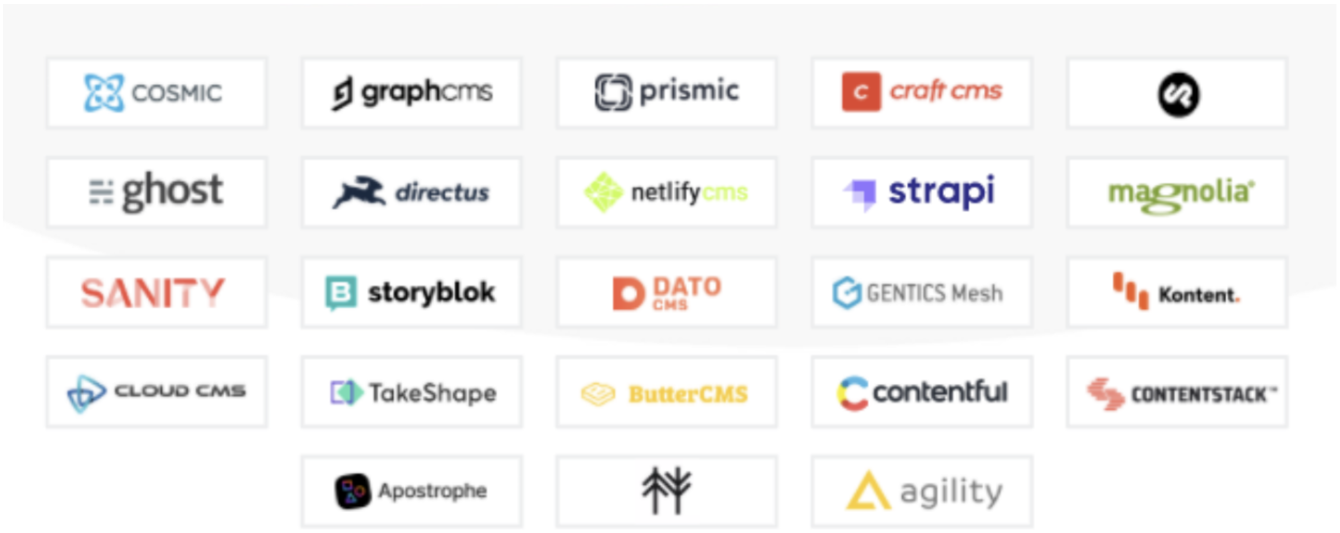
Another big factor to weigh when choosing an SSG is your dev team's proficiency and language of choice. Since the Jamstack ecosystem is so large, there is a static site generator for most languages out there.

Choosing your static site generator can be closely linked to the infrastructure that it will run on. Not all build and hosting environments are tech agnostic. Netlify is tech agnostic, and can run any SSG on it--or even multiple SSGs. Netlify gives your teams flexibility to build in the language they are comfortable with while also giving them room to try out the best framework at any time.

For instance, the Netlify marketing site (www.netlify.com) is an 11ty site, while the Netlify app (app.netlify.com) is a React application. They both are built on the same platform and benefit from the same workflows, while being two separate apps owned by different teams.

This is extremely crucial because as your teams grow, it is possible some teams might lean towards Gatsby, while others choose Next.js, Nuxt, or a new framework for whatever they're building.

Tip: *Jamstack.org has a helpful [tool for finding open-source static site generators](#) by framework or language.*



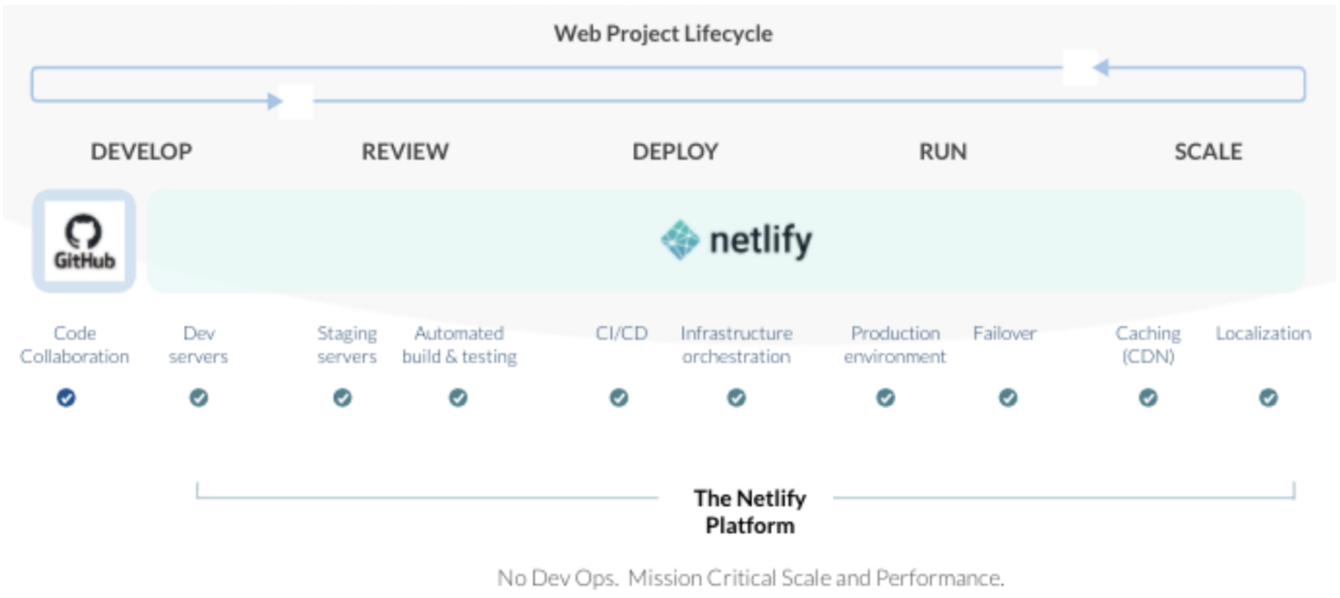
Most teams already have a content management system in place, whether that's a headless CMS or a monolith. While a migration is a good time to evaluate whether you need a new headless CMS, it's also possible to continue using the headless versions of your older CMS. A lot of the monolithic CMSs like WP, Drupal and Sitecore now have a headless version that can be used with Jamstack apps. This not only reduces the migration work, but also gives your content teams the workflows they are used to while gaining the performance benefits of the Jamstack.

Choose where you will build and host

This is one of the most crucial steps that your team will have to take as you make the migration process. Based on the goals for your site, you'll want to find a platform that not only provides the infrastructure, but *maintains* it. makes it easy for developers to build and deploy sites globally, and also build apps and sites in any framework of their choice.

All other primitives needed to expand scope of your web properties like build automation, serverless functions, and redirects are all part of the platform to help build complex web apps at scale.

Content Source: Headless CMS



At Netlify, we focus on empowering the frontend developer to build sites by pushing code to Git, while managing the rest of the review/deploy workflow for them. We provide a platform that will take care of the build and hosting infrastructure, scaling, and maintenance for your teams, so teams can focus on building the site.

SECTION 3

The Migration

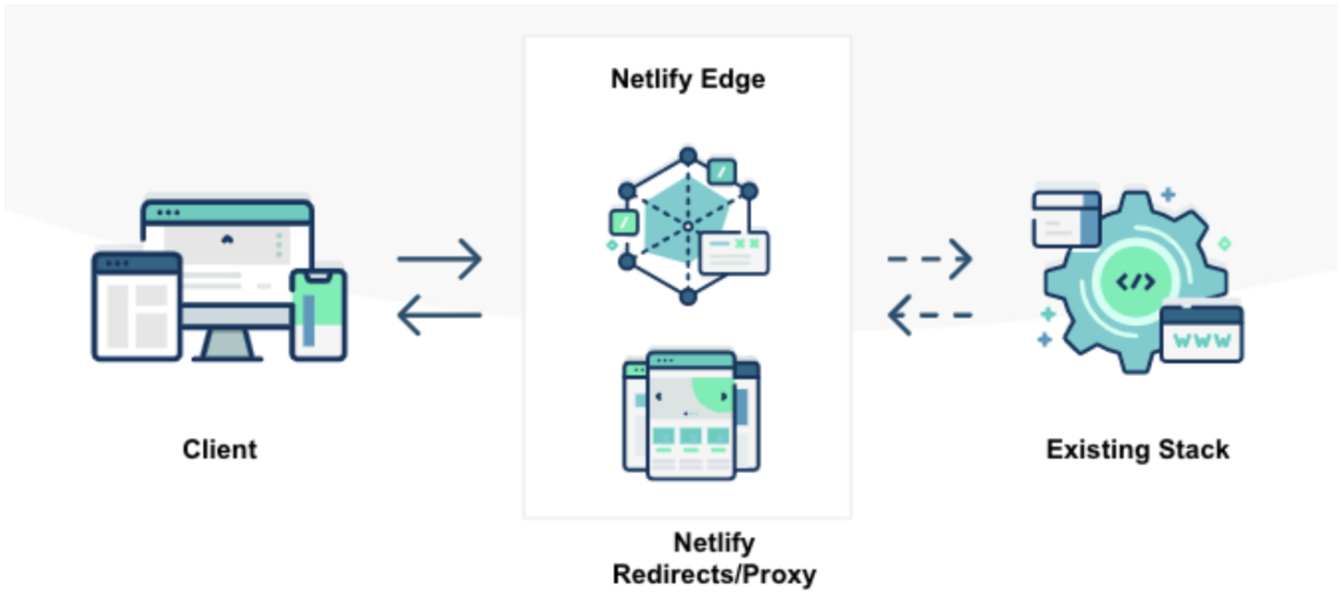
The Migration

Finally, we've selected the tools, and are ready to commit some changes.

You do not have to migrate your entire site on one day and do a "big bang" migration. Based on your timelines for migration, we help customers adopt the Jamstack gradually, even page by page.

In fact, about 90% of the enterprise customers the Netlify team has onboarded have migrated to the Jamstack incrementally. This is a low risk move while also getting the benefits of speed for the Jamstack section of your site.

Understanding Incremental Migrations

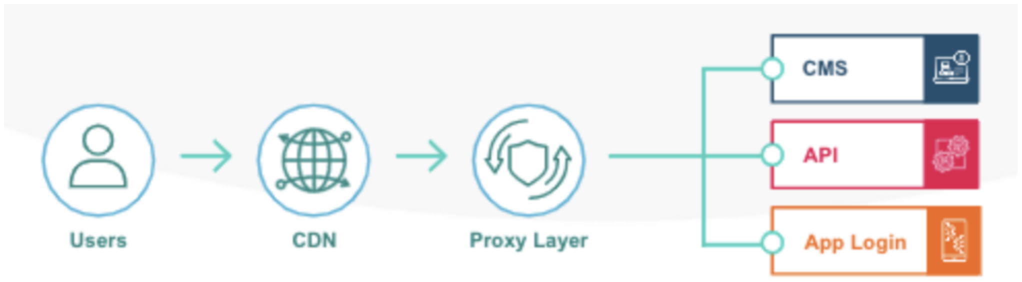


Incrementally migrating to the Jamstack on Netlify is possible thanks to two things:

- **Netlify redirects**
- **The proxy API**

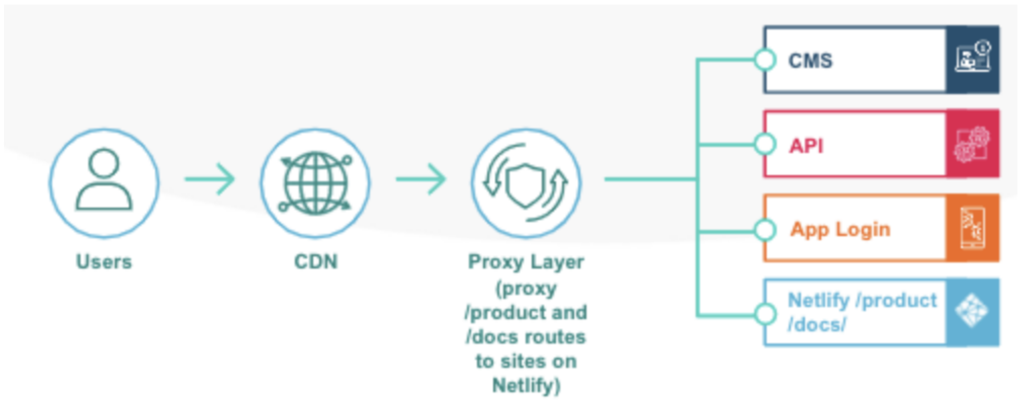
Netlify provides access to highly optimised edge-based routing. This means that it is possible to direct all of your site traffic to Netlify at the DNS level, and for Netlify to invisibly and rapidly route every request to your existing platform unless it can satisfy the request from resources it hosts as a Jamstack site. This way, content on Netlify will be served by the Netlify CDN, and any other paths will be proxied from your existing stack providing a zero downtime, gradual migration as an additive approach.

Phase 0: the legacy stack



Phase 0 would look something like this, where your site is on a legacy stack and probably has a CDN and proxy layer to route your user traffic to the website, the application or other services.

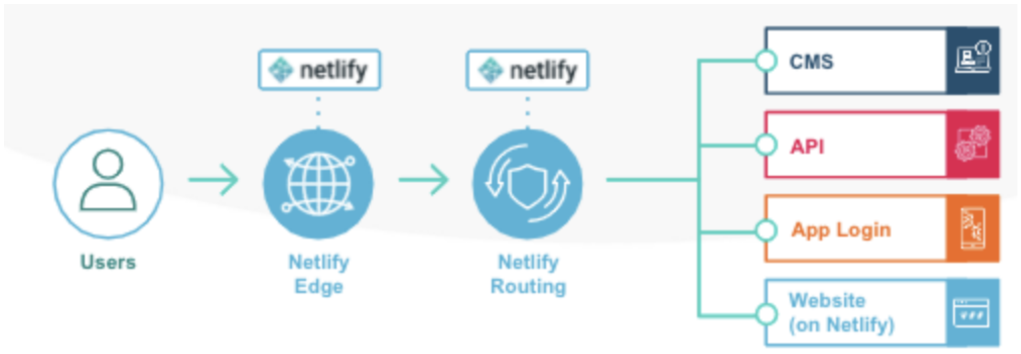
Phase 1: Migrate Pages



Users
CDN
Proxy Layer (proxy /product and /docs routes to sites on Netlify)
CMS
API
App Login
Netlify /product /docs/

As you build certain pages or sections on Netlify, they will now be hosted on our Edge Network. Now when users hit those routes, say /product and /docs, they will be routed to the sites on our edge via the proxy layer.

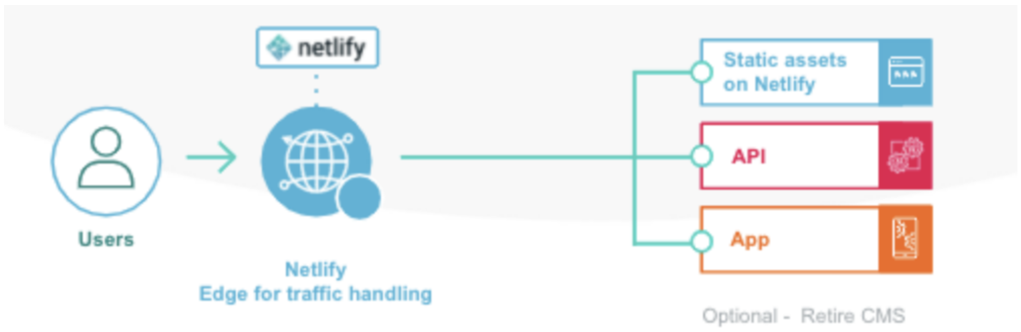
Phase 2: Migrate Logic



As more of the site is built on Netlify, we highly recommend you add all your business logic to our Edge including redirects and proxies to give your end users a uniform experience on your domain.

Phase 2 involves building and hosting your web properties on Netlify. Parts of the site that are built on Netlify will be served by us and any parts that we cannot serve, we proxy that to your legacy infrastructure from our Edge. This removes the need for your existing CDN as the Netlify Edge will take over.

Phase 3: Retire the Monolith



Finally, you can retire your legacy monolith completely after completing the Jamstack migration of your site and decoupling the rest of your backend services.
Migrating DNS for a site that's already in production doesn't need to be terrifying.

Related resource: Live DNS site migration checklist

CONCLUSION

Our technology choices have impacts. The way we choose to architect our sites--and the tools we select to build them--affect site performance, the functionality we can bring to bear, users' experiences, and the workflows your team uses to build and edit the site.

It's no wonder, then, that developers, content editors, designers, and product managers alike are so opinionated about the tools we use to build web applications.

That's one of the reasons people are choosing Jamstack. It's an iterative, flexible approach that allows teams to select the tools they want, and unlock productive workflows for your frontend teams.

We'll include the questions you need to ask, and the common gotchas, at every step of the process, to ensure that they bake in the Jamstack best practices right from the start, while also ensuring the migration and go live from the old to new stack is extremely smooth and risk free.

Contact Sales

netlify