Three Steps to Move to the Modern Web

A GUIDE TO MIGRATING WEB APPLICATIONS TO THE JAMSTACK WITH NETLIFY

netlify

Overview

Migrating to a new web architecture can seem daunting, especially if you're migrating off a monolith. Your path to the modern web might be closer than you think.

Netlify helps companies migrate from the legacy stack to the Jamstack, to deliver all the benefits of speed, security, and streamlined workflows for your web applications.

This guide shows you how to approach this kind of migration and arms you with questions, checklists, and common pitfalls to be prepared for. We'll cover the barriers to migration, benefits of the Jamstack model, how to select your tools, and plan your migration.

Table of Contents

Jamstack Explained

What is Jamstack?

A successful migration needs to ensure that any new site architecture meets your requirements. Before we dive into the necessary steps to undergo a migration to Jamstack, it's important to understand core Jamstack principles, including mechanics of the architecture, the workflows it enables, and the methodology behind it.

Jamstack is an ethos for how to build for the web. With Jamstack, the UI is compiled, the front-end is decoupled, and data is pulled in from a vast API ecosystem. By decoupling the front-end from the back-end infrastructure and pre-rendering as much content as possible, Jamstack apps are more efficient to build, and highly performant when it comes to Core Web Vitals and user experience.

What kinds of applications does Jamstack support?

Below are examples of organizations that are using Netlify and the Jamstack to support:

- Dynamic SaaS applications at scale: Twilio Console
- Highly performant and e-commerce stores that drive more conversions: Manscaped and Victoria Beckham Beauty
- Large-scale site builders: Spring, powering more than 6 storefronts
- Global, highly trafficked marketing sites: Gemini with more than 1 billion monthly web requests
- Technical docs as code: Apollo GraphQL

Why choose Jamstack?

Better performance:
The significant work done by front-end developers to deliver exceptional performance can be undermined when it is either inefficiently rendered on servers at request time, or compromised by a tightly-coupled, opinionated back-end. Instead of waiting for pages to build on the fly, Jamstack sites generate at deploy time. To minimize the time to first byte, nothing beats pre-built files served over a Content Delivery Network (CDN).

Higher security:
With server-side processes abstracted into microservice APIs, Jamstack reduces surface areas for attacks. And you can leverage the domain expertise of specialist third-party services.

Lower cost, simpler 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 an excellent solution for scaling and providing a better web experience.

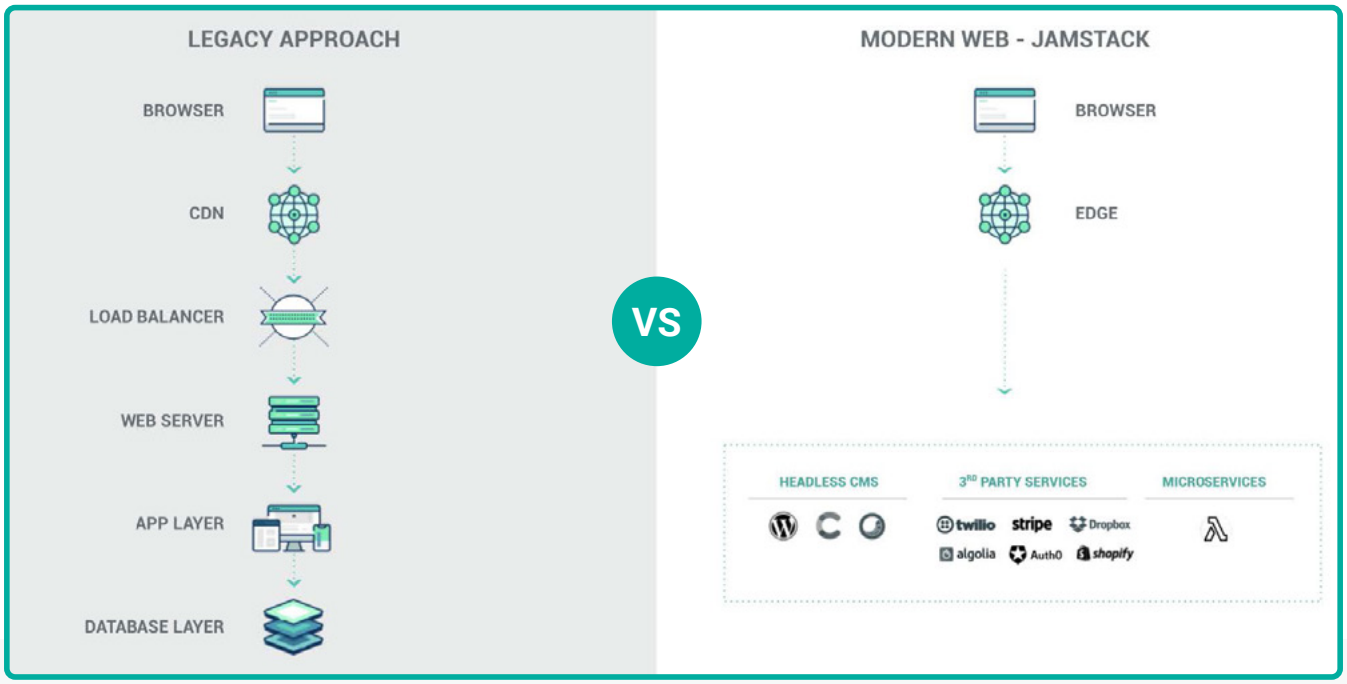
Better developer experience:
Loose coupling and separation of concerns allow for more targeted development and debugging.

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 and preserves important details.

[Jamstack Explained]

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 that enhance experiences with custom logic and secure calls to APIs.



Terms to Know

Monolith: A fully interconnected system, where every piece is part of a single whole. Updating one feature means rebuilding and redeploying the entire system.

Microservices: Individual pieces of functionality that are independently-built, operated, deployed, and scaled. They connect to other services by calling each other.

Decoupling: The process of creating a clean separation between systems or services. By decoupling the services needed to operate a site, it becomes easy to swap services in and out as you see fit and to use specialized third-parties instead of homegrown applications.

[Jamstack Explained]

Changes in web development practices have paved the way for a more efficient, modern web. These changes include:

Popularization of Git: This is a distributed version-control system for tracking changes in source code during software development, which has ushered in improvements in collaborative developing.

Continuous integration/continuous development (CI/CD): The universe of software build tools arrived for automation and CI/CD, improving the integration, testing, delivery, and deployment of an application throughout its lifecycle.

Static site generators (SSGs): SSGs were initially used for efficiency – a script that takes in data, content, and templates, processes them, and outputs a folder full of 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 front-end developers, who can now 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, serves it on a CDN, and, when set up the correct way, can bring the benefits of speed and security for web applications.

Click to explore Jamstack.

Netlify and the Jamstack

There are many moving parts to consider when building a Jamstack application. These include the edge network and cache invalidation, reliable infrastructure (required to build the site), and coordinating domains and DNS records for go-live.

The process of manually stitching together elements, such as deployment pipelines, build tools, edge networks, object stores, and API runtimes was an inherent barrier that needed to be removed to make the Jamstack a truly viable approach. That's where Netlify comes in.

Netlify's Approach to Jamstack

Netlify provides a first-class developer experience for teams building Jamstack applications, along with global, enterprise-grade network, serverless functions, and high-performance build environments to optimize your development workflows and performance.

Step One:
Audit Your Web Applications

Before you begin modifying any part of the codebase, the first 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 tool stack your team is using now and what functionalities are needed for the new site.

Audit your web applications according to these three categories:

+ **Architecture and Web Tooling**
+ **Site Stakeholders**
+ **Pain Points**

Scope the solution to your needs.

While many platforms may lay claim to a large list of features, few of these features are actively used. Migrating to a microservice gives you the opportunity to assess and cater to the essential features needed by your sites and applications.

Architecture and Web Tooling

Monolithic applications package a lot of functionality together, which can obscure your understanding of 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 front-end from the back-end. This allows for more modular development, and for teams to take advantage of the extensive ecosystem of third-party tools for better website functionality. It also means that teams migrating off a monolith must proactively select tools to replace that out-of-the-box functionality.

As an example, if you have an ecommerce app built on a monolith, you'll need to account for the fact that individual services, such as payments, checkout flow, storefront, and fulfillment will need to be broken into standalone services while your front-end is decoupled and built statically. While this flexibility is a key reason for migration to the Jamstack, it does require more upfront planning.

When assessing your architecture and web tooling, answer these questions to help you determine which parts to keep and which ones to change:

- **Is the entire app a monolith or is it decoupled?**
- **What languages does your architecture demand of your team?**
- **Is the code version controlled?**
- **Where is the code executed?**
- **Is a CMS like WordPress or Drupal used to build the site?**
- **Where is your site 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?**

A Jamstack migration doesn't always require swapping out tools. For example, if you use Sitecore as your CMS, you can find ways to "Jamstack-ify" it so that it can still be used for content editing, yet decoupled and served as a pre-generated site.

Customer Example:

How a European shopping mall leader embraced lightning fast headless Sitecore on Netlify with a predictable cloud spend

KLEPIERRE

Faced with the challenges of the digitally transformative shifts to the shopping experience, European mall operator Klépierre opted to massively overhaul their web presence. Their existing Sitecore architecture worked well for content authoring, but Sitecore's heavy infrastructure didn't allow for the high-performance experience expected by customers.

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

After encountering performance and price barriers with their planned Sitecore redesign, Proximity shifted their technical approach. With only three months left to complete 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, they massively reduced cloud spend while still delivering performant sites on the client's timeline.

While the initial reason to move to the Jamstack was cost-savings, the Proximity team soon realized the additional benefits of improved workflows, simpler cloud management, and better website performance with Netlify and the decoupled Sitecore architecture.

Click here to see the full case study.

[Step One: Audit]

Site Stakeholders

The next part of your web application audit is to identify the site stakeholders. Web development typically involves many separate groups which are required to build, design, maintain, and scale the site, as well as create content. Different teams have different concerns. Ask these questions to understand how different teams work together and what their individual workflows are, so that you can meet their requirements when using the Jamstack:

- **Who builds the site?**
- **Who deploys the site?**
- **Who maintains the site?**
- **Are multiple engineering teams involved to oversee different components, such as the front-end and back-end? If so, how do they work together?**
- **Who is the business owner of the site?**
- **What does each team care about?**

Pain Points

After assessing the web tools and stakeholders, complete your web applications audit by conducting an analysis to discover what is working well, and what might be causing some pain points. Use these questions as a guide:

Site speed: How performant is your site?

Is the site fast in some regions, but slow in others?

Is this because you serve all your customers from a server in a single region?

Are the front-end engineers impaired by the back-end technology, or can they exercise all their skills without compromise?

Time to market: How long does it take to ship new features or changes to the site?

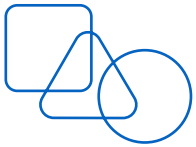
Can the developers ship features quickly enough with their dependency on multiple teams?

Your front-end teams build out the app or pages, but depend on DevOps to spin up the environments for testing, staging, and QA. Does this lead to a delay in shipping updates to the site?

Development workflow deficiencies: What needs to change?

If multiple teams work together to build a site, what does the handoff look like between these teams? Is it ideal? Why were these processes set up like this?

Who is the point of contact if the site breaks? If there is no DevOps team, how much time does the front-end team spend on cache invalidation, administering servers, and scaling the CDN?



Migrating to a new stack is an opportunity to address these pain points, but only once you have a good understanding of the problems that you are trying to solve for.

Step Two:
Determine the Technology and People

Now that you have more insight into the reasons why you want to migrate, the next step is to determine which web property to begin with, and who will build the site and the infrastructure to host it.

[Step Two: Technology & People]

Which property will you move first?

You can categorize the different properties accordingly:

Content-heavy sites

These include campaigns, landing pages, and .com sites that are not primarily delivering deep interactions and/or transactional functionality. Content-focused sites provide a low-risk project to prove the value of the Jamstack to other stakeholders.

Web applications

These include anything that requires functionality, such as login, a database, or user management, and has dynamic functionality at runtime.

The key to migrating an app with functionality is to decouple the front-end from the back-end, which requires breaking different services into their own standalone applications. You'll need to use APIs – either your own or third-party services, like Algolia for search or Auth0 for authentication and user management. Auditing the third-party APIs you use will be helpful in this process.

Large site with high volumes of pages

These include publications and e-commerce sites with over 100,000 products and pages.

Migrating a large site at once might seem overwhelming. So instead, begin by breaking the site down into sections. These could be logical sections based on ownership, such as docs, community, and products, or functional sections based on how often they are updated. For more information about this type of migration, visit our blog post, How to build large sites on Netlify.

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

[Step Two: Technology & People]

Who will build it?

The main collaborators in a Jamstack site might look different than in your previous web architecture initiatives. That's because the Jamstack empowers front-end developers. With the Jamstack, front-end code is no longer restricted to being a product of a back-end system.

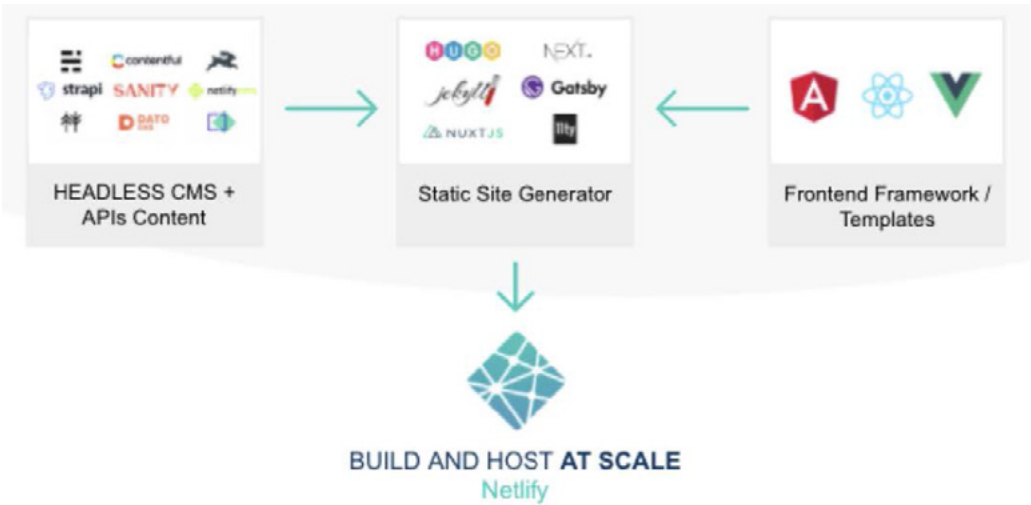
When using the Jamstack, your developers should have less exotic proprietary skills, and can draw from a larger, more portable, and more available general modern front-end developer community. This makes their skill sets transferrable and relevant in modern web development. Confirm that your development team can support the migration and is comfortable with JavaScript, APIs, and markup.

What is your tech stack?

The requirements to build a Jamstack are different than what's needed to build a monolithic app. Everything from code, to content building and hosting is 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, to pre-build the site into static assets**
4. **A platform/infrastructure, to help you build and host the site**

While these are broad areas, there are plentiful options when it comes to selecting what works for your needs. Next, some points to consider for your tech stack:



How to Choose a Static Site Generator (SSG)

You'll need to decide how you generate your site. Choosing a static site generator goes back to the kind of app and type of site you are building:

Content-heavy sites:
Find a tool which prioritizes page and URL generation, such as Jekyll, Hugo, and Nuxt.

Complex, app style functionality:
Gatsby and Next.js are good choices because they come with more advanced, client-side features that are ready to plug in and go.

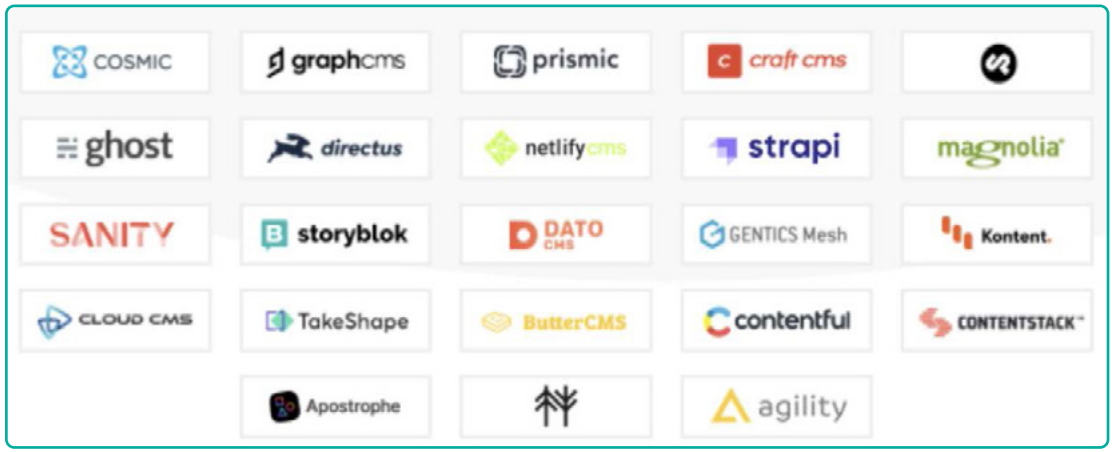
Large application:
If you have larger sites, then you will have longer builds. Hugo and 11ty have impressive generation times, and Netlify's On-Demand Builders functionality can also assist with reducing build time.

Another significant 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 SSG can be 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 or multiple SSGs on it. With Netlify, your teams have the flexibility to build in the language they are comfortable with, while also trying out the best framework at your convenience.

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

This is important, because as your teams grow, they will have different preferences. Some might lean towards Gatsby, while others choose Next.js, Nuxt, or a few different frameworks, for whatever they're building.



How to Choose a Static Site Generator (SSG) *continued*

Most teams already have a content management system (CMS) in place, whether it's a headless CMS or a monolith. While a migration is a suitable 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 WordPress, Drupal, and Sitecore, now have a headless version that can be used with Jamstack applications. 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.

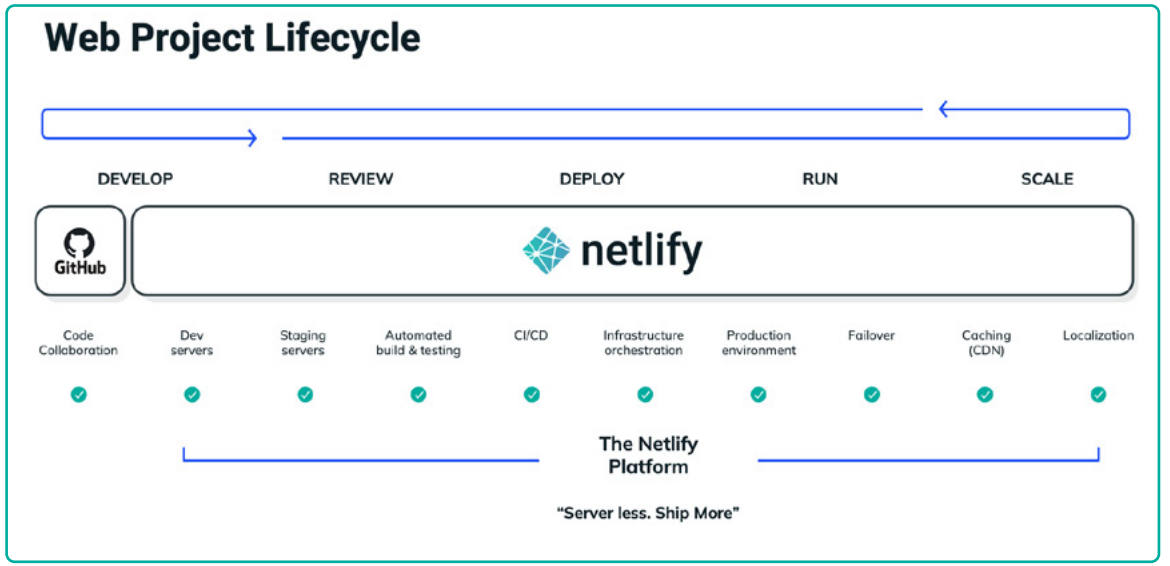
[Step Two: Technology & People]

Choose where you will build and host

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

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

Netlify empowers front-end developers to build sites by pushing code to Git, while managing the rest of the review and deploying workflows for them. The Netlify platform manages the build and hosting of the infrastructure, and scaling and maintenance for your teams, so you can focus on building your site.



Step Three:
Perform the Migration Incrementally

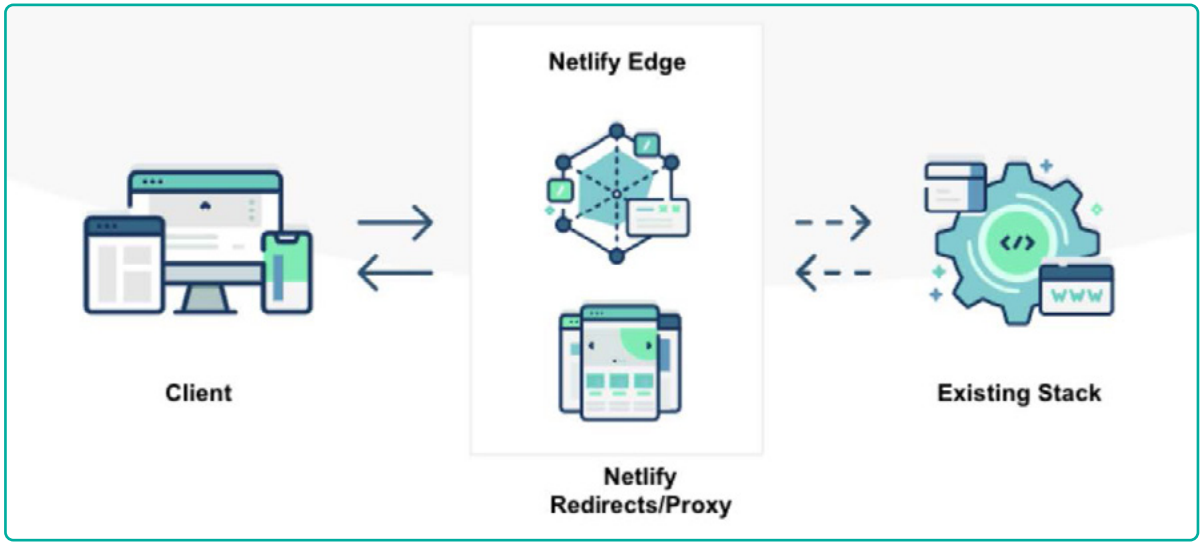
You don't have to migrate your entire site in a single day and do a "big bang" migration. Based on your timelines, Netlify helps customers adopt the Jamstack gradually, even page by page. About 90% of Netlify's customers have migrated to the Jamstack incrementally. This combines a low-risk migration with the benefits of speed for the Jamstack sections of your site.

[Step Three: Migration]

Understanding Incremental Migrations

Incremental migration to the Jamstack on Netlify is possible because of Netlify redirects and the proxy API.

Netlify provides access to highly optimized edge-based routing. This means that you can direct all your site traffic to Netlify at the DNS level, and Netlify invisibly and rapidly routes every request to your existing platform, unless it can satisfy the request from resources it hosts as a Jamstack site. With this approach, 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.



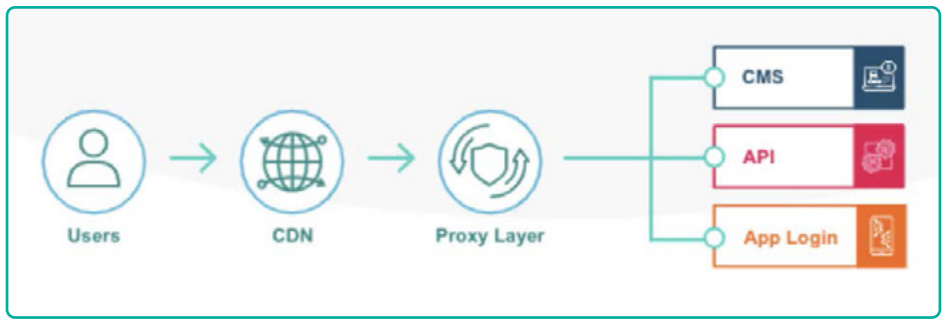
[Step Three: Migration]

Phases of Migration

Below is an illustration of what the typical phases of a migration would be.

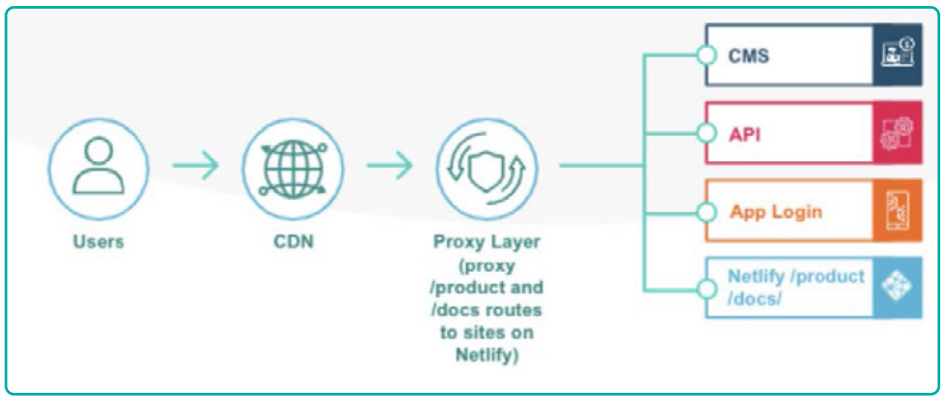
Phase 0: Legacy Stack

During this phase, your site is still on a legacy stack and includes a CDN and proxy layer that routes user traffic to the website, application, or other services.



Phase 1: Migrate Pages

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

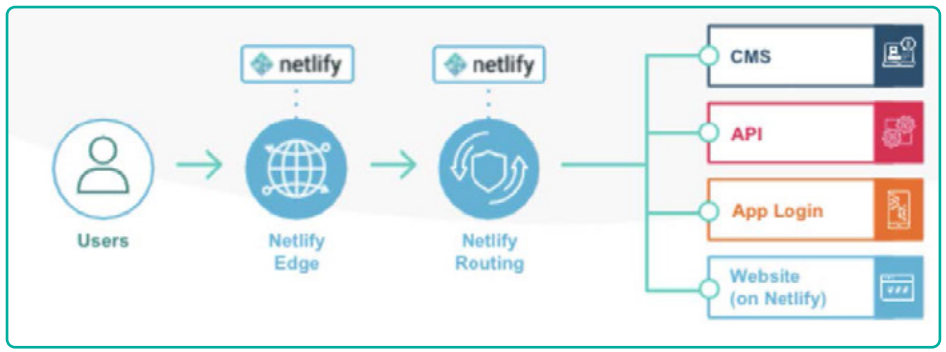


[Step Three: Migration]

Phase 2: Migrate Logic

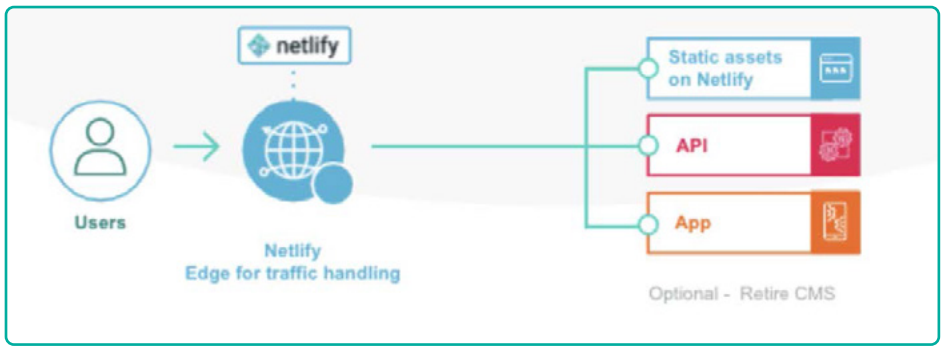
As more of the site is built on Netlify, it is highly recommended you add all your business logic to the Netlify Edge, including redirects and proxies, to give your end users a uniform experience on your domain.

Phase 2 involves building and hosting your web properties on Netlify. Parts of the site built on Netlify will be served by the Netlify platform, and any parts which the platform can't serve will be proxied to your legacy infrastructure from the Netlify Edge. This removes the need for your existing CDN, since the Netlify Edge will assume this function.



Phase 3: Retire the Monolith

After completing the Jamstack migration of your site and decoupling the rest of your back-end services, you can retire your legacy monolith completely. For information about best practices on live DNS site migration, visit this checklist.



Next Steps

Technology choices have impacts. The methods to architect websites and the tools to build them affect site performance, functionality, user experiences, and the workflows that your teams use to build and edit the site.

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

That's one of the reasons why people are choosing Jamstack. It's an iterative, flexible, approach that allows teams to select the tools they want, while unlocking productive workflows for front-end teams.

Netlify experts can help you migrate away from an older, monolithic stack by scoping your project, providing you with the questions that you need to ask, and advising you on common pitfalls at every stage of the process, ensuring that you're using Jamstack best practices from the beginning.

Contact a Netlify expert to get started.