Technical Analysis of Infrastructure Migration Pathologies: Transitioning from Vercel to Netlify Deployment Paradigms

The contemporary cloud-native ecosystem is defined by a rigorous competition between specialized hosting platforms that abstract away the complexities of traditional server management. Among these, Vercel and Netlify have emerged as the primary contenders for the deployment of modern web applications. While both platforms provide a seemingly similar developer experience characterized by Git-based workflows and automatic provisioning, the underlying technical architectures are significantly different. When an engineering team initiates a migration from Vercel to Netlify, the emergence of a "black screen" or a blank page is often the first symptom of a deeper systemic failure in the translation of build-time artifacts to runtime environments. This report provides an exhaustive technical analysis of the mechanisms governing these transitions, the etymology of deployment failures, and the specific remediation strategies required to ensure infrastructure parity.

The Evolution of Hybrid Compute Paradigms

The shift from monolithic web servers to decoupled, globally distributed infrastructure has necessitated a new class of "Framework-Aware" hosting. Vercel's core value proposition is built upon its status as the creator and primary maintainer of Next.js, leading to an infrastructure that is deeply integrated with the framework's internal logic.[1] Its architecture utilizes "Framework Defined Infrastructure" (FDI), where the build process automatically provisions the necessary serverless and edge resources based on the detected patterns in the code.[1] In contrast, Netlify offers an open-platform philosophy that prioritizes a broad plugin ecosystem and built-in features like native form handling, identity services, and background functions that extend beyond the capabilities of a single framework.[2]

These philosophies manifest in the way the platforms handle compute resources. Vercel provides a highly optimized environment for Next.js features such as Incremental Static Regeneration (ISR) and Server Components, often offering higher memory limits for serverless functions to accommodate heavy server-side rendering (SSR) tasks.[1] Netlify focuses on the "Composable Web," providing a mature set of primitives that work across diverse frameworks like Gatsby, Nuxt, and Hugo, while maintaining a strong emphasis on developer experience through its robust Command Line Interface (CLI) that mirrors production environments locally.[3]

Infrastructure Attribute	Vercel (Next.js Optimized)	Netlify (Open Ecosystem)
Compute Memory	Up to 4GB	Fixed 1GB
Execution Timeout	10s (Hobby) to 800s (Pro)	60s (Standard)

Function Runtime	Node.js, Python, Go, Ruby	Node.js, Go
Edge Runtime	V8-based (Node-like)	Deno-based
Native Database	No (External Integrations)	Yes (Serverless Postgres/Neon)
Background Jobs	Limited	Native (up to 15 mins)
Image Optimization	Optimized for Next/Astro/Nuxt	Universal Image CDN

[1]

The Etiology of the "Black Screen" Symptom

The "black screen" error is a clinical manifestation of a failure in the application's hydration or initialization phase. Unlike a server-side crash that typically returns a 500-series status code, a black screen often occurs when the initial HTML document is successfully delivered, but the client-side JavaScript bundle fails to execute or locate its mount point.[6] In the context of a Vercel-to-Netlify migration, this phenomenon is usually triggered by one of three primary technical failures: asset resolution errors, MIME type mismatches, or runtime environment configuration disparities.

Asset Resolution and the "Catch-All" Trap

A frequent cause of the black screen is the misconfiguration of the publish directory. In a Single Page Application (SPA) environment, the build process generates a set of static assets (JS, CSS, and HTML) in a specific folder, such as dist for Vite or .next for Next.js.[7] If Netlify is configured to serve the root directory instead of the build output folder, it may find an old index.html but fail to find the corresponding JavaScript bundles.[8]

This is exacerbated by the use of SPA "catch-all" redirects. A common rule like /* /index.html 200 is designed to support client-side routing by serving the main HTML file for any path that does not match a static file.[12] However, if the path to a JavaScript file (e.g., /static/main.js) is incorrect due to a publish directory mismatch, the redirect engine will catch the request and return the content of index.html instead of the script.[13] The browser's attempt to execute HTML as JavaScript results in a silent failure or a console error, leaving the user with an uninitialized black screen.[13]

MIME Type Enforcement and Script Blocking

Modern browsers enforce strict MIME type checking for module scripts. If a server responds to a script request with a MIME type other than application/javascript (or similar valid types), the script is blocked.[13] During migration, many developers encounter an error stating: "Failed to load module script: Expected a JavaScript module script but the server responded with a MIME type of 'application/octet-stream'".[15] This typically occurs when the build process has not correctly mapped the file extensions or when the Netlify server cannot identify the file type because it is looking in the wrong directory.[16]

Furthermore, if the "catch-all" redirect returns the HTML file (MIME type text/html) in response

to a .js request, the browser will throw a "Disallowed MIME type" error.[13] This is a secondary effect of the asset resolution failure described above, where the redirect engine prioritizes the fallback rule over the actual file delivery because the file is not where the server expects it to be.[13]

Client-Side Rendering (CSR) and Animation Blocks

In some instances, the black screen is not a failure of script loading, but a failure of script execution logic. In frameworks that use animation libraries like Framer Motion or GSAP to handle initial page transitions, the root element may be set to opacity: 0 by default.[17] If the JavaScript execution is halted by a missing environment variable or a failed API call that was previously handled by a Vercel-specific proxy, the "fade-in" animation never triggers.[17] This leaves the application in a state of "invisible success," where the code is running, but the UI is hidden from the user.[17]

Framework-Specific Migration Pathologies

The technical requirements for a successful migration vary significantly depending on the underlying framework. Next.js, Vite, and Create React App (CRA) each have unique build outputs and directory requirements that must be reconciled with Netlify's deployment engine.

Next.js Deployment on Netlify Runtime v5

Next.js represents the most complex migration path due to its deep ties to Vercel infrastructure. To bridge this gap, Netlify utilizes the @netlify/plugin-nextjs, commonly referred to as the Next.js Runtime.[18] The transition from Runtime v4 to v5 has introduced performance improvements but also specific troubleshooting requirements for Next.js 14 and 15.[19]

One notable issue in Runtime v5 involves the rendering of the next/image component. Developers have reported instances where images fail to render, leaving black backgrounds or blank spaces where optimized assets should be.[20] This is often traced to misconfigurations in next.config.js, specifically within the remotePatterns section. An empty port string or excessive wildcards in the pathname can cause the Netlify Image CDN to fail the asset request.[21] Removing these specific empty declarations and narrowing the wildcard patterns often restores functionality.[21]

Next.js Feature	Netlify Implementation Mechanism	Potential Failure Mode
Static Pages	Served directly from Global CDN	404 if publish directory is not .next [22]
Dynamic SSR	Converted to Netlify Functions (Node)	500 if dependencies are missing or bundle > 250MB [23]
API Routes	Converted to Netlify Functions (Node)	Incorrect pathing in client-side calls [24]

Image Optimization	Netlify Image CDN	Black screen if remotePatterns are malformed [21]
Middleware	Netlify Edge Functions (Deno)	Runtime crash due to Node/Deno syntax mismatch [25]
ISR	On-demand Builders	Cache invalidation delays or failures [18]

[18]

Vite-Based React and Vue Applications

Applications built with Vite are increasingly common for SPAs that do not require the full overhead of Next.js. When migrating these to Netlify, the most critical factor is the publish directory setting. By default, Vite outputs to a dist folder at the project root.[9] If the project is structured as a monorepo or has the application in a subdirectory (e.g., /frontend), the publish directory must be set to frontend/dist.[8]

Failure to specify the correct base directory can lead to Netlify attempting to build from the repository root, missing the local package.json and failing to execute the vite build command entirely.[8] Even if the build succeeds, an incorrect publish path will result in the "Page Not Found" screen or a blank page because the server is serving an empty directory.[7] Developers are encouraged to run npm run build and npm run preview locally to verify the output folder before committing the configuration to Netlify.[8]

Refactoring Edge Functions: From Node.js to Deno

A significant architectural hurdle in the Vercel-to-Netlify migration is the fundamental difference in the Edge computing runtimes. Vercel Edge Functions run on a lightweight execution environment built on the V8 engine, which maintains high compatibility with Node.js APIs.[4] Netlify Edge Functions, however, are built on the Deno runtime.[25] This shift necessitates a complete refactoring of any middleware or edge-level logic.

Environment Variable Access Syntax

One of the most immediate points of failure in an edge migration is the method of accessing environment variables. Vercel allows the use of the standard Node.js process.env global.[25] In the Deno environment used by Netlify, process is not defined. Attempting to access process.env will throw a reference error, causing the edge function to crash and the request to fail.[25] Netlify requires the use of the Netlify.env.get() method.[25]

Module Resolution and Built-in Imports

Deno handles imports differently than Node.js. It does not look for modules in a node_modules folder by default and requires explicit file extensions or URL-based imports.[25] To use Node.js built-in modules (like crypto or buffer) in a Netlify Edge Function, developers must use the node: prefix (e.g., import { Buffer } from "node:buffer").[25] Furthermore, standard Deno

modules must be imported via their full URLs from providers like deno.land.[25]

Geolocation and Contextual Metadata

The API for accessing request metadata such as geolocation also differs between the platforms. In Vercel, geographic data is typically found on the request.geo object.[25] On Netlify, this data is accessed via a context object provided as the second argument to the edge function handler.[25]

Metadata Feature	Vercel (Node/V8)	Netlify (Deno)
Environment Access	process.env.VAR	Netlify.env.get("VAR")
Built-in Modules	import { x } from "module"	import { x } from "node:module"
Geolocation	request.geo.city	context.geo.city
File Extensions	Not required (usually)	Required (e.g., .ts, .mts)
Package Manager	node_modules	URL-based or Import Maps

[25]

Configuration Management via netlify.toml

The netlify.toml file is the primary mechanism for declaring the site's infrastructure-as-code on the Netlify platform. While Vercel often relies on the vercel.json or automatic dashboard settings, Netlify's file-based configuration takes precedence over UI settings and is essential for complex migrations.[26]

Standard Build and Publish Directives

The [build] section defines the core lifecycle of the deployment. For a migrated Next.js site, the configuration must explicitly target the .next directory.[22] For static sites or other frameworks, the command and publish keys are the most frequent sources of error. If the build command is npm run build but the local configuration of the framework has been modified to output to a folder other than dist, the publish key must reflect that change to avoid a blank deployment.[7]

Redirects and Rewrites Syntax

Redirects in netlify.toml are defined using an array of tables. This format is more structured than the _redirects file and allows for advanced features like signed proxy redirects and role-based access control.[26] A critical point for migration is the force attribute. By default, Netlify will not trigger a redirect if a file exists at the requested path.[26] Appending a ! to the status code in _redirects or setting force = true in netlify.toml overrides this shadowing behavior.[26]

Ini, TOML

```
[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200
  force = false # Set to true to override existing files
```

Functions and Edge Functions Scoping

The [functions] and [build.edge_functions] sections allow for the definition of directory locations and bundling behavior. When migrating, it is often necessary to explicitly point Netlify to the refactored function directories.[24] Additionally, the included_files directive is vital for SSR pages that require access to local data files (like .json or .md files in a /content folder) that are not automatically picked up by the function bundler.[23] Failure to include these files will result in runtime errors where the function cannot find its data source, leading to 500-series errors and an empty UI.[23]

Diagnostic Protocols for Migration Failures

When an application displays a black screen after migration, a structured diagnostic protocol is necessary to isolate the failure point within the networking, build, or runtime layers.

Browser-Level Debugging

The first step in any diagnosis is the inspection of the browser's developer tools. The network tab provides immediate feedback on whether assets are failing to load (404) or are being served with incorrect MIME types.[13] A 404 on the main entry point script (e.g., index.js) almost always indicates a publish directory mismatch.[8] If the script loads but the console shows a ReferenceError, the issue likely resides in an environment variable that was not correctly ported or a global object that is unavailable in the Netlify environment.[17]

Log Auditing and Request Tracing

Netlify provides separate logs for the build process and the runtime execution of functions. A "Successful" build log does not guarantee a working site. Developers must check the "Functions" tab in the Netlify dashboard to see real-time execution logs.[17] If a serverless function is crashing, the stack trace will reveal whether it is due to a missing module, a timeout, or a memory limit violation.[17] Netlify's serverless functions have a fixed memory limit of 1GB and a standard timeout of 60 seconds, which is more restrictive than Vercel's Pro tier.[1]

Local Emulation via Netlify Dev

One of Netlify's most powerful diagnostic tools is the netlify dev command.[3] This tool starts a local server that emulates Netlify's production environment, including the redirect engine,

serverless functions, and environment variable injection.[3] If the black screen persists in the local emulation but not during the standard npm run start, the issue is definitively rooted in the Netlify-specific configuration (e.g., a problematic redirect rule or a function configuration error).[3]

External Interference: VPNs and Security Software

An often-overlooked cause of rendering issues is the interference of Virtual Private Networks (VPNs). Specifically, users of NordVPN have reported instances where the "isWhitelisted" flag in local storage, combined with VPN-specific security policies, can block the execution of deployment previews or certain platform-specific scripts.[10] Killing the VPN process (rather than simply disabling it) has been documented as a fix for these specific rendering blocks.[10]

Production Readiness and Performance Optimization

Ensuring that a migrated site "works and functions" extends beyond fixing the black screen; it requires achieving performance and reliability parity with the previous Vercel deployment.

Image Optimization and Content Delivery

Vercel provides automatic image optimization that is deeply integrated with the Next.js next/image component, including automatic conversion to WebP or AVIF formats.[1] Netlify offers a similar Image CDN that works across all frameworks.[5] For a successful migration, developers must verify that the next/image components are correctly interacting with Netlify's CDN.[22] This often requires ensuring that the @netlify/plugin-nextjs is up to date, as older versions of the plugin may not support the latest Next.js optimization features.[19]

Caching and Incremental Regeneration

Netlify handles Incremental Static Regeneration (ISR) through its "On-demand Builders".[18] This mechanism allows for the dynamic generation of pages that are then cached at the edge for subsequent requests.[18] To optimize performance, developers should configure their serverless functions in the region closest to their primary data sources (e.g., the database) to minimize latency during the initial generation phase.[22]

DNS Cutover and Domain Management

The final stage of migration is the DNS transition. Netlify recommends using a www or another subdomain as the primary domain for optimal use of their global CDN.[22] For a smooth cutover, developers should plan for DNS delays by lowering the Time to Live (TTL) values on their existing DNS records prior to the switch.[18] Tools like dig or whatsmydns.net can be used to track the propagation of nameserver changes in real-time.[18]

Launch Phase	Critical Action	Objective
Pre-Migration	Document Vercel build settings	Configuration baseline [22]

	& env vars	
Code Conversion	Refactor Edge Middleware to Deno/mts	Runtime compatibility [25]
Build Test	Verify local build with netlify dev	Eliminate config errors [3]
Deployment	Clear Netlify cache and redeploy	Fresh asset generation [19]
Post-Launch	Audit network tab for 404s/500s	Identify hidden regressions [17]
DNS Cutover	Point nameservers to Netlify	Traffic migration [18]

[3]

Advanced Troubleshooting: The Legacy Next.js Runtime

For projects running older versions of Next.js (v10 to v13.4), Netlify provides the "Legacy Runtime" (v4).[23] Troubleshooting this runtime requires attention to the "Large Function Error." If a generated function exceeds the 250MB size limit, the deployment may fail or behave unpredictably.[23] This is often caused by large native dependencies like chromium or a massive number of pre-rendered pages being bundled into the function.[23] To mitigate this, developers can use the fallback: "blocking" strategy in getStaticPaths to defer page building until the first request, thereby reducing the initial bundle size.[23]

In summary, the transition of a web application from Vercel to Netlify is not merely a change of hosts but a significant migration of runtime logic and build-time configurations. The "black screen" is a high-level symptom of several potential failures, primarily rooted in directory mapping, MIME type enforcement, and the Deno-Node runtime divide. By systematically auditing the publish directory, refactoring edge-level code for Deno compatibility, and leveraging Netlify's local emulation tools, engineering teams can successfully navigate this transition while maintaining the performance and reliability expected of modern web infrastructure.

Works cited

1. Vercel vs Netlify | Vercel Knowledge Base, accessed January 30, 2026, https://vercel.com/kb/guide/vercel-vs-netlify
2. Vercel vs Netlify: Which One Should You Choose? - Codecademy, accessed January 30, 2026, https://www.codecademy.com/article/vercel-vs-netlify-which-one-should-you-choose
3. Netlify vs Vercel: 2025 Comparison, accessed January 30, 2026, https://www.netlify.com/guides/netlify-vs-vercel/
4. Vercel vs Netlify 2025: The Truth About Edge Computing Performance - DEV Community, accessed January 30, 2026,

https://dev.to/dataformathub/vercel-vs-netlify-2025-the-truth-about-edge-computing-performance-2oa0
5. Vercel vs Netlify vs Cloudflare Pages: 2025 Comparison - Digital Marketing Agency, accessed January 30, 2026, https://www.digitalapplied.com/blog/vercel-vs-netlify-vs-cloudflare-pages-comparison
6. Deployed Site on Netlify Loads as Black Screen – How to Fix? - Support, accessed January 30, 2026, https://answers.netlify.com/t/deployed-site-on-netlify-loads-as-black-screen-how-to-fix/141120
7. [SOLVED] weird errors on netlify build (vite-react) - Threads - Appwrite, accessed January 30, 2026, https://appwrite.io/threads/1176563965589983232
8. Fixing a Blank Page in a React To-Do App on Netlify | by Yetnayet ..., accessed January 30, 2026, https://medium.com/@lakewyetnayet93/fixing-a-blank-page-in-a-react-to-do-app-on-netlify-68cfee85ec19
9. Deploying a Static Site - Vite, accessed January 30, 2026, https://vite.dev/guide/static-deploy
10. Blank screen on Vercel deployment · vercel next.js · Discussion #59944 - GitHub, accessed January 30, 2026, https://github.com/vercel/next.js/discussions/59944
11. Conflict between /public/_redirects and netlify.toml - Support, accessed January 30, 2026, https://answers.netlify.com/t/conflict-between-public-redirects-and-netlify-toml/137758
12. How to solve Vercel and Netlify "Page Not Found" after page refresh [SOLVED], accessed January 30, 2026, https://dev.to/devvsakib/how-to-solve-vercel-and-netlify-page-not-found-after-page-refresh-solved-2ol7
13. Failed to load module script: Expected a JavaScript module script but the server responded with a MIME type of "text/html". Strict MIME type checking is enforced for module scripts per HTML spec - Support, accessed January 30, 2026, https://answers.netlify.com/t/failed-to-load-module-script-expected-a-javascript-module-script-but-the-server-responded-with-a-mime-type-of-text-html-strict-mime-type-checking-is-enforced-for-module-scripts-per-html-spec/122743
14. Site is deployed but showing white screen and no errors - Netlify Support Forums, accessed January 30, 2026, https://answers.netlify.com/t/site-is-deployed-but-showing-white-screen-and-no-errors/87439
15. Loading module from my site was blocked because of a disallowed MIME type (“text/html”), accessed January 30, 2026, https://answers.netlify.com/t/loading-module-from-my-site-was-blocked-because-of-a-disallowed-mime-type-text-html/110101
16. Blank screen on netlify after a react or a vite deploy - Stack Overflow, accessed January 30, 2026, https://stackoverflow.com/questions/78472255/blank-screen-on-netlify-after-a-r

eact-or-a-vite-deploy
17. Blank screen after deployment - v0 - Vercel Community, accessed January 30, 2026, https://community.vercel.com/t/blank-screen-after-deployment/9530
18. Migrating from Vercel to Netlify - Complete Next.js Migration Guide - Tech For Palestine, accessed January 30, 2026, https://techforpalestine.org/guides/vercel-to-netlify
19. Cannot use nextjs runtime v5 - Netlify Support Forums, accessed January 30, 2026, https://answers.netlify.com/t/cannot-use-nextjs-runtime-v5/155519
20. Black Background Screen Issue with Next.js Image Components on Netlify - Support, accessed January 30, 2026, https://answers.netlify.com/t/black-background-screen-issue-with-next-js-image-components-on-netlify/129283
21. Netlify Runtime v5 and Next.js Image Issue : r/webdev - Reddit, accessed January 30, 2026, https://www.reddit.com/r/webdev/comments/1bf092n/netlify_runtime_v5_and_nextjs_image_issue/
22. Vercel to Netlify migration checklist, accessed January 30, 2026, https://docs.netlify.com/resources/checklists/vercel-to-netlify-migration/
23. Troubleshooting Next.js on Netlify, accessed January 30, 2026, https://docs.netlify.com/build/frameworks/framework-setup-guides/nextjs/legacy-runtime/troubleshooting/
24. Migration guide - step-by-step instructions | 5.0.1 - Netlify Developers, accessed January 30, 2026, https://developers.netlify.com/sdk/get-started/migration-guide/migration-steps/
25. Migrate from Vercel to Netlify with an AI agent | Netlify Docs, accessed January 30, 2026, https://docs.netlify.com/resources/migrate/vercel-ai-agent-migration/
26. File-based configuration | Netlify Docs, accessed January 30, 2026, https://docs.netlify.com/build/configure-builds/file-based-configuration/
27. Redirects and rewrites | Netlify Docs, accessed January 30, 2026, https://docs.netlify.com/manage/routing/redirects/overview/
28. Redirect options | Netlify Docs, accessed January 30, 2026, https://docs.netlify.com/manage/routing/redirects/redirect-options/
29. 404 error not found during deployment - Help - Vercel Community, accessed January 30, 2026, https://community.vercel.com/t/404-error-not-found-during-deployment/12371
30. Why is my deployed project showing a 404 error? | Vercel ..., accessed January 30, 2026, https://vercel.com/kb/guide/why-is-my-deployed-project-giving-404
31. Production launch checklist | Netlify Docs, accessed January 30, 2026, https://docs.netlify.com/resources/checklists/production-checklist/