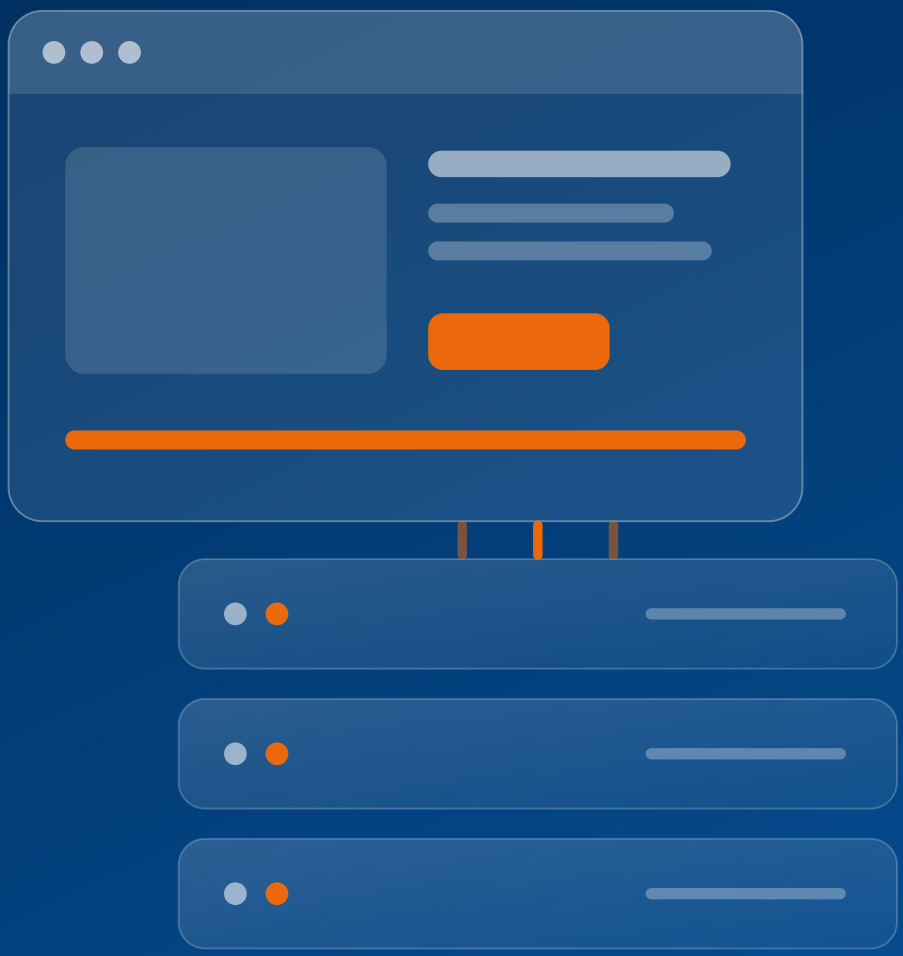
Hypernode | Hyvä



PLAYBOOK

The Hyvä Migration Playbook

A practical guide for planning your migration from Luma to Hyvä

Contents

– Preface 3

01 The Purpose of This Playbook 4

What You'll Find Inside 4

Who This Playbook Is For 4

02 A Quick Business Case For Hyvä 5

The 5 Signs Your Storefront Has Hit a Ceiling 5

What Level of Performance Can You Expect From Hyvä? 6

The Business Case: Mapping Your ROI 7

03 Hyvä Migration: Defining the Scope of Work 8

The Migration Complexity Matrix 9

How Much Does a Hyvä Migration Cost? 9

Six Technical Bottlenecks That Stall Launches 10

Hosting & Infrastructure Requirements 11

04 Hyvä Migration Roadmap 12

05 Protecting the Project 15

The Migration Risk Matrix 15

– The Bottom Line on Your Hyvä Move 17

Your Immediate Next Steps 17

PREFACE

Preface

In late 2025, Hyvä and Hypernode entered a strategic partnership. The shared goal is clear: make Magento faster, simpler, and easier to scale.

For years, Magento merchants have faced the same challenge. Yes, the platform is powerful and highly flexible. However, frontend complexity, performance bottlenecks, and operational overhead often make that flexibility expensive to maintain. Too many stores find themselves choosing between Magento's customisation and the simplicity of SaaS platforms.

Since partnering, Hyvä and Hypernode have been working closely to bring SaaS-like simplicity to Magento.

For years, building and running a fast Magento store felt like an uphill battle against the platform's own architecture.

FRONTEND

Hyvä changed the game on the frontend by throwing out Magento's notoriously bloated, legacy Luma setup. In its place is a lean, modern architecture that strips away the complexity and lets pages load instantly.

INFRASTRUCTURE

But a lightning-fast frontend is only as good as the engine driving it. That's where Hypernode comes in. Hypernode handles the heavy lifting under the hood, fine-tuning everything from server-level caching and deployment workflows to isolated staging environments, all built exclusively for Magento performance.

When you pair them together, merchants and agencies finally get what they've always wanted: a Magento store that is faster to develop, quicker to load, and simpler to maintain.

SECTION 01

The Purpose of This Playbook

01

The most common question we hear from the Magento community isn't “Is Hyvä worth it?” That much is clear. Countless real-world success stories and results have already answered that.

Instead, the question we hear most is: "How much work is involved in moving to Hyvä?"

This playbook aims to make Hyvä migrations predictable. We outline the technical work involved, typical timelines, project phases, and what the day-to-day effort looks like in practice.

What You’ll Find Inside

This playbook is a practical deep dive into what a Luma-to-Hyvä migration actually entails. We’ll break down:

Technical Realities

Understanding the core architecture changes.

Project Phases

Mapping the journey from initial scoping to launch day.

Infrastructure Setups

Configuring your servers to get the most out of your code.

Hurdles & Blind Spots

Identifying common pitfalls to watch out for along the way.

Whether you are trying to convince a client to switch or mapping out your own migration timeline, this guide serves as your step-by-step manual.

Who This Playbook Is For

A successful migration requires tight alignment between the people managing the business and the team writing the code. We’ve designed this playbook to give both sides exactly what they need:

Merchants & E-commerce Directors

The decision-makers who need to understand the bottom-line business impact, justify the investment, map out realistic budgets, and know what to expect regarding timelines before signing off.

Agencies & Technical Leads

The engineers and project managers responsible for scoping the actual development hours, hunting down third-party extension conflicts before they break the build, and managing client expectations from day one.

SECTION 02

A Quick Business Case For Hyvä

02

Why the Magento Community is Moving to Hyvä

For years, Magento's default Luma theme was the standard choice for building storefronts. But being honest: it was and is infamously bloated. Passing Google's Core Web Vitals with a Luma site requires time-consuming, complex code gymnastics. It feels like fighting against the platform rather than working with it.

That's why merchants and agencies are switching from Luma to Hyvä.

Hyvä has fast become the new gold standard for Adobe Commerce and Magento Open Source. By replacing thousands of lines of outdated JavaScript libraries and clunky, decade-old code dependencies, Hyvä fixes Magento's biggest flaw: its lack of speed.

For development teams, the shift changes everything:

Faster build times

Developers spend their time building features rather than wrestling with Magento's complex layout hierarchy.

Easier maintenance

Less code means fewer bugs, lower technical debt, and smoother upgrades down the road.

A frontend that actually cooperates

Instead of fighting archaic tech stacks, developers get a clean slate that behaves exactly how it's supposed to.

The 5 Signs Your Storefront Has Hit a Ceiling

Knowing why Hyvä is popular doesn't mean you should jump immediately. A frontend rebuild is a serious investment of both time and money; it shouldn't be done just to chase shiny new tech.

To justify a migration, you first need to diagnose whether your current storefront has genuinely hit a ceiling. Here are five clear signs that your legacy Luma frontend is actively bottlenecking your business growth:

1. **Failing Google's Speed Tests:** Your Core Web Vitals and Lighthouse scores are constantly stuck in the red, and no amount of optimisation patches seems to fix them.

2. **Terrible Mobile Conversion Rates:** Mobile browsers drag under the weight of Luma's heavy script overhead, leading to choppy scrolling, delayed clicks, and abandoned carts.

3. **Sky-High Developer Bills:** Even minor frontend tweaks or basic bug fixes take your developers hours to untangle and test because the theme architecture is so convoluted.

4. **High Bounce Rates on Product Pages:** Your heavy category grids or media-packed product pages take more than a couple of seconds to render, causing users to leave before your product images even load.

5. **"Upgrade Paralysis":** You are actively putting off critical security patches because you're terrified that a core update will completely break your fragile, highly customised Luma theme.

What Level of Performance Can You Expect From Hyvä?

The financial returns of a frontend rebuild are driven entirely by one factor: sustained, uncompromised speed. Before mapping out the exact ROI, it's useful to understand how Hyvä achieves this performance baseline and how that technical shift alters the day-to-day reality of running your store.

Hyvä Performance Before Customisation

LUMA

With Luma, achieving strong Core Web Vitals is an uphill battle from day one. Heavy CSS files and massive JavaScript bundles mean your developers are forced into a cycle of continuous, complex optimisation just to keep the site afloat.

HYVÄ

With Hyvä, the frontend starts from a drastically lighter architecture. This provides an incredibly strong performance baseline right out of the box. Because you aren't inheriting decades of technical debt, many implementations achieve green Lighthouse and Core Web Vitals scores with minimal initial optimisation.

Essentially, your team starts with a clean, highly optimised foundation. Their focus immediately shifts from "fixing performance failures" to simply "maintaining performance" as new features are introduced.

How That Speed Manifests in the Real World

Chasing perfect scores on a test server is an engineering metric. The actual business value of this architecture shows up when the site is hit with real traffic, heavy code customisations, and daily catalogue updates:

- **Pages pop.** Customers stop hitting an invisible wall when they click through your catalogue. When a heavy category grid or product gallery loads instantly, mobile shoppers stay engaged instead of bouncing out of frustration.
- **You stop fighting Google's algorithm.** By fixing the technical foundation, you remove the hidden speed penalties that keep your pages buried on page two or three of the search results.

- **The frontend finally throws out KnockoutJS and RequireJS.** This means a customer's phone isn't choking on massive, unoptimized JavaScript files anymore. It's a massive relief for anyone trying to buy something on a spotty 4G connection.
- **Developers stop wrestling with the platform.** Hyvä drops Magento's old labyrinth of complex XML structures, archaic layout blocks, and messy LESS files. In their place are modern, predictable utilities like Tailwind CSS and Alpine.js.
- **The learning curve for your engineering team drops through the floor.** Because the codebase is clean and predictable, your developers can ship new UI features, launch marketing experiments, and run A/B tests in a matter of hours rather than days.

The Business Case: Mapping Your ROI

If your business is experiencing one or more of those bottlenecks, the conversation naturally turns to the numbers. Because fixing these core infrastructure issues requires a complete storefront rebuild, it demands a real upfront budget.

To measure the true return on investment, it's important to look past the initial price tag and map out your savings across two major financial shifts:

TOP LINE

Top-Line Revenue Growth (Traffic and Sales Lift)

Speed is a direct driver of revenue. When you drastically lower load times, you win on two critical fronts. First, Google rewards snappy performance with better organic search visibility. Second, actual shoppers (especially on mobile) no longer encounter the performance friction that causes them to abandon carts. More traffic combined with higher conversion rates yields an immediate impact on top-line revenue.

BOTTOM LINE

Bottom-Line Cost Reduction (Slashing Ongoing Dev Bills)

When your frontend isn't a tangled web of legacy code, your developers can work more efficiently. Features, tweaks, and marketing experiments that used to take days to build, test, and debug on Luma can often be shipped in a matter of hours. Over the course of a year, this spike in developer productivity drastically lowers your Total Cost of Ownership (TCO).

SECTION 03

Hyvä Migration: Defining the Scope of Work

The Extension Audit and Migration Effort

Hyvä itself isn't exclusively what drives the timeline or the budget. The real variable is the pile of third-party extensions already running on your current Luma store.

Because Hyvä tosses out Magento's default frontend tech, every single extension that touches the user interface has to be checked. You cannot skip this.

When mapping out the work, your dev team will sort every existing plugin into three distinct buckets:

BUCKET 1

Ready to go

These modules were built natively for Hyvä or fully support it out of the box. Zero extra effort needed.

BUCKET 2

Patched by the community

A massive perk of the Hyvä ecosystem is its community Git and Slack group. For most popular extensions (think major checkout, shipping, or marketing tools), someone has already written a compatibility wrapper. You just install the patch and tweak the styling.

BUCKET 3

Require a custom rewrite

This is where your budget actually goes. If you use highly customised or obscure extensions, they won't work. Your developers will need to open the hood, shift data logic from legacy frontend scripts into modern PHP ViewModels, rip out the old legacy JavaScript, and rebuild the interface from scratch using Alpine.js.

Getting this sorted before writing a single line of code is the only way to avoid surprise bills halfway through the project.

The Migration Complexity Matrix

Use this breakdown to estimate where your project falls on the spectrum of effort:

Complexity Level	Design Approach	Frontend Compatibility Reality
LOW EFFORT	"Lift-and-shift" using a pre-built Hyvä theme.	Backend extensions only, or all frontend extensions have existing Hyvä compatibility modules.
MEDIUM EFFORT	Customising a Hyvä base theme to match existing branding.	A mix of compatible modules and a small number of custom frontend features requiring limited Alpine.js refactoring.
HIGH EFFORT	Bespoke UI requiring heavy frontend design.	Heavy reliance on custom or legacy frontend code (e.g., KnockoutJS/jQuery) that requires significant rework for Hyvä compatibility.

How Much Does a Hyvä Migration Cost?

100–500+

DEVELOPMENT HOURS

The overall budget depends completely on engineering labor. To launch a tailored storefront that properly replicates your unique brand styling and business requirements, a professional migration typically demands a realistic scope of 100 to 500+ development hours.

Because of this, you must beware of suspiciously cheap implementation quotes. Some low-cost providers bid low by planning to deliver a standard out-of-the-box template with little more than a logo swap, meaning you risk paying more later to actually finish the custom setup.

Six Technical Bottlenecks That Stall Launches

You can't just install Hyvä and hope your old site architecture figures it out. Because you are fundamentally changing how Magento renders pages, several technical areas require a deep dive before you write a single line of code.

01 The Checkout Rebuild

This is almost always the heaviest lift of the entire project. Luma's default checkout is a notoriously fragile setup built on KnockoutJS and Magento UI components. If you have custom B2B shipping workflows, loyalty tier structures, or niche payment gateways, they won't just carry over.

> **PRO TIP**
>
> We recommend saving the checkout integration for the final phase of development, allowing the team to stabilize the main catalog pages first.

02 The Frontend Extension Trap

A massive chunk of standard Magento extensions is hardwired to look for RequireJS or legacy layout structures. Since Hyvä completely strips those out, any extension that displays something on the screen to your customers will require a compatibility module or a direct code rewrite.

03 Search and Layered Navigation

Advanced search bars, instant filtering, and infinite scroll utilities live and die by frontend JavaScript performance. Because these tools dynamically alter what the shopper sees on the page, their integrations require careful adjustment to match Hyvä's Alpine.js setup.

04 Leaving Legacy Code Behind

You cannot bring your old Luma custom theme modifications with you. Any bespoke layouts, heavily overridden template files, or custom CSS hacks built over the lazy years will need to be discarded. You are building the storefront presentation fresh.

05 CMS and Page Builder Drift

If your marketing team relies heavily on Adobe Commerce Page Builder or complex, customised CMS blocks to build landing pages, expect some formatting layout drift. Hyvä handles styles via Tailwind CSS, meaning old page builder layouts usually need a styling patch to render properly after the switch.

06 Tracking and Analytics Layers

Don't forget your data tracking. Google Tag Manager, analytics setups, ad pixels, and personalisation scripts all interact with the frontend DOM. They need to be audited and re-tested carefully to make sure they still capture data correctly without tanking the server speed gains you just paid for.

> **NOTE**
>
> Because Magento's routing engine remains intact, your core URLs and category paths won't change; you simply need to verify that your structured data and rich snippets align with your prior theme outputs.

Hosting & Infrastructure Requirements

While Hyvä significantly reduces frontend complexity, it doesn't magically solve every performance issue on its own.

A frontend can only render as quickly as the backend delivers data.

To fully benefit from Hyvä and achieve strong Core Web Vitals performance, the underlying Magento infrastructure also needs to be properly optimised.

> **IMPORTANT**
>
> To successfully build and run the theme, your environment must meet standard technical prerequisites: a modern PHP stack (version 8.1 or higher) and server-level access to Node.js/NPM to allow for Tailwind compilation workflows.

Server Response Time (TTFB)

Hyvä improves frontend rendering performance, but a slow Time to First Byte (TTFB) will still make the storefront feel sluggish.

To achieve consistently fast performance, the infrastructure should be optimised for Magento's workload patterns. This typically includes a tuned PHP stack, efficient database configuration, Redis support, and fast NVMe-based storage for handling backend queries.

Advanced Caching & CDN

Hyvä works well with standard Magento caching setups, but peak performance requires a Magento-aware caching strategy.

This includes properly configured Varnish and Redis layers, along with a CDN setup (such as Fastly or Cloudflare) that aligns correctly with Magento's cache invalidation behaviour to prevent stale or inconsistent content delivery.

Crucially, ensure your server-side caching is optimised to correctly process Hyvä ESI (Edge Side Includes) blocks for highly dynamic layout elements like the mini-cart.

Isolated Staging Environments

A Hyvä migration should never be performed directly in a live environment.

Reliable testing requires isolated, production-like staging environments where extensions, checkout flows, integrations, and frontend behaviour can be validated safely before deployment.

Managed Hyvä Hosting by Hypernode

Built by Magento experts, for Magento experts

Our hosting platform, built on over 10+ years of dedicated Magento experience, provides the exact server-side TTFB tuning, pre-optimised Varnish caching, and developer-friendly staging environments required to maximise your Hyvä store's speed safely.

Discover Hypernode →

SECTION 04

Hyvä Migration Roadmap

04

In most cases, the Magento backend remains largely unchanged. The primary effort is focused on rebuilding and optimising the frontend layer.

A typical Hyvä migration roadmap looks like this:

1

PHASE 1

Audit & Discovery

GOAL Eliminate surprises before development begins.

WORK The project starts with a full extension and frontend audit. The team catalogues all third-party modules and custom functionality to determine which features are backend-only, which already have Hyvä compatibility modules, and which require custom frontend development to shift logic into modern PHP ViewModels.

2

PHASE 2

Server & Environment Setup

GOAL Build a rock-solid server foundation so developers can work safely without touching the live store.

WORK Before writing code, the team spins up an isolated staging environment running PHP 8.1+ and Node utilities. If you're upgrading your hosting as part of this project, this is the window to get the new stack ready.

That means tweaking your PHP versions, configuring Redis and Varnish caching, and getting your CDN dialled in specifically for Magento. You want the server-side plumbing sorted before the new frontend code arrives.

3

PHASE 3

UX/UI Design

GOAL Get the look and feel of your store ready for the new framework.

WORK It doesn't matter if you're doing a pixel-for-pixel copy of your current site or a total redesign. Your frontend devs will build every button, navigation menu, and product grid using Tailwind CSS and Alpine.js. This is the stage where you completely ditch the old Luma layout files for good.

> **PRO-TIP**
>
> Share your Tailwind configuration file with your design team early on. Aligning layout designs with native utility classes upfront prevents developers from needing to write heavy custom CSS overrides down the line.

4

PHASE 4

Theme Setup & Foundation

GOAL Get the base theme running and lock in your speed baseline.

WORK This is where developers install the raw Hyvä theme and set up the build tools for Tailwind and Alpine. At this stage, the store is basically a blank slate, but it's incredibly fast.

You'll usually see near-perfect Core Web Vitals scores right here, before any custom code or extensions start slowing things down a little bit.

5

PHASE 5

Extension & Custom Development

GOAL Hook up all your features without breaking your new speed gains.

WORK This is the heaviest development phase of the whole project. Your team will spend this time dropping in community compatibility patches, rebuilding your bespoke features, making sure search filters actually work, and ripping out old KnockoutJS and RequireJS dependencies wherever they find them.

> **PRO TIP**
>
> Per our sequencing strategy, the core checkout is left untouched during this phase to maintain a clean build pipeline.

6

PHASE 6

QA & Testing

GOAL See if you can break the site in staging before your customers can break it in production.

WORK The team goes into full bug-hunting mode to make sure everything holds up. Instead of just clicking around casually, they stress-test the store across a few critical areas.

- Running Lighthouse speed checks to ensure the custom features didn't kill your performance,
- checking layouts on actual phones and tablets,
- pushing fake orders through the checkout to catch payment bugs,
- And making sure your tracking pixels and search metadata blocks are reading cleanly.

7

PHASE 7

Go-Live & Post-Launch

GOAL Flip the switch safely and make sure the site holds up under real traffic.

WORK This is launch day. The new frontend goes live on your production server. Before letting real traffic in, the devs make sure the CDN is behaving, old layout caches are completely wiped, and a cache-warming script is running so those first visitors don't hit a slow page.

Once the site is live, the team switches to monitoring actual user data, keeping tabs on conversion rates, and tracking real-world Core Web Vitals to catch any weird edge cases under load.

> **PRO TIP**
> Moving forward, establish a monthly routine to check for performance regressions and verify dependencies stay optimised.

SECTION 05

Protecting the Project

05

Risks and How to Mitigate Them

Every major platform upgrade carries inherent risk. The difference between a successful migration and a costly delay comes down to how those risks are anticipated and managed. Here is the reality of what can go wrong during a Hyvä replatforming and exactly how to protect your project.

The Migration Risk Matrix

The Risk	How to Fix It (Mitigation Strategy)
SEO Drop-off Your organic traffic and rankings tank because the new page structure or metadata layout throws off Google's web crawlers.	**Lock Down Your Data Early** Your URLs shouldn't change since the backend stays the same, but the page structure code will. Take strict SEO benchmarks before writing any code. Double-check your metadata, headings, and schema markup templates in staging, then baby-sit Google Search Console the second you go live.
Missed Deadlines The project stretches months past your target launch date, burning through your budget.	**Build an MVP, Not a Wishlist** Scope creep kills migration timelines. Do not use a frontend rebuild as an excuse to launch ten new business features at the same time. Migrate what you currently have first, get it stable, and build new tools after you're live.
Broken Tech Stack Third-party tools, payment gateways, or fulfilment apps stop talking to each other after the switch.	**Abuse Your Staging Environment** Third-party tools, payment gateways, or fulfilment apps stop talking to each other after the switch. Refactor core layout logic into PHP ViewModels and use an Alpine-driven framework. Abuse your staging sandbox to validate API syncs.
Cart Abandonment Spikes Sales dip because the new checkout flow behaves weirdly, the layout shifts, or it glitches on mobile browsers.	**Isolate your checkout deployment** Keep the core theme build and catalogue milestones decoupled from transactional checkout changes to guarantee path stability, testing mobile payment contexts heavily before production.

The Risk	How to Fix It (Mitigation Strategy)
Bloat Creeps Back In Your fast new store gets dragged down by marketing pixels, tag managers, and heavy personalisation scripts.	**Enforce Strict Script Audits** Treat this migration as a spring cleaning. Delete any tracking scripts your marketing team isn't actually looking at. For the ones you keep, defer them so they load after the main page renders, keeping the site snappy.
Slow Hosting Bottlenecks You don't see the massive speed jump you paid for because your backend server infrastructure is too slow to keep up.	**Fine-Tune Your Caching** Ensure server arrays meet baseline environment dependencies (PHP 8.1+). Optimise your database, tweak your Redis and Varnish configurations, and get your CDN dialled in specifically for Magento traffic before you flip the switch.

CONCLUSION

The Bottom Line on Your Hyvä Move

Preparation is the Key to a Smooth Migration

Switching to Hyvä is one of the single best moves you can make for a Magento storefront right now. It finally cuts the anchor that is Luma, rescues your Core Web Vitals, and gives your developers a clean, modern framework they'll actually enjoy working in.

But let's be entirely realistic: this isn't a "plug-and-play" theme swap. It is a full-scale frontend rebuild.

A smooth, on-budget migration doesn't depend on the framework itself. It depends entirely on how well you do your homework before writing any code:

- How accurately you map out your store's current technical quirks.
- How ruthlessly you audit your existing extensions and scope their move to PHP ViewModels.
- How well does your hosting environment support the new architecture and core software versions?

Sorting these variables out upfront completely kills the guesswork, giving you a realistic project scope, a solid timeline, and a budget that won't suddenly double halfway through.

Your Immediate Next Steps

Do not try to estimate your migration based on assumptions or a generic agency template. To get a realistic, bulletproof scope, look at your current setup today:

1. **Map out your custom code:** Identify every custom frontend feature you've built over the years, especially around the checkout flow, product page galleries, and complex search filters.

2. **Run a plugin headcount:** Catalogue every single third-party extension currently running on your site. Sort them into buckets: what's already Hyvä-ready, what needs a community patch, and what requires a total rewrite from scratch.

3. **Audit your hosting environment:** Talk to your hosting partner. Make sure your infrastructure fulfils core PHP 8.1+ environment guidelines and can handle the staging workflows, aggressive caching tweaks, and performance testing needed to pull this off safely.

Hypernode | Hyvä

Don't put a fast frontend on a slow server.

You are investing in a Hyvä migration to achieve top-tier performance. But Hyvä can only render as fast as the backend delivers data.

Frontend performance is only one part of the equation. If the underlying infrastructure is not optimised, the full benefits of Hyvä will not be realised.

Hypernode helps teams ensure their Magento infrastructure is ready to support a modern Hyvä frontend. Whether during migration planning or after launch.

If your goal is to maximise the return on your Hyvä implementation, it is worth reviewing whether your current hosting setup is helping or limiting that outcome.

Talk to us about optimising your Hyvä infrastructure →