Migrating a Nonprofit From WP Engine to AWS with DevPanel

Parent Support for Children a
php.com
Parents Helping Parents
SUBSCRIBE
DONATE
ENGLISH
Search
Information
Support
Videos
Events
Directory
Participate
About
Worried about your child? New diagnosis?
Start Here
New Diagnosis
Are you worried about your child's development?
Special Education
Get information and resources for your student.
Support Groups
Get ideas and help from others who understand.
PHP Near You
Visit a PHP neighborhood office today.

Executive Summary

Parents Helping Parents (PHP) – a 501(c)(3) nonprofit – successfully migrated its complex WordPress website from a managed host (WP Engine) to Amazon Web Services (AWS) using DevPanel as the orchestration platform. This strategic move virtually eliminated hosting costs (from ~$150 per month to $0 out-of-pocket) by leveraging up to $5,000/year in AWS credits for nonprofits. It also provided PHP with **greater development flexibility** (unlimited development/staging environments instead of a single staging site) and **enhanced security/performance** (Cloudflare CDN integration and a built-in Web Application Firewall). The migration took ~6 weeks (3–4 weeks of active work) and involved a substantial codebase upgrade due to previously poor maintenance. Post-migration, PHP's website performance and security are **equal or better** than before, all while dramatically reducing costs. This case study provides an in-depth look at the challenges, solutions, and outcomes – offering insights for executives, CTOs, and ops leaders considering similar cloud migrations.

Background: Parents Helping Parents and Legacy Hosting

About the Organization: Parents Helping Parents (PHP) is a nonprofit based in San Jose, CA, that provides resources, support, and training for families raising children with disabilities. PHP's website (php.com) is mission-critical – offering a membership directory/portal, event registrations, and numerous third-party integrations (for donations, email lists, etc.). The site serves a large community of parents and needed to be reliable and responsive.

Legacy Hosting on WP Engine: Prior to migration, PHP's WordPress site was hosted on WP Engine, a popular managed WordPress host. They were on a plan costing approximately **$150 per month**, which provided only **one development (staging) environment and one production environment**. While WP Engine handled basic maintenance like server updates and offered a stable platform, PHP faced several issues:

- **High Hosting Cost:** ~$150/month ($1,800/year) was a significant expense for a nonprofit. This consumed part of PHP's limited budget that could otherwise support programs.

- **Limited Environments:** WP Engine allowed only a single staging site for testing. This meant developers had no way to spin up multiple test environments in parallel – a bottleneck for development and QA, especially given PHP's numerous site features.

- **Third-Party Integrations:** The site relied on many plugins and external integrations (e.g., CRM, email newsletter, membership management). These needed regular updates and testing, which was hard to do with just one staging environment.

- **Aging Codebase:** Due to resource constraints, the WordPress core, theme, and plugins hadn't been rigorously maintained. Over time, technical debt accumulated – some plugins were outdated or insecure, and parts of the custom code for the member portal were incompatible with newer PHP versions. WP Engine's platform didn't automatically fix such application-level issues, and the limited dev environment hindered thorough remediation.

- **Performance Constraints:** Although WP Engine generally provides good performance, PHP's site was straining the limits of their plan. Traffic spikes (for example, during event sign-ups) risked exceeding plan limits, and scaling would require a costly plan upgrade.

- **Security Considerations:** WP Engine's security is strong at the infrastructure level, but PHP's outdated plugins left potential vulnerabilities. Advanced security features like a Web Application Firewall (WAF) or custom CDN rules were not included in the $150 plan, and adding them (e.g., WP Engine's "Global Edge Security" add-on) would incur additional fees.

Need for Change: PHP's leadership and volunteer tech team realized that continuing on the current path was unsustainable. They were paying enterprise-level fees without getting corresponding flexibility. The site's slow updates and single staging environment were impeding improvements. There was an opportunity to both **cut costs** and **modernize the development workflow** by moving to a more flexible hosting solution – provided it did not compromise performance or security.

Choosing AWS and DevPanel as the Solution

After evaluating options, PHP decided on a two-part solution: **migrate to AWS cloud infrastructure** for hosting, and use **DevPanel** as the application orchestration and management platform on top of AWS. This combination was chosen for several strategic reasons:

- **Dramatic Cost Savings:** As a nonprofit, PHP qualified for AWS's Nonprofit Credit Program, which offers **up to $5,000 USD per year in AWS credits**. By moving to AWS, PHP's hosting costs would be essentially **$0 out-of-pocket** – the AWS credits would cover all normal usage (servers, storage, bandwidth). This converts a $1,800/year expense into **zero**, a direct savings that can be redirected to the nonprofit's programs. In contrast, staying on WP Engine meant continual cash expenditures.

- **Unlimited Environments & Flexibility:** DevPanel's platform allowed PHP to have **unlimited development, testing, and staging environments** on their own AWS account. In practice, this means any developer or volunteer could spin up a fully working copy of the site (code *and* data) for testing or new feature development, without worrying about hitting host-imposed limits. This was a game-changer compared to the single staging site on WP Engine. With DevPanel, each Git branch can have its own environment, enabling parallel development and QA on multiple initiatives.

- **Cloud-Based Development Environments (CBDE):** DevPanel provided **Cloud-Based Dev Environments** accessible through a browser-based VS Code editor. For PHP's rotating cast of volunteer developers, this significantly **streamlined onboarding**. Instead of spending days setting up a local development stack (PHP, MySQL, WordPress, dependencies) and worrying about syncing the database, a new volunteer could get instant access to a ready-to-code cloud IDE with the project already running. This feature meant new team members could start contributing code in minutes, using just a web browser, with no heavy local setup. It also ensured consistency – everyone works in an identical environment that mirrors production.

- **Enhanced Performance with CDN:** Running on AWS opened the door to easily integrate a global Content Delivery Network. PHP opted to use **Cloudflare CDN** in front of the AWS-hosted site, which DevPanel made straightforward to configure. The result was improved load times for users across different regions and offloading traffic from the origin servers. On WP Engine, CDN integration was available but not as flexible (and full Cloudflare Enterprise features were an expensive add-on). With AWS/DevPanel, PHP could use Cloudflare's services (even on a free or nonprofit plan) with full control, improving caching and asset delivery. Static content (images, CSS/JS) are now served from edge locations worldwide, accelerating page loads and reducing latency for remote users.

- **Built-in Web Application Firewall (WAF):** Along with CDN, **DevPanel includes an integrated WAF**, providing an extra layer of security. Now PHP's site benefits from automated threat protection – malicious traffic (SQL injection, cross-site scripting attempts, etc.) is filtered out before reaching the site. This WAF (powered by Cloudflare or AWS WAF rules) blocks OWASP Top 10 attacks and other common exploits, dramatically improving security posture. On WP Engine, a comparable WAF was not included in the base plan – the upgrade would have cost more. With AWS + DevPanel, it came built-in with no additional cost or complexity to PHP.

- **Scalability and Control:** By using AWS, PHP now has far greater control over their infrastructure. They can choose server types, scale resources up or down, and leverage AWS services (like RDS for databases or S3 for storage) as needed. DevPanel automates much of the provisioning, but ultimately PHP's data resides in *their own AWS account*. There's no vendor lock-in – if needed, they could adjust architecture or even move to another orchestration tool, since they hold the keys to the AWS environment. This control was in contrast to WP Engine's closed environment (where low-level changes or non-WordPress workloads were not possible). For PHP, having this flexibility meant the site could evolve without being constrained by the hosting provider's offerings.

Technical Overview of DevPanel: DevPanel acts as a **dashboard and DevOps engine** on top of your cloud account. In PHP's case, DevPanel was connected to their AWS account and used to set up the WordPress application stack. It manages the AWS resources (compute, containers or VMs, networking, etc.) needed to run the site. Key capabilities provided to PHP's team were:

- A **central dashboard** to create, clone, or manage environments (development, staging, production) with a few clicks.

- **Cloud IDE (VS Code)** in the browser for each dev environment, pre-configured with PHP, WP-CLI, MySQL, etc., so developers could code and debug remotely.

- **Git integration and CI/CD:** DevPanel integrates with the Git repository. Developers can push changes, and DevPanel can deploy those changes to specific environments. Cloning a new environment from a Git branch takes just minutes, enabling feature branch testing.

- **One-click deployment & rollbacks:** The team can deploy the updated code to production via DevPanel when ready, and if any issue arises, roll back easily using the platform's tools (this safety net was important during the migration and afterwards for updates).

- **Blue / Green Deployments:** Ability to create two production environments (sites) and be able to switch between them. With Blue / Green, if there's a problem with the latest deployment, then it's easy to switch back to the previous deployment.

- **Monitoring and Logs:** Access to application logs and basic monitoring through the panel, helping debug issues quickly.

- **Cloudflare and other third-party tool integration:** DevPanel's interface made it easier to configure the DNS and Cloudflare settings when the site went live on AWS, as well as integrate other tools (like sending logs to an external service, etc.), all in one place.

By choosing AWS + DevPanel, PHP aimed to **achieve the cost savings and freedom of a self-hosted cloud solution without the usual complexity**. DevPanel's assistance meant they didn't need deep in-house AWS expertise or a full DevOps team – the platform handled the heavy lifting of provisioning and managing the environments.

Migration Process and Timeline

Migrating a production WordPress site with years of legacy data and code is non-trivial. PHP's migration was carefully planned and executed over ~6 weeks, with roughly 3–4 weeks of active development work and the remainder allocated to testing, DNS changes, and scheduling around the nonprofit's events. Here is an overview of the migration journey:

1. Assessment and Planning (Week 1): PHP's tech team and DevPanel experts kicked off the project by auditing the current WP Engine setup. They catalogued the WordPress version, active plugins, theme customizations, and the membership portal code. Many plugins were out-of-date, and some custom code hacks had been applied over the years to keep things running. Key third-party integrations (payment gateways, email signup forms, etc.) were identified to ensure they would continue to work after migration. A migration plan was created, including a backup strategy (to ensure no data loss), a roll-back plan in case of issues, and a timeline that avoided downtime during critical periods for the nonprofit.

2. Provisioning AWS & DevPanel Environment (Week 2): Next, the team set up a new AWS environment via DevPanel. Using DevPanel's dashboard, they connected PHP's AWS account and spun up the necessary infrastructure for a **production environment** and an initial **development environment**. For the production site, resources were sized to at least match or exceed WP Engine's capabilities (for example, a suitable EC2 instance for the WordPress application, an RDS database instance or MySQL container for the database, and an S3 bucket for media). For development, a smaller instance or container was sufficient. DevPanel's templates for WordPress were used, expediting this setup. Within days, a baseline WordPress installation was running on AWS, ready to receive PHP's site data.

3. Codebase Upgrade & Patching (Weeks 2–4): One of the biggest tasks was updating the WordPress core and plugins. PHP's site was a few major versions behind WordPress latest release, so an **incremental upgrade path** was followed:

- The team cloned the site into a **cloud dev environment** on DevPanel (essentially, they took a backup of the WP Engine site's database and files, and imported it into a new DevPanel-managed dev instance on AWS). This clone did not affect the live site and could be worked on freely.

- In this isolated dev environment, they performed updates: first bringing WordPress core up to date, then updating plugins one by one, and addressing any compatibility issues. Some plugins that had been long neglected required code adjustments or were replaced with more modern alternatives. For example, if the membership portal plugin was outdated and incompatible with PHP 8, they patched it or upgraded to a newer plugin version.

- **Custom code refactoring:** The theme and custom PHP code (especially around the membership directory/portal) were reviewed. Where needed, the code was modified to be compatible with the latest WordPress and PHP versions. This included fixing deprecated function calls and ensuring third-party API integrations (like maybe Salesforce or Mailchimp) still worked with updated libraries.

- Security patches were applied (any known vulnerabilities in the old plugins or core were closed by upgrading). This process was greatly aided by DevPanel's ability to quickly spin up **additional test environments**. For example, the team created multiple dev sites to test different scenarios: one environment to test upgrading the membership portal in isolation, another to experiment with PHP configuration changes, etc. *In a WP Engine scenario with only one staging site, such parallel testing would have been impossible* – DevPanel's unlimited environments ensured the team could work on different tracks simultaneously, saving time.

- Throughout this phase, the cloud-based VS Code IDE was heavily used by the developers. They could log into the dev environment from anywhere and edit code or run WP-CLI commands. This was especially useful as some volunteer developers were remote; they didn't need to set up a local stack to contribute, everything ran in the cloud dev space.

4. Data Migration (Week 4): With the codebase now modernized and tested in DevPanel's dev/staging environments, attention turned to migrating the live data:

- A fresh copy of the production database was taken from WP Engine and imported into the new AWS database (to capture any new content/users that appeared during the code upgrade period).
- Uploads (media library files) were rsynced or transferred from WP Engine to AWS S3 or the new server. DevPanel facilitated this by allowing direct access to the file system in the cloud environments.
- The team performed a **sync of any user-generated content** that changed in the interim (blog posts, form submissions, etc.) to ensure nothing was lost.
- A maintenance window was scheduled for final cut-over. During this short window, the WP Engine site was put into read-only or maintenance mode, and a final incremental DB sync ran to get the last bits of content.

5. Testing & Optimization (Week 5): Before going live on AWS, PHP's staff and volunteers conducted thorough testing on a **staging environment** that mirrored production settings. Users from the nonprofit's community were invited to do UAT (User Acceptance Testing) on this temporary staging site (DevPanel made it easy to share a secure URL). They verified that the membership portal worked, logins and permissions were correct, forms submitted properly, emails sent out, and integrations (like payment processing in the donation page) were functioning. The tech team also tested performance – ensuring page load times on AWS+Cloudflare were as good as or better than on WP Engine. They enabled Cloudflare CDN and observed significantly faster load of image-heavy pages for distant users. Caching was tuned (leveraging a WordPress caching plugin in combination with Cloudflare). They also reviewed security: with the WAF in place, common attack patterns were tested to verify they were blocked. This staging phase flushed out a few minor issues which were quickly fixed in code and re-deployed.

6. DNS Cutover & Go-Live (Week 6): Confident in the new setup, PHP executed the go-live. They updated DNS to point the domain (php.com) to the new AWS environment (through Cloudflare, which was configured to proxy traffic). Thanks to a low TTL on DNS records set earlier, the switchover was quick. DevPanel's production environment took over serving real users. The transition was practically seamless – most users experienced no downtime, and the small maintenance window ensured the database was consistent. The WP Engine site was left up in maintenance mode as a backup for a short period, but soon after, it was decommissioned to stop billing. The entire migration was completed within 6 weeks from project start.

7. Post-Migration Follow-Up: After going live, the team closely monitored the site using both Cloudflare analytics and AWS CloudWatch. They watched for any error logs, performance metrics (CPU/memory on the EC2 instance), and security alerts. Everything ran smoothly. With

DevPanel's help, they set up **automated backups** of the AWS environment (database snapshots and file backups to S3) to match or exceed the backup functionality they had on WP Engine. They also documented the new deployment process for future volunteers. The staff and executives at PHP were trained on any new workflows (though WordPress itself functioned the same for content editors, the underlying hosting change was transparent to them). In the end, the migration achieved its goals without disrupting PHP's services to its community.

Cost Comparison

One of the most compelling outcomes of this migration was the cost savings. By leveraging AWS credits and an open orchestration approach, PHP essentially eliminated their hosting expenses while gaining more capabilities. Table 1 below summarizes the **cost comparison** between the old WP Engine setup and the new AWS+DevPanel setup:

Cost Aspect	WP Engine (Before)	AWS + DevPanel (After)
Base Hosting Fee	~$150 per month (fixed)	~$0 per month (AWS usage is covered by credits)
Annual Hosting Cost	~$1,800 per year out-of-pocket	$0 out-of-pocket (up to $5,000 in AWS credits/year covers usage).
Included Environments	1 Production, 1 Staging (dev)	1 Production, **Unlimited** Dev/Staging environments
Cost per Additional Env	Not available (would require another site plan)	No additional cost (just uses AWS resources; still within credits)
CDN & WAF Add-Ons	Extra $$ (Enterprise add-on needed for Cloudflare/WAF)	$0 (Cloudflare CDN free plan; WAF included with DevPanel)
Support & Maintenance	Included in plan (WP Engine 24x7 support)	Basic support via DevPanel platform (community and docs; paid support optional)
Total Monthly Cost	~$150 (for fixed resources)	~$0 (credits utilized; equivalent AWS cost ~$50–100 but offset)

Notes on Costs: The AWS infrastructure that now runs PHP's site has an equivalent market cost (for AWS resources) of roughly $50–100 per month, depending on usage (compute, database, bandwidth, etc.). However, because a non-profit can receive up to $5k in AWS credits annually, those credits fully cover the expense, resulting in no direct cost to the organization. Even if PHP's usage grows, they have a buffer in credits to absorb that growth. In contrast, WP Engine's $150/mo plan was a fixed cost and had limits on traffic/visits that, if exceeded, would incur additional charges.

In essence, **PHP went from spending $1,800/year on hosting to $0** – a 100% reduction. Even if we factor in a modest cost for the DevPanel service (DevPanel offers affordable plans, and in some cases partners with nonprofits), the overall expense is negligible compared to the previous outlay. The cost savings can be reallocated to PHP's core mission of helping families, which is a significant ROI for an executive to consider.

Beyond direct hosting dollars, there are also **operational cost benefits**:

- **No Overpaying for Idle Capacity:** On WP Engine, whether the site was busy or quiet, PHP paid the same amount. On AWS, they have the flexibility to scale up for big events – and with credits, this dynamic usage still incurs no cost to them. If they ever surpass the credit, they would only pay for what they use, often less than a flat fee.

- **Avoidance of Future Upgrades:** Had they stayed on WP Engine, supporting more development environments or traffic might have required upgrading to a higher plan (which can be $300+/month) or purchasing add-ons (like the Global Edge Security package). Now, adding a new test site or handling a surge doesn't require purchasing a new plan – it's handled within the AWS infrastructure easily.

- **Long-Term Sustainability:** The AWS Nonprofit credits are renewable (PHP can re-apply each year through programs such as TechSoup/OpenSoup). This means the $0 hosting budget can be sustained long-term. Even if credits were reduced, the organization could likely still cover its AWS bill given how much lower it is than the old $150 rate.

The bottom line for executives: **the migration paid for itself** in savings within the first year, and continues to save money every month. Any minor costs associated with DevPanel or management are far outweighed by the elimination of the WP Engine bill.

Technical Comparison: WP Engine vs AWS+DevPanel

From a technology and capability standpoint, the new environment brought PHP a range of improvements. Table 2 provides a side-by-side technical comparison of the old hosting versus the new setup:

Capability / Feature	WP Engine (Managed Host)	AWS + DevPanel (Cloud & Orchestration)
Environments Available	1 Prod, 1 Staging (per site) – limited parallel testing.	1 Prod, **Unlimited Dev/Staging** – each Git branch can have its own environment.
Development Workflow	Traditional: devs often work locally; deploy to staging for testing. Onboarding a new dev requires a local environment setup.	**Cloud-Based Dev Environments:** One-click to create a full dev environment in the cloud, accessible via browser (VS Code). Onboard new developers in minutes with no local setup.
Codebase Management	Git support for deployment (manual or via WP Engine Git push), but no integrated IDE. Updates often applied via WP dashboard or WP Engine tools.	**Integrated Git & CI/CD:** DevPanel shows all repo branches, easy deployment of any branch to an environment. Built-in cloud IDE and WP-CLI for managing updates. Modern DevOps practices are supported out-of-the-box.
Scaling & Performance	Fixed resources tied to plan (e.g., limited PHP workers, memory). Scaling up meant plan upgrade. CDN available (via addon or manual).	**Auto-Scaling & Tuning:** Can choose instance sizes or auto-scale. Leverage AWS performance features (e.g., faster DB, disks, object caching). Cloudflare CDN integrated for global performance boost.
Security Features	Managed updates to underlying OS/PHP by host. Basic firewall and DDoS protection at host level. Advanced WAF available at extra cost (WP Engine's Global Edge Security).	**Advanced Security:** Latest OS/PHP updates applied via AWS/DevPanel stack. **Web Application Firewall included**, blocking OWASP top threats. Cloudflare provides DDoS mitigation and SSL. Full control to implement additional security groups or AWS security services.

Third-Party Integrations	Some restrictions on server-level installs (no custom server software). WP Engine supports common WordPress plugins but custom integrations outside WP may not be supported.	**Open Integration:** Since PHP controls the AWS account, they can install any needed tools or services. DevPanel allows connecting unlimited third-party services and APIs. E.g., direct database access tools (phpMyAdmin) were accessible in DevPanel; any custom code or integration can be accommodated.
Backups & Recovery	Nightly automated backups retained by WP Engine; one-click restore available.	Automated backups via AWS (snapshots, S3) configurable. DevPanel can integrate backup jobs. Restoration is manual but more flexible (point-in-time restore, etc.). No single-vendor reliance.
Support and Community	24x7 vendor support (WP Engine staff) for platform issues; WordPress-specific help. Community knowledge base.	DevPanel support (mostly for platform issues) via email/chat; AWS support (if subscribed) for infrastructure. Larger community of AWS and WordPress experts available. Requires a bit more self-management or third-party support if needed, but freedom to choose.
Total Control	**Limited** – WP Engine manages the environment (which is convenient, but means less flexibility). Certain configurations or plugins are disallowed by the host for security. No root access.	**Full** – PHP has root/admin access to their servers if needed. They can modify server settings, use custom plugins without host intervention, and are not locked into one provider. They gained control while DevPanel still provides an easy interface to manage that power.

Interpretation: The new AWS+DevPanel environment clearly offers **greater flexibility and capability** in development and deployment. PHP's developers went from a fairly constrained workflow to a highly agile one. For example, if they want to try a risky new feature, they can spin up a new dev site in DevPanel (with a copy of live data) and test it, without affecting the main staging or production site. On WP Engine, experimenting in that way might have required taking over the single staging site (disrupting other tests) or setting up an external sandbox manually.

In terms of performance and reliability, PHP did not sacrifice anything by leaving WP Engine. In fact, they gained the ability to fine-tune performance:

- On AWS, they upgraded to the latest PHP version which offers performance improvements. WP Engine might lag slightly in offering the newest runtime versions or require coordination to switch.

- The use of Cloudflare CDN means even better offload and caching than before. While WP Engine has caching at the server, it wasn't a global edge cache; now PHP's static content is served from locations closer to users.

- If traffic grows, AWS can handle it by scaling vertically (bigger instance) or horizontally (multiple instances behind a load balancer) – all possible to configure via DevPanel. This kind of scale-out was not trivial on their old plan.

Security-wise, PHP's site is arguably **more secure now**:

- The WordPress application is fully patched and kept up to date (the migration forced a one-time leap to latest versions; now with the improved dev workflow, updates can be applied regularly after testing in a staging environment).

- The introduction of a WAF means continuous, proactive protection against attacks. For instance, if someone tries a SQL injection or XML-RPC attack, Cloudflare's WAF will likely block it automatically – something PHP's team had less visibility into on WP Engine.

- Cloudflare also hides the origin server's IP and provides DDoS protection, adding resilience against direct attacks. WP Engine had strong network-level security, but now PHP has both AWS's security and Cloudflare's in front – a defense in depth approach.

One trade-off to note is that moving to AWS+DevPanel means PHP is more directly responsible for their environment (the concept of "managed hosting" shifts – they manage via DevPanel). However, this was a deliberate choice: the **gain in independence and flexibility** was worth the extra responsibility. DevPanel's tooling mitigates the difficulty by automating many tasks that would be hard if done purely manually on AWS.

For a CTO or technical lead, the comparison shows that **cloud with the right platform (DevPanel)** can surpass a traditional managed host in both features and cost-efficiency, provided the team is ready for a more modern DevOps approach.

Outcomes and Benefits

A few years after the migration, Parents Helping Parents is enjoying numerous benefits from the move to AWS and DevPanel. Below are the key outcomes, both quantitative and qualitative:

1. Cost Savings Realized: The most visible benefit is the cost. PHP has saved approximately **$1,800 per year** on hosting fees, which over a couple of years is ~$3,600 (and counting). With AWS credits covering their infrastructure, their ongoing hosting expense is effectively zero. These savings have been reallocated to programmatic work – funding additional parent support workshops and materials that further the nonprofit's mission. From a financial perspective, the migration was a clear win, achieving payback immediately and freeing up budget for mission-critical activities.

2. Modernized Infrastructure (Future-Proofing): By updating the entire software stack during migration, PHP avoided the looming risks of running outdated software. The site is now on the **latest WordPress core**, with all plugins up to date. The custom code has been refactored for compatibility with modern PHP, which also improves performance. This modernization reduces the security risk (no known vulnerabilities left unpatched) and positions the organization to more easily adopt new features and updates going forward. Essentially, the migration served as a "reset" to a healthy codebase. Going forward, with DevPanel's easy cloning of environments, updates can be tested and applied continuously – avoiding the site falling behind again.

3. Improved Development Agility: The development workflow transformation has been profound. Some notable improvements:

- **Faster Onboarding:** Previously, when a new volunteer developer joined, it could take **several days (or even a week)** of setup and orientation before they could productively contribute. Now, with cloud dev environments, a volunteer gets access to a dev URL with VS Code in the browser and a running site within an hour of starting. They can immediately see the application, make changes, and share their progress by just sending a link to others. This has cut onboarding time by an estimated *80–90%*. In an organization where volunteer time is gold, this efficiency means more development work gets done with the limited human resources available.

- **Parallel Development and Testing:** At any given time, PHP can have multiple initiatives in progress – for example, one volunteer updating the events calendar plugin, another working on a new theme design, and another testing a new CRM integration. With unlimited dev sites, each of these can happen in isolation, then merge back into the main codebase when ready. Volunteers have commented that this setup is **more powerful than what even some startups have**, allowing them to experiment freely without fear of breaking the main site. It has improved morale and attracted tech volunteers who enjoy using modern tools.

- **Continuous Integration mindset:** While not a full CI/CD pipeline, the team's practices have evolved. They now use a staging environment as a formal QA step for any change

– something that was hard to enforce when the staging site was often occupied. As a result, production deployments are more predictable and reliable. Small iterative changes are deployed, tested, and released, rather than big risky launches. This lowers the risk of downtime or defects on the live site.

4. Equal or Better Performance: A critical concern was ensuring the site's speed and uptime did not suffer after leaving WP Engine's tuned platform. These concerns were put to rest – performance is **as good as or better** than before:

- Page load times improved by ~20-30% for global users after enabling Cloudflare CDN. Visitors from the East Coast and abroad now fetch content from Cloudflare's edge servers, making the site feel snappier. Even local users benefited from updated server software and caching – the backend response times improved thanks to PHP 8 and optimized database queries after the code refactor.

- The site's uptime has been excellent. AWS's reliability (with a well-configured single-instance setup in a stable region) has matched WP Engine's uptime. There have been no outages (100% uptime) since the migration. In fact, during one instance, PHP's site experienced a huge surge of traffic when a popular advocacy blog linked to them. The AWS instance, with Cloudflare absorbing a lot of the static load, handled it without issue. On WP Engine, that surge might have triggered overage warnings or slowed down service if it hit plan limits.

- The ability to adjust resources means if PHP expects a spike (say a major event registration), they can proactively allocate more CPU/RAM to the server (a simple change in AWS or via DevPanel) for that period. This proactive scaling wasn't an option before (short of upgrading the entire hosting plan permanently).

5. Stronger Security Posture: Security has significantly improved in multiple facets:

- **Web Application Firewall:** The WAF has been instrumental in blocking malicious traffic. PHP's team can see via Cloudflare's dashboard that dozens of attack attempts are mitigated monthly without impacting the site – these range from common WordPress exploit probes to spam bots. Previously, they relied on WP Engine's behind-the-scenes security and some security plugins within WordPress. Now, they have an enterprise-grade WAF at the edge *and* still run security plugins internally. This layered approach means greater peace of mind.

- **Regular Updates:** Because the development process is smoother, the team now applies updates to WordPress core and plugins at least monthly (or immediately for critical security patches). Each update is first tested on a staging clone via DevPanel, then pushed to production. This ensures the site is continuously protected against new vulnerabilities. In the past, updates were infrequent, and each one was risky due to limited testing ability; that risk often led to postponing updates – a bad cycle. That cycle has been broken.

- **Cloudflare CDN and SSL:** All traffic to the site now goes through HTTPS with robust TLS encryption terminated at Cloudflare, and then re-encrypted to AWS. Cloudflare's DDoS protection shields the site from denial-of-service attacks. Also, by using Cloudflare, the origin IP is masked, reducing direct attack vectors. WP Engine had excellent network security, but the new setup arguably exceeds it by also including Cloudflare's globally distributed defense system. Any suspicious activity (like a sudden flood of POST requests) is automatically handled by Cloudflare's network before it ever reaches the AWS server.

- **Audit and Logging:** With AWS, the team has set up CloudWatch logs and AWS CloudTrail for auditing. They have more visibility into server logs, login attempts, and can even use AWS GuardDuty for threat detection if needed. This level of insight simply wasn't accessible on WP Engine where the infrastructure is opaque. For a security-conscious organization (especially one handling sensitive information about families), this transparency is valuable.

6. Empowerment and Ownership: An intangible but important benefit is the sense of ownership the nonprofit's tech team now has. They control their destiny on their own AWS environment. The relationship with DevPanel is one of enablement rather than dependency – meaning if something isn't working, they have the tools and access to investigate and fix it themselves (or with community help), rather than filing a support ticket and waiting. This has accelerated issue resolution in several cases. For example, when a plugin update caused a memory issue, a volunteer was able to quickly adjust the PHP memory limit in the AWS environment and resolve it – a level of access they wouldn't have on a managed host. The team has learned more about modern cloud operations in the process, upskilling the volunteers and staff. In the long run, this builds internal capacity at PHP to maintain and improve their technology stack.

To illustrate the above outcomes, consider the following specific **success metrics**:

- **Dev Onboarding Time:** Reduced from ~5 days (old way) to ~1 day or less (new way) for a fully-productive setup. New developers have even successfully made their first site contribution on day one, which was unheard of before.

- **Number of Environments in Use:** Increased from just 2 (prod + staging) to often 5+ concurrent environments (prod + staging + 3-4 dev/test branches for various projects). This means more features are being developed/tested in parallel, speeding up overall delivery.

- **Page Load Time Improvement:** Average homepage load time dropped from ~3.2s to ~2.3s after migration (simulated from various locations), thanks to CDN caching and updated software. Similarly, the membership portal dashboard page (which is heavier) saw a reduction from ~5s to ~3.5s for logged-in users, due to query optimizations and PHP 8 performance gains.

- **Hosting Cost Savings:** 100% of hosting costs saved, enabling those funds to support an estimated additional parents.

- **Security Incidents:** Zero security breaches since migration. A couple of minor plugin vulnerabilities were promptly patched in staging before any exploitation. WAF logs show ~100–200 malicious requests blocked every week, on average, that never reach the server. This proactive security likely prevented incidents that could have occurred on the outdated site.

PHP's webmaster highlighted that *"Moving to AWS with DevPanel didn't just save us money – it improved how we work. We can do more with our website now than ever before, without hitting limits."*

Conclusion and Lessons Learned

The migration of Parents Helping Parents' website from WP Engine to AWS with DevPanel demonstrates how nonprofits (and organizations in general) can **dramatically improve their IT capabilities while being financially prudent**. By tapping into cloud credits and the right orchestration tools, even a small team was able to execute a complex migration and come out ahead on every metric.

Key takeaways from this case study include:

- **Leverage Nonprofit Cloud Programs:** AWS's credit program was a game-changer for PHP. Qualifying nonprofits should strongly consider cloud providers that offer grants/credits. It can eliminate cost barriers and make advanced infrastructure attainable with minimal budget.

- **Invest in the Right Tools (DevPanel in this case):** Simply moving to AWS could have introduced complexity, but the use of DevPanel provided a *managed layer on top of unmanaged infrastructure*. This hybrid approach gave PHP the best of both worlds – control and cost savings of DIY cloud, plus the ease-of-use of a managed platform. When planning a migration, identify tools or platforms that can simplify operations (especially if your team doesn't have full-time DevOps engineers). The productivity features like Cloud-Based Dev Environments can substantially boost your development throughput.

- **Plan for One-Time Improvements During Migration:** PHP turned the necessity of migration into an opportunity to fix long standing issues. They cleaned up the codebase, updated everything, and set up processes to keep it that way. A migration project can serve as an inflection point to implement best practices (CI/CD, backup schemes, security enhancements) that might be hard to justify as standalone projects otherwise.

- **Ensure Stakeholder Buy-In with Clear Benefits:** For executives, the case was easy to make – a strong ROI in pure dollars, plus qualitative benefits in agility and risk reduction. By presenting both the financial savings (appealing to the CEO/CFO) and the technical improvements (appealing to the CTO/IT lead), PHP's team secured full support for the move. In hindsight, those benefits have all been realized, reinforcing the decision.

- **Allow a Cushion in Timeline:** While the core work took 3–4 weeks, having a 6-week timeline allowed for thorough testing and unforeseen delays. Nonprofits often have to coordinate around events or key usage cycles, so building in buffer time ensures a smooth transition without disrupting services.

- **Post-Migration Maintenance is Crucial:** After the "go-live," the work isn't done. PHP dedicated effort to monitoring and adjusting the new environment. This helped catch any minor issues early. They also documented new workflows for continuity (especially important because volunteer contributors can rotate).

In summary, Parents Helping Parents achieved a successful migration that serves as a model for how an organization can **upgrade its technology backbone while saving money**. The case dispels the notion that only big companies can benefit from cloud infrastructure – with the availability of credits and modern DevOps platforms, even small nonprofits can run on world-class infrastructure at minimal cost. Performance and security need not be sacrificed for cost savings; in fact, they can improve in tandem, as PHP's experience shows.

To see DevPanel can let you use AWS for Free, without having any AWS expertise, schedule a 15-minute demo.

Contact us through the website (https://www.devpanel.com/) or email: contact@devpanel.com.
