

Table of Contents

INTRODUCTION	3
AN EFFECTIVE CONTENT MIGRATION STRATEGY	3
STEP 1: EXTRACT AND AUDIT (THE CONTENT INVENTORY).....	3
STEP 2: TRANSFORM (DATA MAPPING).....	3
STEP 3: LOAD (LEVERAGING DRUPAL CORE)	4
SINGLE CODEBASE VS. MULTISITE STRATEGY	4
DRUPAL CORE CONFIGURATION MANAGEMENT.....	5
SOFTWARE DEVELOPMENT WORKFLOW & CONFIGURATION MANAGEMENT	5
SUPPORTING DEVELOPMENT ENVIRONMENTS.....	6
CONFIGURATION MANAGEMENT AND AUTOMATION	6
CACHING ARCHITECTURE & PERFORMANCE	6
DRUPAL INTERNAL CACHING	7
EXTERNAL APPLICATION CACHING (VARNISH AND MEMCACHE).....	7
CONTENT GOVERNANCE AND ACCESS CONTROL.....	7
CHOOSING THE RIGHT STRATEGY	8
SECURITY	8
PLATFORM MAINTENANCE AND PATCHING.....	8
STRATEGIC MODULE-BASED DEFENSE	8

Introduction

Website redesigns are complex, cross-departmental initiatives, but they represent one of the most impactful digital investments an organization can make. Historically, research shows that a significant majority—up to 81%—of these projects are driven by the urgent need to address poor performance metrics, such as low conversion rates or high bounce rates.¹ A successful, well-executed transition can dramatically boost user engagement and deliver tangible Return on Investment (ROI).

This comprehensive guide supports large organizations—including government entities or businesses with multiple departments—through the process of a website redesign and content migration to Drupal. The essential, enterprise-level best practices outlined herein focus on important aspects involved in a website redesign into Drupal with the goal of achieving a seamless transition and improving the overall digital experience and website performance. Success hinges on establishing several critical pillars that move beyond simple content migration. We will explore:

- Aligning Content to the New Information Architecture
- Implementing a solid CI/CD methodology and selecting the right development tools for continuous delivery.
- Defining a secure and efficient content workflow using key Drupal modules for moderation and governance.
- Optimizing the platform for high performance across all devices while ensuring security.
- Adopting these standards will ensure a successful transition from legacy systems to a modern, robust, and scalable Drupal-based content management platform.

An Effective Content Migration Strategy

Step 1: Extract and Audit (The Content Inventory)

To properly inform the new site architecture, development must be contingent upon the completion of an exhaustive content audit and migration strategy.

Every single piece of existing content should be accounted for within a comprehensive inventory, typically managed via a detailed spreadsheet. This inventory provides the necessary information to differentiate content that can be handled through an automated migration process versus content that will require manual creation due to new information architecture or component-based designs. These decisions, driven by the sheer volume of content, will dictate the scope and timeline of the migration and website redesign.

Step 2: Transform (Data Mapping)

Once content is audited and categorized, the focus shifts to the transformation phase. This process ensures all extracted content is highly structured and optimized for ingestion into the Drupal CMS. Content must be exported from the source system into a consistent format, typically JSON, CSV, or direct database tables. The critical final step is detailed data mapping, which aligns all exported content fields directly to the new Drupal content types, taxonomies, and fields.

ROT analysis

Identifies Redundant, Obsolete, Trivial content. It is important to identify the content that will be:

- **Removed** (archived or deleted).
- **Obsolete** (migrated and improved, often involving a manual process).
- **Trivial** (migrated as-is, typically through automation).

Step 3: Load (Leveraging Drupal Core)

The final phase involves loading the structured data into the Drupal CMS. Drupal's pre-load functions are specifically leveraged to perform any final necessary data cleansing or validation. This automated and pre-validated process significantly simplifies the final cutover, ensuring the seamless migration of the most current content at the time of go-live.

Migrate Custom & Contrib Modules

Drupal's core Migrate, Migrate Tools, and Migrate Devel modules provide the essential framework and tooling required to write, execute, and debug all custom migration routines.

Single Codebase vs. Multisite Strategy

A foundational architectural decision for multi-departmental organizations is whether to use a single, shared codebase to run multiple independent sites (multisite) or to maintain separate, independent Drupal installations for each site.

Approach	Advantages	Disadvantages
Single Codebase	Easier maintenance, consistent updates, shared themes and modules. Separate databases.	A single bug affects all sites; less isolation .
Multisite Installation	Flexibility for diverse departments; separate databases, allowing independent configurations.	More complex maintenance, higher hosting overhead .

When to Choose Which Approach:

- **Choose Multisite (Single Codebase) if:** Your sites share 70% or more of their functionality, content types, and branding (e.g., university departmental sites, government agency sub-sites). The primary driver is centralized maintenance and shared governance.
- **Choose Separate Installations (Single Site per Entity) if:** Your sites have vastly different features, use completely different sets of contributed modules, or require independent deployment schedules (e.g., highly specialized marketing campaign sites or separate business units with distinct security requirements).

Ultimately, while a single codebase offers substantial maintenance advantages, it requires a commitment to centralized governance and disciplined development.

Drupal Core Configuration Management

Important Drupal configurations—including content types, taxonomies, fields, and views—reside within the database and must be treated with the same rigor as the codebase. To ensure consistency and environment synchronization, this configuration data is managed via version control systems (Git). Developers leverage [Drush](#) commands to export the entire site configuration into YAML files, establishing a configuration-as-code workflow that enables historical tracking and simplified rollbacks of configuration changes.

Helpful Tip

Developers use the [Config Split](#) module to define condition configurations based upon the environment. This allows for the devel module to be implemented in the development environment, but not on the live site.

Enterprise best practice strongly recommends that all site configurations—including views, content types, and taxonomy terms—follow a strict **Configuration-as-Code** lifecycle, originating in lower environments and being deployed as version-controlled YAML files.

Software Development Workflow & Configuration Management

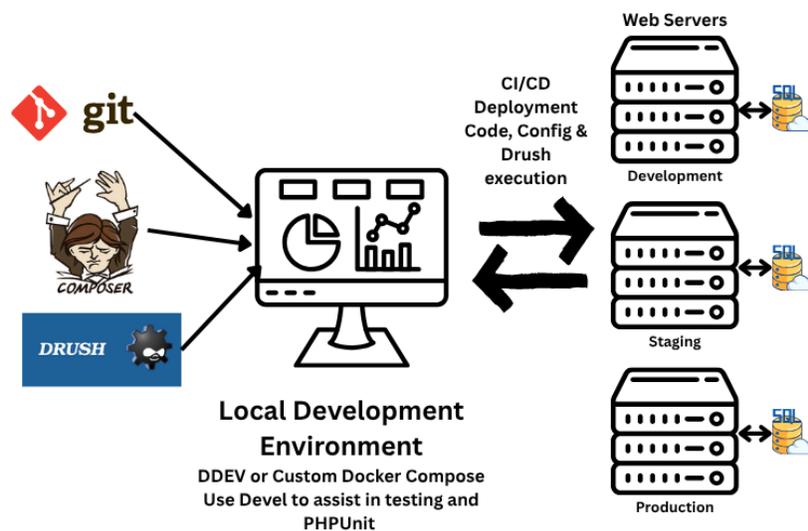


Figure 1: Drupal Development Workflow

Successful software development and deployments rely on a standardized, multi-tiered environment strategy and automation to ensure infrastructure parity, code quality, and reliable deployments.

Supporting Development Environments

Local Development Environment: The Local Development Environment is a production-mirrored workspace where all code and configuration changes are developed. Leveraging development tools such as [DDEV](#) or custom Docker/Docker Compose setups allows developers to precisely match the operating system, Linux distribution, database flavor, and version used in the production environment. This alignment minimizes environment-specific bugs and configuration differences. Developers run local [PHPUnit](#) tests, ensuring new code meets unit and functional requirements before deploying to integrated development environment. All code and infrastructure configuration changes are captured using a [feature branch](#) for review and validation.

Dependency Management

Utilize **Composer** as the required PHP dependency manager. It ensures that the exact versions of Drupal core, contributed modules, and third-party libraries are consistent across all environments.

Integrated Development Environments: When multiple developers contribute code and configuration updates to a Drupal project, a separate, development environment is essential. This environment serves as a crucial integration point where merged code is subjected to comprehensive integration testing and the execution of the automated regression test suite. This suite includes end-to-end scenarios managed by tools like Cypress, Selenium or Behat. This step is designed to catch conflicts, bugs, or broken functionality before the changes are promoted to staging or production environments.

Dedicated Staging Environments: A staging environment is critical for final quality assurance and risk mitigation before deployment to production. This environment must closely mirror the production setup and is specifically used for:

- **User Acceptance Testing (UAT):** Comprehensive validation by key stakeholders.
- **Performance Testing:** Validating system stability and speed under expected load profiles.
- **Client Sign-off:** Securing final business approval prior to the live deployment.

Configuration Management and Automation

Once development is complete in the local environment, all database configurations are exported as code using the Drupal-native drush commands. This process ensures that configuration changes (like new fields or content types) are deployed reliably alongside code changes.

Through the use of git feature branches, every code push is associated with a particular piece of functionality, security upgrade, or bug fix, ensuring traceability. Automation tools such as Jenkins, GitHub Actions, or CircleCI should be leveraged to manage the build, test, and deployment processes.

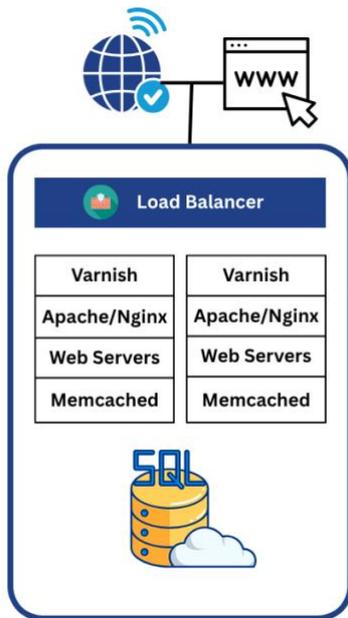
Caching Architecture & Performance

Caching is arguably one of the most important components of the production infrastructure, essential for maintaining site availability and optimizing resource consumption. Correct caching configuration is paramount for enterprise-level websites. An effective Drupal performance strategy relies on a multi-layered approach, distributing the caching load across the application stack to ensure optimal response times for all users.

Drupal Internal Caching

Drupal core includes a sophisticated caching layer that has matured significantly. By default, Drupal stores cache data in the database, but clears it dynamically based on content updates or time-based expiration. For performance at scale, developers should configure Drupal to use external, high-speed mechanisms in addition to Drupal's internal caching system.

External Application Caching (Varnish and Memcache)



Varnish and Memcache operate at different layers of the application stack and complement each other perfectly to handle both anonymous and authenticated traffic:

- **Varnish (Reverse HTTP Proxy):** Varnish sits directly in front of the web servers, caching entire page responses for anonymous users and delivering them in milliseconds without ever touching Drupal or the backend. For anonymous traffic—which often comprises the majority of enterprise website visitors—Varnish reduces server load drastically.
- **Memcache (In-Memory Object Caching):** Memcache provides high-speed, in-memory object and query caching that benefits all users, but is particularly critical for authenticated users who bypass Varnish. By storing database queries, loaded entities, rendered views, and permission checks in memory instead of requiring disk-based database lookups, Memcache reduces authenticated page load times from seconds to milliseconds.

The combination ensures optimal performance: anonymous users get instant page delivery from Varnish, while authenticated users benefit from Memcache-accelerated database operations.

Beyond Varnish and Memcache, organizations may want to implement a Content Delivery Network (CDN) as the outermost caching and security layer to reduce the load on the web servers. CDNs like Cloudflare, AWS CloudFront, or Akamai protect and improve your Drupal infrastructure performance. CDNs can provide essential security services through integrated Web Application Firewalls (WAF). These systems protect against sophisticated attacks including Distributed Denial of Service (DDoS) attacks, SQL injection, cross-site scripting (XSS), and credential stuffing. According to industry data, CDN-powered websites are protected from comment spam, email harvesting, SQL injection, cross-site scripting, and DDoS attacks, while improving site performance by 30% on average and reducing bandwidth consumption by 60% [2](#).

Content Governance and Access Control

Effective content governance in Drupal requires a layered strategy that controls the lifecycle of content and who can access or edit specific parts of the site. This combined approach ensures editorial accountability.



Workflows & Content Moderation (Core Drupal functionality) combined allows administrators to define custom editorial states (e.g., Draft, Review, Published) and assign role-based transitions. This capability enforces compliance and accountability by ensuring content must follow a strictly defined, auditable approval path.



Workbench & Workbench Access modules. The Workbench module enhances the editorial experience by providing a central dashboard overview of all content and its current state. Workbench Access adds a hierarchy-based permission layer. This feature allows administrators to restrict editorial user access to content based on specific site sections, taxonomy terms, or organizational groups.



The **Groups Module** offers a powerful, alternative method for content segmentation and access control. Instead of relying on a site hierarchy, it enables the creation of isolated "groups" where all associated users, content, and permissions are strictly managed within that context. This is ideal for managing distinct organizational entities within a single Drupal installation.

Choosing the Right Strategy

The selection between access control modules depends on complexity:

- **Workbench Access** is simpler and hierarchy-based, suited for straightforward editorial workflows where restrictions follow the site structure.
- **The Groups Module** is more robust, using explicit user and content groupings for complex departmental separations and integrated, siloed site management.

Security

Platform security is non-negotiable for enterprise level websites. A strong security posture is achieved through constant vigilance, proactive maintenance, and the strategic deployment of both application-level and infrastructure-level defenses.

Platform Maintenance and Patching

Proactive maintenance is important to reducing malicious attacks to your website. Organizations must subscribe to the Drupal Security Team's advisories (typically released on Wednesdays) and immediately apply all security patches to Drupal core and contributed modules to minimize the vulnerability window and address security vulnerabilities.

Strategic Module-Based Defense

Drupal modules can significantly enhance the platform's native security capabilities by implementing defense-in-depth measures against common web threats.

- **Security Kit (SecKit)** implements critical security headers and settings to reduce and eliminate exploitative risks associated with cross-site scripting, cross-site request forgery and clickjacking.
- **Password Policy & Login Security.** Password Policy enforces strict password requirements, while Login Security introduces IP-based login throttling and block to defend against brute-force and credential-based attacks.
- **CAPTCHA.** Should be deployed on all public forms to eliminate malicious spam bots and protect form integrity.

Security extends beyond the Drupal application. Key infrastructure hardening steps include:

- **Access Masking:** Proactively hiding the default user login page from anonymous users eliminates easy targets for bot-driven spam accounts and reduces server load from brute-force login attempts.
- **File System Protection:** Ensure executable permissions are strictly restricted on user-uploaded content directories (e.g., `/sites/default/files`) to prevent the execution of malicious scripts.
- **Database Isolation:** Use read-only database connections for web traffic wherever possible and ensure sensitive data is encrypted at rest.

Successfully building an enterprise Drupal platform requires a firm commitment to five key disciplines. First, ensuring data integrity through the ETL migration process. Second, establishing development governance using Configuration-as-Code and CI/CD. Third, choosing the right Multisite strategy for scalable architecture. Fourth, guaranteeing high performance and stability with multi-tiered caching. Finally, implementing comprehensive security hardening from application to infrastructure. By integrating these best practices, your organization transforms its website into a secure, high-performing digital asset that delivers sustained return on investment.