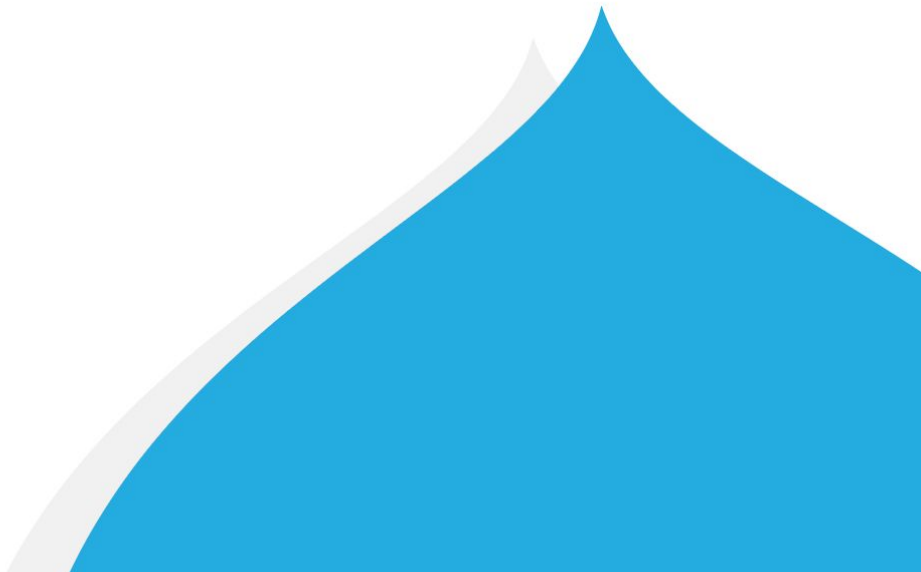
# 5 Mistakes to Avoid on Your Drupal Website

dev.acquia.com



# Table of Contents

# Introduction

Drupal is one of the most flexible content management systems in existence.

This ebook provides best practices in five crucial areas of building and maintaining an effective Drupal site.

Acquia's consulting staff has discerned patterns of mistakes as they have audited numerous Drupal sites. In this ebook, you'll benefit from lessons learned through those Acquia site audits.

- Architecture: Best practices for structuring content, how to build the display, and how to organize functionality.
- Security: How to avoid specific types of attacks, as well as Drupal best practices that help protect a site.
- Performance: Tools for performance analysis, common approaches to optimizing your site, and caching mistakes to avoid.
- Infrastructure: Best practices for your software stack, including Varnish and Memcached layers.
- Maintenance: How to set up best practices throughout the life of your site, including development, deployment, and maintenance phases.

This ebook assumes that you are familiar with Drupal site building and that you know PHP, Drupal-specific programming, such as hooks, and the MySQL database.

# Architecture: Content

Content is the essence of your website, the reason it exists. Determining the structure of content is the first step in creating website architecture.

## Best Practice

Plan your content structures, including fields and content types. Clear content architecture helps ensure good performance, a better user experience, and easier maintenance. (You may find overlap here with Display Architecture, because Views often depend on certain content types.)

**Mistake**: Too many content types.
**Result**: This will confuse content creators.
**Example**: Content types “news” and “article,” which are almost identical.
**Solution**: Reuse and standardize content types.

**Mistake**: New fields created for every content type.
**Result**: This is a waste of resources and drain on performance.
**Example**: Two different fields for school city and teacher city.
**Solution**: Reuse and standardize fields. Check your field report at example.com/admin/reports/fields.

**Mistake**: Content types with no nodes.
**Result**: An unneeded content type adds unnecessary complexity.
**Solution**: Reassess your needs as you build the site. Filter your content list to identify and delete unused content types.

# Architecture: Display

Drupal is a powerful tool for displaying content in different regions, formats, and displays. Display architecture includes the Views, Panels, and Context modules.

## Best Practice

The ease of changing the look and feel of your website is an indication of good display architecture.

- Plan your display architecture to render content only when needed.
- Optimize and reuse as much as possible.
- Always separate logic and presentation.
- Start with a solid base theme and learn it thoroughly.

**Mistake**: A new View for every list.
**Example**: Three separate Views for jobs in London, Paris, and Lisbon.
**Solution**: Analyze any new View you create to determine if you can reuse a View you already have, and use Contextual Filters to render lists based on specific parameters.

**Mistake**: PHP code or other logic in the database or in template (.tpl.php) files.
**Example**: PHP code that determines visibility of a scores block in a sports section.
**Solution**: Write all logic, including PHP, calls to web services, and SQL queries, in modules or theme preprocess functions if necessary.

## Recommended Tool

- **Theme Developer module**: With this module enabled, you can mouse over different areas of a webpage to see what template renders that section.

# Architecture: Site or Functionality

Site architecture includes how the site works, the number of modules, and how they interact.

## Best Practice

- Keep your site lean, using the minimal amount of code and fewest number of modules necessary.
- Use contrib modules whenever possible rather than writing custom code.
- Views is now in Drupal 8 Core and not a contrib module.
- Follow Drupal standards for custom code.
- Reevaluate your architecture periodically.

**Mistake**: Too many modules. More than 200 enabled modules indicates a need for analysis to be sure all are necessary.
**Example 1**: Original plan included multiple languages, but site ended up in English only. All multilingual modules and contributed modules are installed and enabled.
**Solution**: Reevaluate your site periodically, and disable, uninstall, and remove unused modules from the code base.

**Mistake**: Too many roles, which makes maintenance and security checking difficult.
**Example**: Original plan anticipated need for numerous roles, but most not used. Often roles attempt to match job titles too closely.
**Solution**: Evaluate roles and permissions on your site. Group into functional roles that can cascade and inherit permissions.

**Mistake**: Creating custom code when a contrib module already does the job well.
**Example**: Custom module to create forms on the fly that can be sent by email to site admins.
**Solution**: In this case, the well-tested Webform module provides this functionality, along with flexibility for site admins. Be sure no contrib module already does what you need.

**Mistake**: Hacked core or contrib modules. Behavior will be unpredictable. Updating is difficult.
**Solution**: If core or contrib doesn't do quite what you need, build a custom module using hooks to alter behavior.If you inherit a site, use Acquia Insight or the Hacked! module (see Recommended Tools).

**Mistake**: Custom code using the wrong hooks or using the Drupal API incorrectly.
**Example 1**: Using hook_init, which loads on every page, for something used only on the home page.
**Example 2**: Custom modules with hardcoded strings for nids, tids, and vids. When these change in the future, troubleshooting the cause of resulting problems is very difficult.
**Solution**: Plan carefully when using custom code. Find the right hooks and syntax using drupal.org's API documentation.

## Recommended Tools

- **Simplytest.me**: On this site, simply enter the name of a Drupal module. The site will spin up a Drupal instance for you to test the module for 30 minutes.
- **Hacked! module**: This module scans your Drupal installation, including contrib modules and themes, and determines if they have been changed. Used with the Diff module, result screens will tell you the exact lines that have changed. Absolutely not to be used on production sites.
- **Acquia Insight**: Our service does similar scans to the Hacked! module, but provides additional site configuration and security checks as well.

# Security

Good security practices protect your site from hacker attacks. Drupal has good security built in if used correctly.

## Best Practice

Once you begin to configure your site you might introduce new security issues. Plan configuration so that only trusted users have permissions that involve security risks.

- Keep core and contrib modules updated. You may not opt to do some module updates if the fixes or improvements have no direct effect on your site, however you should always apply security updates as soon as possible. Subscribe to security announcements on Drupal.org.
- Use strong passwords. Passwords are the most likely candidates for points of failure in your site security.
- Use the Password Policy module to devise a set of constraints before your users set their passwords.
- You can also set passwords to expire.

- Limit file uploads and what files are served. Limit the file types allowed and limit uploads to trusted users only. Check your permissions for specific content types and files types allowed in field uploads.
- Use the Security Review module and Acquia Insight. The Security Review module will analyze your site configuration and report methods for fixing errors. Use this module only on a staging or test site. Disable and remove the module on production sites. Our service, Acquia Insight, provides additional site configuration and security checks as well.

## Guard Against Attacks in Custom Code

Following are three attacks to guard against in custom code:

### Avoid SQL Injection

**Mistake**: Using SQL queries in code rather than using Drupal API.
**Example**: The code: *db_query("select * from table where id=$_GET['id']"); allows for the attack example.com/index.php?id=1 union select * from users;*
**Solution**: Use Drupal's database abstraction layer.

### Avoid XSS—Cross-site Scripting

**Mistake**: Displaying visitor parameters without checking them allows visitors to inject client-side scripts into pages viewed by other users.
**Example**: The code: *<?php echo "Your number is ". $_GET['id']; ?> allows the attack Index.php?id=<script>alert("UAAAT??");</script>*
**Solution**: Clean (sanitize or filter) any input from untrusted users before returning to the browser for rendering.

### Avoid CSRF—Cross-site Request Forgery

**Mistake**: URLs containing wildcards (%) that are not protected and form code entered directly into the site. HTTP Post from forms can allow a request to originate from anywhere, not just your site as you expect.
**Solution**: Use Drupal's Form API, which protects against these attacks by inserting a token in every form.If you render any sort of URL that should be protected, make sure that you either ask for a confirmation with the Form API or use token with the URL verify that the token is present and valid on the response handling.

# Performance

Performance is crucial for providing a great user experience. If the site is slow or balky, even great functionality won't keep the site visitor engaged.

## Best Practice:

The first action for improving performance is analyzing what the website is doing. With the answer to this question, optimize as much as possible, then implement caching.

**Analyze—Tools:**

- Devel for viewing database queries which run on each page.
- XHProf is generally the best tool to start with. Profiling is what helps you find the issues to begin with. Read about Using XHProf with Acquia Cloud.
- New Relic will analyze your site at a low level and report slow database queries, external queries, and specific pages. Read more information about New Relic, which is part of Acquia Network.
- Yottaa focuses on front-end performance. This service will tell you how fast your site loads in different locations around the world. More information about Yottaa, which is part of Acquia Network.
- Read more about tools for parsing a slow query log.

**Optimize—Common Problem Areas:**

- Using complex queries that take too much time and don't use an index.
- Calling functions too often.
- Keeping unused modules enabled on your site. Disable any unused modules.
- Misconfiguring cron. See more about configuring cron.
- Using the default views pager, which requires an additional COUNT query. Use Views Litepager, which provides pagers without the COUNT function.
- Using database logging (dblog). It is enabled by default in Drupal 7, and errors can fill up your database quickly. One

common solution is to use syslog instead, but this merely masks the problem by making the logs less accessible. A better solution is to fix all PHP notices and warnings to reduce logging overhead.

- Use the Fast 404 module to serve static 404s for image, icon, CSS, or other static files, rather than bootstrapping Drupal.
- Not aggregating CSS and JavaScript files. See how to turn CSS and JavaScript aggregation in Drupal.

**Caching—Common Mistakes**

- Having no cache strategy at all. Not taking the time to understand how content can be cached
- (per user, per group, per role, and so on) is the worst mistake.
- Clearing caches too often.
- Caching at too low a level, such as using views cache instead of Blocks or Panels pane cache.
- Using basic caching, such as block caching or panels pane caching.
- Using a caching strategy that is too complex for real needs of site.

## Recommended Tools

- Simple Tips to Improve Drupal Performance: No Coding Required for more details about performance tuning.
- When and how caching can save your Drupal site, and When and how caching can save your site part 2: authenticated users, Hernani Borges de Freitas.
- Datasheet for Acquia's Performance Audit.
- Practical performance tips in Acquia's Library: Improving website performance

# Infrastructure

Infrastructure covers the stack your website lives on, including the server, the database, and any software layers, such as Varnish or Memcached, which ensure your visitors have a snappy experience. Planning the infrastructure from the start and developing on the same environment can greatly reduce variables and risk at launch time. Having a reliable multiple environment configuration and a solid disaster recovery plan shouldn't be left to last-minute decisions. When it is, mistakes start arising. Here's a few tips to avoid the most common errors.

## Best Practice:

- Size your stack correctly, not too large, not too small. This can ensure you're economically prepared for anything.
- Bottlenecks can arise from the hardware or from processes hogging memory. Check logs for errors and prepare for growth and spikes. Your stack is only as fast as the slowest component. Focus your efforts there; you'll probably find low hanging fruit.
- In terms of security, it's crucial to configure to protect from internal attacks as well as external attacks

### Size Your Stack Properly

**Mistake**: Server's hardware capacity is sufficient but misconfigured.
**Example**: Database server set large enough, with 48GB of memory, but InnoDb buffer pool set for only 1GB.
**Solution**: Take into account all aspects of stack configuration. Use tools such as mysqltuner.pl (see Recommended Tools) to analyze your database.

### Let Varnish Take the Hit

**Mistake**: Misconfiguration causes traffic to bypass Varnish and hit the server.
**Solution**: Check response headers to ensure that pages you expect to be cached are indeed cached. Ensure that modules aren't setting session variables unnecessarily.

## Avoid Exposure to Vulnerabilities

**Mistake**: Remote connections to the database, Memcached, or Solr are allowed.
**Example**: Assuming an external firewall will provide adequate protection, the port that runs Memcached is not protected via IP tables.
**Solution**: As many as 50 percent to 70 percent of attacks can be internal. Forbid remote connections to the database, Memcached or Solr, and maintain this configuration through any infrastructure changes.

# Recommended Tools

- mysqltuner.pl or MYSQLTuner: This is a Perl script that you can download from Github. It will present current configuration variables and status data for your MySQL installation, along with some basic performance suggestions.
- Infrastructure workshops: are useful for operations teams new to Drupal or LAMP requirements. Especially
- if you're building out your own infrastructure, instead of going with a managed solution like our own hosting.
- Drupal Multi-site Infrastructure: has more specifics about the configuration of multi-sites.

# Maintenance

The life cycle of a website begins from initial plans and extends to the end of the site. The site exists in three different phases: development, deployment, and maintenance. After the site is launched, your website lifecycle practices become critical to the success of changing and maintaining your site.

## Best Practice:

- Keep your code under version control.
- Maintain separate environments for the different stages of the site, and keep them up to date.
- Restrict access to the production site for all but the most trusted users.
- Review all logs periodically, including Apache, Drupal, and MySQL.
- Review and assess your architecture periodically, and plan for the future.

### Maintain Reliable Collaboration Through Version Control

**Mistake**: Not using a version control system (VCS).
**Example**: Copying code occasionally to backup folders as a method of version control.
**Solution**: Use a version control system. Git is the most popular among Drupal developers, but any system is fine if used. Be sure you leave meaningful commit messages so that colleagues can understand the changes you have made.

### Keep Organized

**Mistake**: Keeping extraneous files in the VCS repository.
**Example**: Image asset files, holiday pictures, or database dumps pushed to the repository.
**Solution**: Keep the VCS as clean and small as possible.

### Deploy with Version Control

**Mistake**: Uploading files to production through FTP.
**Solution**: Deployment must come directly from the VCS repository.

## Stay Secure During Deployment

**Mistake**: Production environment not properly secured.
**Example**: Any developer can take a snapshot of production to install on their laptop.
**Solution**: Allow developers to take snapshots and have other access to development and staging environments, but allow access to the production environment only for the most trusted users.

## Test on Environments as Similar to Production as Possible

**Mistake**: Development and staging environments out-of-date or missing functionality.
**Example**: Testing only in production because the other environments have old data or no connection to an external service.
**Solution**: Maintain testing environments as similar to production as possible. With these, you can easily copy from production and execute tests to ensure that changes will work when moved to production.

## Keep Your Site Up to Date

**Mistake**: Sites using out-of-date code for core and contrib modules.
**Solution**: Keep all modules and core as current as possible.