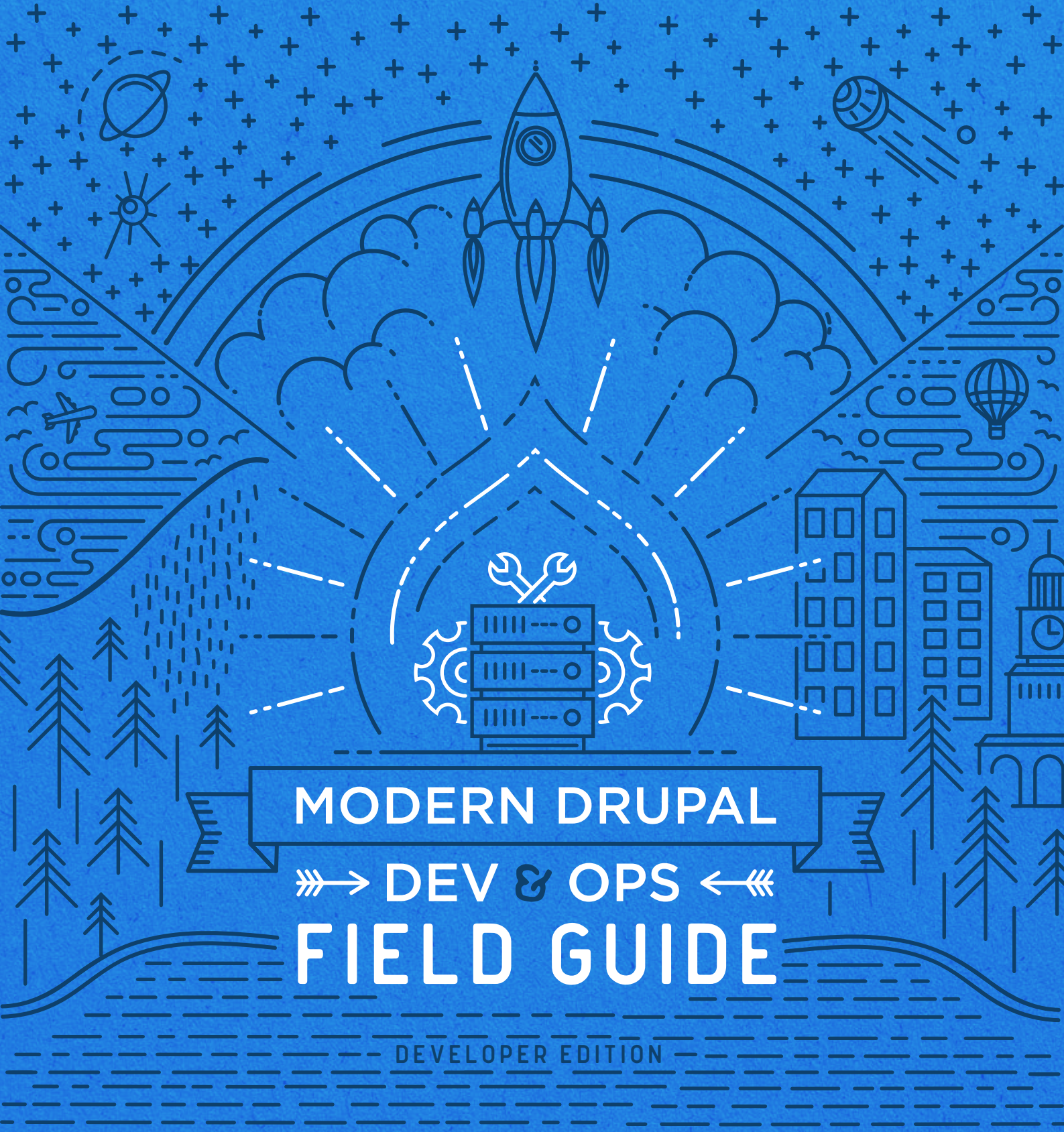


MODERN DRUPAL

DEV & OPS

FIELD GUIDE

DEVELOPER EDITION

platform.sh

TABLE OF CONTENTS

Drupal in 2017 is not Drupal in 2001.

Implementing modern DevOps practices can bring great impact on project lead time, cost, maintenance costs, and failure rates:

- Up to 60% overall cost reduction
- 3x higher developer productivity
- 14x faster feature and UAT sign-off

Integrating DevOps can seem complex, but with the right techniques, Drupal teams can be more productive than ever. We have gone through the journey ourselves and have developed guidelines for our own internal Drupal projects, as well as received feedback from our numerous clients and agency partners.

To get you started on DevOps best practices, here are six steps that you should follow.

1

Embrace Change

The development model of Drupal has changed and your teams need to change with it.

EMBRACE CHANGE

Major Drupal versions used to be far apart with no migration path between them, but this has changed for the better with Drupal 8 and beyond.

Here are five ways to prepare and adapt to change with ease:

Use Drupal 8. Don't start new projects on Drupal 7 unless absolutely necessary. The more you wait the worse your migration cost will be.

Use Composer. A Composer-based workflow is better suited to a continuous delivery infrastructure and makes leveraging a wider variety of existing code far easier.

Be build-oriented. Use dependency management and configuration management exclusively in your development process.

- Make every Drupal project's Git repository as lean as possible by only including the project-specific code and configuration.
- Never commit Drupal core to your repo. Use Composer.
- Don't commit contrib modules. Use Composer.

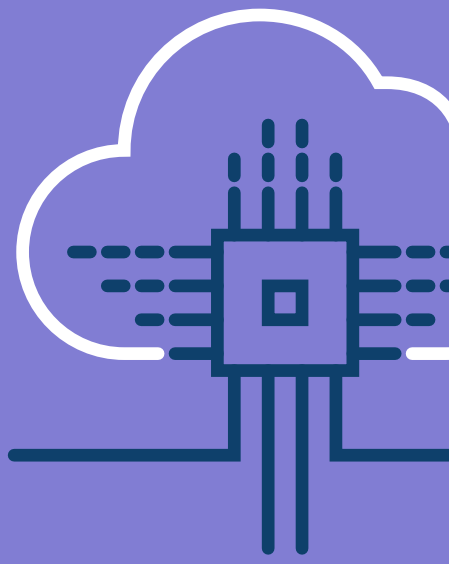
- Never hack core. Never hack contrib either. Use clean patches with the cweagans/composer-patches plugin. Do not forget to contribute upstream. Use Composer.
- Use NPM or Yarn for your JavaScript dependencies, use webpack to pull in CSS and JS dependencies.
- Use Gulp, Grunt, or another task runner to compile static assets. Do not commit compiled assets to the repository.
- Always commit lock files to your repository.

Use Config Management to export configuration to Git. Use Features for Drupal 7, or use the Configuration Management system for Drupal 8.

Promote internal reuse:

- If you repeat the same type of projects create an internal distribution with an installation profile as a starting point.
- Do not commit internal modules directly to your distribution repository. Use a private Composer repository instead (either your own with Satis or via a service like Private Packagist).

2



Leverage modern infrastructure elements

LAMP had its day. A more modern infrastructure can save you a lot of time in the development cycle and a lot of heartache when you need to scale.

LEVERAGE MODERN INFRASTRUCTURE ELEMENTS

Use HTTP/2.0. It's faster. Use SSL everywhere, and redirect HTTP to HTTPS.

Use the latest version of PHP wherever possible. Never, ever, use the unmaintained PHP versions below 5.6.

Move caching out of your SQL database to a dedicated caching service such as Redis.

Do not put application logic in the HTTP caching layer (e.g. custom VCLs). Instead, make sure you are correctly setting cache headers in the application.

Do not rely on MySQL search. Use Apache Solr or Elasticsearch if you need search functionality.

Move any background tasks to a dedicated queue server such as RabbitMQ, with a separate queue worker. That will perform faster and more scalably than Drupal's default cron-queue behavior.

Do not rely on insecure services or unencrypted services such as FTP, use a VPN if you need access to internal resources. Only allow SSH access to your servers.

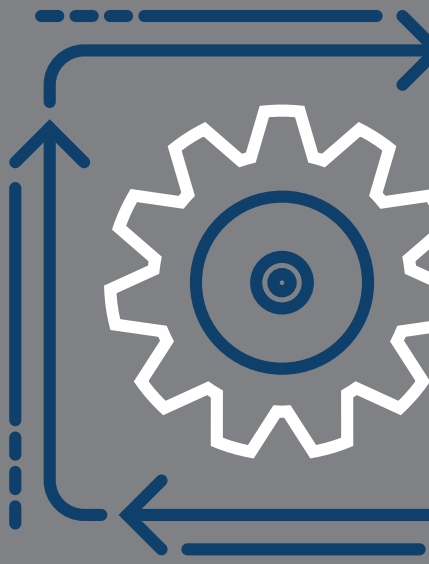
LEVERAGE MODERN INFRASTRUCTURE ELEMENTS

Embrace openness to other tools. Drupal doesn't have to solve all of your problems. Resolve hard problems through infrastructure capabilities. If you need complex aggregations and visualization use Elasticsearch and Kibana. If you need to implement some simple machine learning use Python. If you need a Product Information Manager, use Akeneo. Node.js is cool. Go is fast and robust.

Tie multiple applications together in a single infrastructure for deployment.

Make sure you have backups that contain everything, not just one application or service out of sync with another.

3



Automate all your testing environments

To make the business side happy, developer teams must be agile enough to develop, test and ship quality code continuously. To make that happen you must eliminate the old traditional model of unary prod, staging, dev environments.

Align your development and testing clusters 100% with production.

- Same versions of all components (code as well as infrastructure).
- Do not test on small datasets; test at production scale.

Create ephemeral testing environments for every Git branch or pull request; test features in isolation.

Test deployment and migration processes as well as the features themselves.

Write tests and run them on every Git push.

Make sure all servers are immutable (read-only). Do not allow changes on production other than through a version control system.

Manage developer credentials centrally. Make sure you know who has access to which environment. Restrict access to the production branch.

Use protected Git branches and enforce code review and sign-off before release. The production branch should be stable and deployable to production at any moment.

Embrace continuous performance regression testing on every new feature.

4

Automate deployments and updates

Wouldn’t it be great if developers can spend more time coding new features instead of worrying whether each feature will ship successfully? If your tests were done right in the first place, then you should be able to deploy automatically with no fear.

AUTOMATE DEPLOYMENTS AND UPDATES

Deployments should be repeatable and fully automated. No-one should have access to production servers. Avoid having root access.

Use a configuration management tool for your infrastructure. Make sure OS configuration and firewall rules are managed centrally and have an audit trail.

Deployments should be immutable (you should always be able to destroy an existing cluster and create a new one that is precisely the same). Always commit lock files of versions. Know precisely what version of what infrastructure element you are running (MySQL, nginx etc).

Automate security updates for your infrastructure (operating system, database, web server).

Make sure anything that can be run by the web server is read-only (use a read-only filesystem). Remove anything from the web root that can be removed. Use an unprivileged user for any service that is running. Make sure that writable mounts (file uploads, caches) are not executable on the command line and won't be interpreted by the web server.

AUTOMATE DEPLOYMENTS AND UPDATES

Implement log-rotation and temp files garbage collection to make sure you don't run out of disk space.

Consider implementing services redundancy so security updates can be applied with no downtime.

Make sure your backups can be restored. Prefer cluster-wide, automated, consistent backups (including MySQL, writable mounts, Solr...).

5

Implement observability

Make sure every service is healthy by implementing proactive monitoring.

IMPLEMENT OBSERVABILITY

Monitor CPU, Memory, Disk usage, and global latency (as seen by the user). Configure alerting at levels less than critical.

Integrate alerting with chatops (through Zapier, Hipchat, Slack) implement time-based rules for escalation (like PagerDuty).

Centralize logging, and make sure log access is protected.

Instrument production for app performance monitoring, use tools like Blackfire.io or NewRelic.

Keep a global audit log for changes to the application, the infrastructure, and user access rules.

Monitor cloud resources cost by implementing an optimization strategy for instance reservations.

6

Implement high availability and scaling

Sometimes things will go wrong, no matter how good your code or deployment plan is. Prepare for things to break and make sure plan B is ready. You don’t need your own Chaos Monkey, just the right level of active redundancy.

IMPLEMENT HIGH AVAILABILITY AND SCALING

Implement high availability and load balancing by deploying every service (including MySQL with Galera) to a cluster. Prefer active-active replication when you can.

Implement automated failover. When a cluster member fails or misbehaves, have an automated procedure to kill it and create a new one.

Consider triple redundancy as the minimum per server to allow for zero-downtime scaling and security updates (so you can take one element offline and update it while still having redundancy).

Deploy to a cloud provider that offers instant creation of new machines, automate the creation of machines and their cluster configuration. Consider scaling first vertically (making each cluster member bigger) and only secondly horizontally (adding more members to each service cluster).

Serve all of your traffic through a CDN, consider using a multi-tiered approach (cheap CDN for static assets, full-featured CDN with global instant purging and tag based purging for the Drupal content).

Sounds complicated?
Not if you do one thing.

What if you could follow all six steps at once? Use Platform.sh. It gives you everything, out of the box, ready to use, with zero upfront investment.

Companies using Platform.sh have seen their developer teams achieve better quality results at a faster pace:

Task	What clients typically say
New system setup time	Hours to <30mins
DevOps & ticket reduction	3x less
Deployment time reduction	15x faster
Deployment frequency	Up to 20x faster

You don’t need to care about any of the complicated details. And you can start slow. We can run a traditional Drupal with everything committed to the same repo and because all the tooling is integrated, you can move, step by step, to a modern workflow within days.

platform.sh