MODERNISING DRUPAL MULTI-SITE IMPLEMENTATIONS

Drupal multi-site is easily re-architected to run each site in its own containerised environment. It’s better and it costs less!

Drupal multi-site is no longer the most cost effective way to implement and manage large numbers of similar sites. In fact there are many disadvantages to the multi-site approach, some of which disappear when using containers. But when Platform.sh is the PaaS running those containers, the rest of those problems go away and at the same time you gain a whole load of new advantages.

“We have experience doing this for 100+ site implementations onto Platform.sh in our own private cloud, and the value obtained is huge.”
- Mike Carter, Technical Director, Ixis

Drupal multi-site was a clever idea about 15 years ago, and it allowed many sites to run off a single code base. The problem it solved at the time was finding a way to easily manage many similar sites without running many separate expensive infrastructures.

So let’s take a closer look at multi-site, containers, and the migration of one to the other.

MULTI-SITE DISADVANTAGES

There are many cost, performance and management constraints to a multi-site implementation, as follows:

1. **Deploying multi-site is cumbersome**
   The fact that all sites in multi-site deployments share one physical codebase means that all sites get deployed - at the code level - simultaneously. A deployment, however, is never just code, it is also the scripts that run after the code, most notably feature reverting, cache clearing, and database schema updates. It is typical to need 2-3 expensive and sometimes long-running Drush commands per deploy. These have to be executed on every site, and sequentially, otherwise you inadvertently DoS your own infrastructure.

2. **Deploying multi-site is dangerous**
   Botching a code deployment to a single site is bad; if you have a code condition that breaks your site in some way, and it makes it through your testing to deployment, you will have the problem of a broken or offline site. With multi-site, the same code problem would take all of your sites offline, not just one. Multi-site multiplies the risk of any single deployment by the number of sites that you’re running.
MULTI-SITE DISADVANTAGES

So let’s take a closer look at multi-site, containers, and the migration of one to the other. There are whole load of new advantages.

You’re locked in, all the sites are subject to the same capacity and release level limitations imposed by the common server architecture. If one site experiences a spike in traffic then all the other sites have less resources to draw upon and might be negatively affected. This ‘noisy neighbour’ syndrome frequently happens and sometimes for less predictable reasons such as a DDOS attack or site-specific caching, code or theme-related issues.

3. Deploying multi-site is not agile

Deployment is not just a technical problem, it’s a stakeholder problem. Every site owner and editor needs to know when you’re deploying, and you may possibly have to coordinate downtime windows with them. This is an organisational and logistical problem that is multiplied beyond manageable control when running multi-site. Multi-site forces you to coordinate logistically across potentially unrelated teams to agree on deployment windows, and customer support to deal with change management or problem resolution that results from deployment. Multi-site is the opposite of agile.

4. Shared hardware means noisy neighbours

You’re locked in, all the sites are subject to the same capacity and release level limitations imposed by the common server architecture. If one site experiences a spike in traffic then all the other sites have less resources to draw upon and might be negatively affected. This ‘noisy neighbour’ syndrome frequently happens and sometimes for less predictable reasons such as a DDOS attack or site-specific caching, code or theme-related issues.

5. Requirements dependencies require all sites move in lockstep

Multi-site takes away the choice to improve site usability with various PHP versions and extensions at the per-site level. Multi-site prevents you from upgrading any component or any site without affecting all sites equally.

SPLITTING MULTI-SITE INTO SINGLE-SITE ENvironments

Separating multi-sites into individual environments is easy to do, and it brings many advantages. The basic requirements to make this happen are Git, drush make and Platform.sh.

Git on its own is enough to solve the core problem that Drupal Multi-site previously attempted to solve: sharing a common Drupal codebase across many sites. With modern deployment tools, such as Platform.sh, it is sufficient to maintain Drupal code, such as a Drupal distribution or install profile, in a Git repository that is accessible over the internet. Services such as Drupal.org, Github, and Bitbucket fulfill this need perfectly. Deploying a Git branch or tag to many sites is not only easy, it guarantees that the sites will be running the same code, which is everything that Multi-site ever offered.

With Git + Platform.sh, there are even more possibilities, as developers can choose a layered approach to code management, differentiating between a base distribution, sets of common modules or features, and site-specific components such as integrations, customisations, or themes, and manage each one in an organised and repeatable fashion.

For example, changes that are meant to affect all sites equally can be implemented at the distribution level, whereas changes that only affect one feature can be implemented in a separate Git repository at the feature or common module level, and changes that affect only specific sites can be implemented directly in the Platform.sh Git repository for those specific sites.

“"The cost and complexity some of our multi-site customers are experiencing has become overwhelming”"
There are many cost, performance and management constraints to a multi-site implementation, as well as a whole load of new advantages. Those containers, the rest of those problems go away and at the same time you gain a whole new dimension to your development environment. Some of which disappear when using containers. But when Platform.sh is the PaaS running your applications, you can have it both ways.

Drupal multi-site was a clever idea about 15 years ago, and it allowed many sites to run off one set of infrastructure. Drush commands per deploy. These have to be implemented directly in the Platform.sh Git repository for those specific sites. However, this approach to code management, differentiating between a base distribution, sets of common modules or features, and site-specific components such as integrations, customisations, or themes, is simply not viable. With Git + Platform.sh, there are even more possibilities, as developers can choose a layered approach to code management, whereas changes that only affect one feature can be implemented in a separate repository.

The cost and complexity some of our multi-site customers are experiencing is significant, and they are questioning if it’s worth it. The fact that all sites in multi-site deployments share one physical codebase as Platform.sh, it is sufficient to maintain Drupal code, such as a Drupal distribution or install profile, as Platform.sh, it is sufficient to maintain Drupal code, such as a Drupal distribution or install profile, solves: sharing a common Drupal codebase across many sites. With modern deployment tools, such as Drupal Make and Composer, developers can create new sites, move code, and manage changes without opening tickets.

Botching a code deployment to a single site is a stakeholder problem. Every site owner requires the number of sites that you’re running multiplies the risk of any single deployment by the same capacity and release level limitations imposed by the common server architecture. Deploying multi-site is not agile. It’s a stakeholder problem. Every site owner needs to coordinate downtime windows with them. Deployments, feature changes and general operational management has become more complex, they are less frequent, and they take more time. Multi-site forces you to coordinate logistically across potentially unrelated teams to agree on targeted changes to the right component or any site without affecting all other sites.

IS MY SITE READY FOR THE PLATFORM.SH APPROACH?

If you’re running multi-site then the answer is: yes. Moving to a distribution-based setup should be straightforward and the preceding steps will have simplified and strengthened your architecture. Once you’ve made the initial migration to containers, you are able to:

1. Use multiple Git repositories to manage the separate interdependent layers of the site build. Eg, you may use Drupal.org Git repositories to manage Drupal core, a base distribution, modules and themes. On top of that, a private Git repository might be used to define the build process (eg. Drush Make, Composer), and a number of other Git repositories might be used to manage specific custom modules or themes that are to be used in building the sites. Finally, Platform.sh provides a Git repository for every site, and that can be used for the code that is specific only to an individual site.

2. Deploy many sites in parallel, or sequentially, but in any case all independently of each other. Assuming no problems with the distribution itself, there is no longer any risk of changes to one site breaking another. You don’t need to coordinate between stakeholders on separate sites to plan deployment. Now your deployments are agile.

3. Customise individual sites easily with targeted changes to the right component at the right layer in the right Git repository.

The entire planning, architecture design and migration phase for our first large project (100+ sites) took less than 3 months. Platform.sh allowed us to hugely improve many aspects of the development and deployment process.

"The entire planning, architecture design and migration phase for our first large project (100+ sites) took less than 3 months. Platform.sh allowed us to hugely improve many aspects of the development and deployment process."

Figure 1: How a containerised distribution works on Platform.sh. In this example the output is a single project with multiple environments; in practice the same model is being used to manage dozens or even hundreds of websites.

IS MY SITE READY FOR THE PLATFORM.SH APPROACH?

If you’re running multi-site then the answer is: yes. Moving to a distribution-based setup should be straightforward and the preceding steps will have simplified and strengthened your architecture.

Once you’ve made the initial migration to containers, you are able to:

1. Use multiple Git repositories to manage the separate interdependent layers of the site build. Eg, you may use Drupal.org Git repositories to manage Drupal core, a base distribution, modules and themes. On top of that, a private Git repository might be used to define the build process (eg. Drush Make, Composer), and a number of other Git repositories might be used to manage specific custom modules or themes that are to be used in building the sites. Finally, Platform.sh provides a Git repository for every site, and that can be used for the code that is specific only to an individual site.

2. Deploy many sites in parallel, or sequentially, but in any case all independently of each other. Assuming no problems with the distribution itself, there is no longer any risk of changes to one site breaking another. You don’t need to coordinate between stakeholders on separate sites to plan deployment. Now your deployments are agile.

3. Customise individual sites easily with targeted changes to the right component at the right layer in the right Git repository.

The entire planning, architecture design and migration phase for our first large project (100+ sites) took less than 3 months. Platform.sh allowed us to hugely improve many aspects of the development and deployment process.

"The entire planning, architecture design and migration phase for our first large project (100+ sites) took less than 3 months. Platform.sh allowed us to hugely improve many aspects of the development and deployment process."

Figure 1: How a containerised distribution works on Platform.sh. In this example the output is a single project with multiple environments; in practice the same model is being used to manage dozens or even hundreds of websites.
4. Mix up your technical requirements, running different backend services, components and release levels across different sites as required. One or more sites can try out the latest PHP version, or HipHop VM, without forcing all sites to do the same.

5. Not worry about ‘noisy neighbour’ syndrome, as performance issues are restricted to the container the site is running in. For example, a DDOS attack on one site can’t affect the rest of the estate because there are no shared services such as memory/compute resources.

6. Scale resources for individual sites to deal with unexpected peaks.

7. Leave the management of every aspect of the hosting stack to Platform.sh but still keep enough flexibility to make your own decisions about the infrastructure without opening tickets.

“Deployments, feature changes and general operational management has become so much easier.”

PLATFORM.SH SITE FACTORY: CREATING & MANAGING LARGE NUMBERS OF SITES

Using the Platform.sh site provisioning API, you can quickly create new sites based on a specific starting point that you define. This allows you to churn out sites, on demand, and manage them centrally from common upstream Git repositories. You also retain the ability to heavily customise each individual site and incorporate bespoke components, without losing the upstream connection.

This allows customers to:

1. Easily create new sites, make changes and maintain them as well as decommission old sites, all without any dependence on the hosting provider; no support ticket required!

2. Manage resource allocation for sites independently, eg. upsize, downsize, add storage etc.

3. Gain volume discounting on large numbers of sites

4. Provide each client their own dedicated infrastructure (ie. private cloud)

5. Gain geographical independence for private regions

Environments are completely self-contained and can be easily branched, which makes testing, evaluation, and on-boarding fast and safe.
Platform.sh has turned the previously onerous task of managing multiple sites into a highly streamlined production line.

CONTINUOUS DELIVERY (CD) AND CONTINUOUS INTEGRATION (CI) BENEFITS USING PLATFORM.SH MANAGED DISTRIBUTION FOR A MULTITUDE OF SIMILAR BUT DISTINCT SITES

Currently achieved through a combination of the Platform.sh CLI, and integration of other CI and testing tools through the API. Better bulk-management features are on the roadmap.

1. Easy maintenance of distributions that can be used as the basis for site deploys
2. Automated deployments based on the distribution
3. Retain ability to customise and write bespoke code per site
4. Full testing and QA workflows available
5. Granular access (per site) to specialist teams
6. Broad access to programme management
7. Ease of bulk administrative operations (user management, reporting, maintenance)

Platform.sh is a Git-driven PaaS that hosts multiple applications on multiple technology stacks, while removing DevOps from your continuous delivery workflow. Learn more at https://platform.sh