Effectively managing 1000 Drupal websites
About me

- Solutions Architect @ Acquia
- Experience from:
  - Brightcove.com
  - Nokia Meego
  - Johnson & Johnson Open Solutions
- Twitter: @jakubsuchy
Inside many organisations, the web is broken
Business requirements come from many teams, Resources are scarce, Team members come & go
You end up maintaining not one codebase but hundreds or thousands
This is the amount of time people typically spend on maintaining their site as opposed to creating new features. This is provided you do maintain your Drupal codebases :)
Now imagine you have 100 websites. You can easily imagine that without automation or some sane practices, you will end up having a full time person doing just site maintenance. That’s not good & expensive
Business requirements come from many teams, Resources are scarce, Team members come & go
You end up maintaining not one codebase but hundreds or thousands
This is the amount of time people typically spend on maintaining their site as opposed to creating new features. This is provided you do maintain your Drupal codebases :)
Now imagine you have 100 websites. You can easily imagine that without automation or some sane practices, you will end up having a full time person doing just site maintenance. That’s not good & expensive
Universities are unique
- they typically have either a centralized IT team that maintains and owns all sites - can influence, dictate
- decentralized - many different teams and different responsibilities, each owning their own sites
Imagine a 7 figure sum

Why does this matter? A company had 1000 websites running, each created by a different team, each maintained separately. After a series of security issues they got ready to write a $1m check to someone to “just” upgrade all the sites to newest versions of Drupal code.
Our goals

‣ Reduce site maintenance
‣ Increase efficiency
‣ Foster collaboration
Establish a governance model

- Single team as a gatekeeper
- The trick: Balance of power for site teams
- Only works for centralized teams

Let's take a look at a couple of approaches from both governance and code management perspective. Establish a formal governance over your efforts: Make a single team responsible for updates and developments, create a gatekeeper that will give power to other teams while maintaining control. The trick is to find out how to do this without giving site teams too much power. Of course this only works for centralized teams.
Why do you use Drupal?
Why do you use Drupal? For the community!
This is the best scenario especially for decentralized teams
Consider creating an internal open source community. Running your own open source community can foster collaboration, enable your teams to reduce maintenance overhead and increase speed of development
#9 Break Down Barriers
Create an internal community

- NBC Universal
- University of California, San Francisco
Join LSD - largescaledrupal.com
Take baby steps

- Organize a Camp! Hello!
- Schedule office hours
- Github all the things
- Let a leader emerge
Site (Yellow):
/sites/foo/modules/*
/sites/foo/themes/*
/sites/bar/modules/*
/sites/bar/themes/*

Platform (Brown):
/profiles/foo/modules/*
/profiles/foo/themes/*

Core (Blue):
/modules/*
/themes/*
/includes
... etc.

Site specific themes & modules
Platform theme & modules
Drupal core
Multi-Site

- Works great for single teams
- Easy to keep core & platform in sync
- Hard to test & QA across many sites
- Difficult for many teams
Independent sites

- Lets each site be tested, QA'd individually at their own pace
- Lets independent teams work on individual sites without breaking others
- Difficult to maintain
In this scenario, you create a unified Platform / Distribution that contains functionality common for most of your sites and then let your development teams only develop custom modules they need on top of the platform. This lets you keep a single codebase for majority of your code while you give your teams the flexibility to do their own thing under certain circumstances.

The danger? Give people too much power and they will fork the codebase unless you have very strong governance. If you run in decentralized environment then this approach will struggle unless you have a good internal open source community.
Mmm... Cake

- Segregate responsibility for site-specific development, platform development and QA
- Let dev teams use the tools they're used to (git, Github, Drupal)
How does it do it?

Dev Stage Prod

Platform/Site

Push to production (manual and automated)

Github Repos

Platform 10

Platform 11

Platform 12

Site A

Site Tag 4.3.6

Target Repo DevCloud 3

Target Tag newfeaturebranch

Site A Setup

Platform 10

Tag 1.4.3

Managed Cloud

Site Code Publishing Process

Production environments

Distribution Environment

Code Distribution Management

Acquia

#SolutionsArchitecture

Monday, December 2, 13
Who does it work for?

- **Site Developer**
  - Commits themes, modules to Site repo

- **Platform Developer**
  - Commits changes to repos managing Drupal core, platform features and modules

- **Platform Admin**
  - Manages available deployment environments, setup

- **QA Team**
  - Tests and pushes approved code to production, for 1 site or 100's

- **Production hosting**

---

**Acquia**

Monday, December 2, 13
Developers!

Developers
Commit to repos they've been granted access to in Github

Use Layer Cake to deploy their repos periodically to sandboxes

Developer's personal sandbox where they can test against replica of production stack
Admins!

**Platform Admin**
Sets holds the keys to creating new mappings

---

**Developer A**
Sandbox

**Developer C**
Sandbox

**Developer D**
Sandbox

**Production hosting**
Production HA environments

---

**Site A Setup**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Site Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform 10</td>
<td>4.3.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Target Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.3</td>
<td>newfeaturebranch</td>
</tr>
</tbody>
</table>
Quality Assurance!

QA Team
Tests and pushes approved code to production, for 1 site or 100's

Filter Sites
Keyword: text
Platform: 10b

Deploy Stage -> Prod

<table>
<thead>
<tr>
<th>Site</th>
<th>Platform Version</th>
<th>Site Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site A</td>
<td>10b</td>
<td>4.5</td>
</tr>
<tr>
<td>Site B</td>
<td>10b</td>
<td>3.1</td>
</tr>
<tr>
<td>Site C</td>
<td>10b</td>
<td>1.0</td>
</tr>
<tr>
<td>Site D</td>
<td>10b</td>
<td>7.2</td>
</tr>
<tr>
<td>Site E</td>
<td>10b</td>
<td>134.2</td>
</tr>
</tbody>
</table>

Deploy | Cancel
Site management gotchas

- Testing the upgrade path on all sites. Shoot for eventual consistency but beware of the dangers of falling out of sync.

- When using the Layer Cake approach beware Platform and Site compatibility issues.
How to start

‣ Do an inventory

‣ Start just by moving modules around.

‣ Standardize on top modules.

‣ Once modules are standardized, start capturing your common configurations into Features.

How many sites do we have?
How similar are they?
Are we looking at one platform or many?
Features. Love them. Hate them. Use them.
Overridden Features

‣ Problem: Feature overridden due to configuration changes done in dev, stage or prod.

‣ Fix: You’re doing it wrong!

‣ Adopt a proper dev-> stage -> prod workflow.

‣ Automatically revert Features on deployments. Script drush fr.

‣ “drush fu” your workflow. (best Drush command ever!)
Aaaaand...

‣ It’s all about to change in Drupal 8

Home » Administration » Configuration » Development

Synchronize configuration

Import configuration that is placed in your staging directory. All changes, deletions, renames, and additions are listed below.

1 changed

<table>
<thead>
<tr>
<th>Name</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>views.view.articles</td>
<td>View differences</td>
</tr>
</tbody>
</table>

Import all
What about if you stop maintaining code whatsoever? There are platforms such as Acquia’s Site Factory that let you create hundreds of sites without any code if your sites are easy enough.
What have we learned

‣ Managing a lot of sites is hard. Deal with it, budget for it

‣ Foster an internal community

‣ Create a platform / distribution

‣ Featurize your code

‣ Consider SaaS
Thank you, questions?

Jakub Suchy, jakub.suchy@acquia.com
@jakubsuchy