Opening The Black Box

DrupalCamp CT 8/20/2016

FOUR KITCHENS

Howdy!
I'm an engineer at
Four Kitchens

Dustin Younse
@milsyobtaf

We're a Drupal shop based out of Austin, TX. I head our support team, which means I do a lot of somewhat repetitive work, but every once in a while a client asks us to do something particularly cool, which is what I'm here to talk about.

DrupalCamp CT 2014
DrupalCamp CT 2016 (morning)
DrupalCamp CT 2016 (afternoon)

I guess I’m an Ivy League assistant professor now?

I’ve spoken here before, and I’ll speak here again. Where do I pick up my faculty ID?

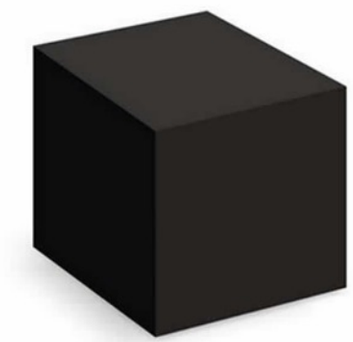
What Is The Black Box?

A Familiar Story?

- Non-technical education
- Self taught web skills
- Above average Googling abilities
- Late nights staring at error messages

Does this sound familiar?

Mental Model of Drupal

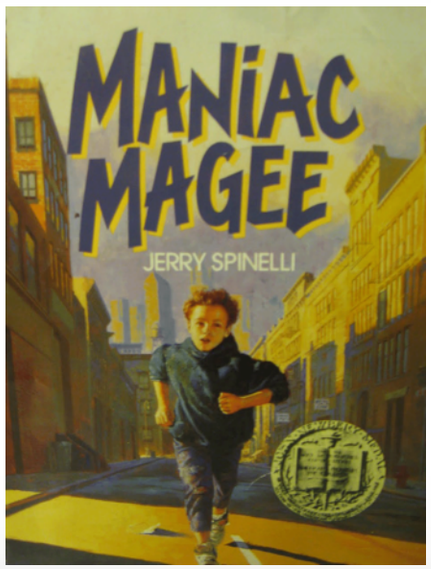


An impenetrable monolith, no ins, no outs, but it just seems to work. Until it doesn’t.

Mental Model of Drupal

Maybe it’s that same monolith, but with a shiny logo and a light. Not sure what the light does, but hey, it’s there.

Mental Model of Drupal

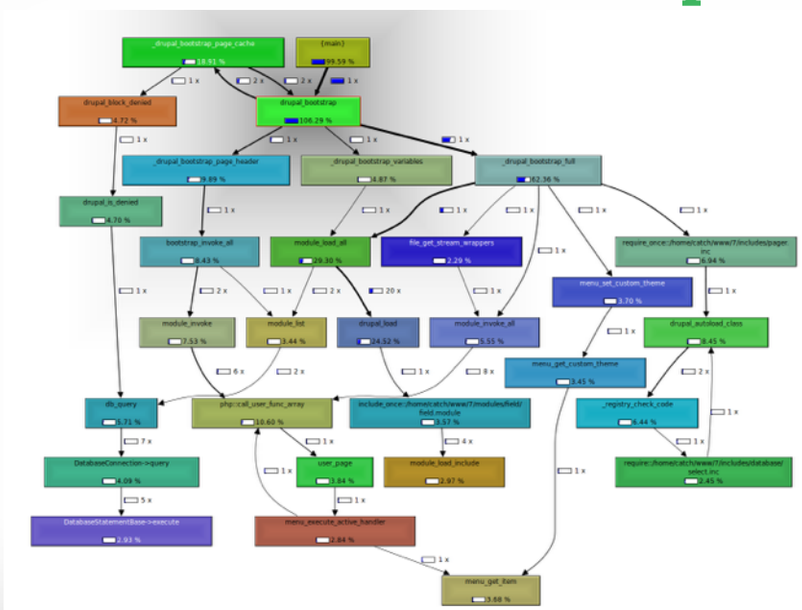


A childhood favorite, which taught me a very important lesson about patience and methodical problem solving. The title character, Maniac, is faced with a seemingly impossible challenge. A local store is offering one full year of free pizza to anyone capable of untangling an incredibly large knot.

Mental Model of Drupal

This not. It takes him days and days of study and probing, but he finally manages to untangle it. I think this makes a much better mental model of Drupal. There are lots of ins and outs, loops and twists, but at the end of the day, it's one single piece of rope. There is a beginning, there is an end, and the trick is simply to figure out how to modify things along the way.

Mental Model of Drupal



This is only the highest level of a Drupal 7 bootstrap process. Each of these boxes probably has 2 or 3 other levels inside of it that look exactly like this. But at the end of the day, there is one input and one output, and we can view behavior at any of these points.

Start At The Beginning

The first thing we need to do is find and become familiar with our tools.

Text Editors

- “Text Editors”
- Text Editors
- IDEs

There are several levels of text editors

“Text Editors”

- Text Edit (Mac)
- Notepad (Windows)
- Nano (Linux / Mac)

These are pretty barebones, allowing you to edit text, but typically don’t have syntax highlighting or formatting options. In a pinch they are useful, but I wouldn’t recommend these for a daily driver.

Text Editors

- Notepad++ (Windows)
- vim (Mac / Linux)
- Dreamweaver (Mac / Windows)
- Sublime Text (Mac / Linux / Windows)
- Atom (Mac / Linux / Windows)

These let you edit your code, use syntax highlighting and code formatting plugins, and often have code autocompletion features available. These are the tool of choice for many developers today.

Integrated Development Environments

- PHPStorm (Mac / Linux / Windows)
- Eclipse (Mac / Linux / Windows)
- Komodo (Mac / Linux / Windows)

These are yet another step up in features, with all the abilities of a solid text editor, plus integrated debugging, advanced code autocomplete and more. While many of these features are available as plugins on text editors, here they come out of the box. And can be difficult to disable, which turns many developers away.

Development Environments

- On your server
- On your machine
- On your machine, but it acts like the server

On your server is a standard starting point for development.
On your local machine is a logical next step, but often doesn’t replicate your server very well.
Then there is the next level of development, where you develop locally on a machine that acts almost identical to the server

Development Environments

- FTP Development
  - “Cowboy coding”
- Local Environments
  - MAMP / LAMP / Vagrant
- Next generation
  - Kalabox

Once you have your tools, you need to begin working out your environment. The first place you wind up is typically working directly on a server. Maybe its a staging server, maybe its the production server, where cowboys like to code.
A step up from this is some sort of local dev environment. This can be something out of the box like MAMP, or an OS X / Linux style LAMP setup, or a Vagrant box. All of these have their advantages and disadvantages. Vagrant is great for closely replicating a production environment, but it isn’t perfect.
Kalabox is a newer system that allows for near perfect replication of complex environments like Pantheon, on your local machine.

FTP Development

- FTP Development
  - “Cowboy coding”
  - No version control up front
  - Little debugging visibility
    - Log files (maybe), debug statements
    - Watchdog logs (maybe)

Developing directly on a server is a straight forward, efficient way of developing. You save your file, and the change is live on the site. This, of course, means you can’t mess up without taking down the site for some period of time. Yeehaw!
In addition, you typically don’t have the protection of version control like Git, until after the fact.
Your debugging is very limited, and very reactionary. You can see what went wrong with log files, but only if you have the correct server permissions. You can see what a variable looks like at a specific point using a debug statement.

Debug Statements in Code

- print_r()
  - Raw dump of a variable or context, no formatting
  - print_r($node);
    - <pre><?php print_r($node); ?></pre>
  - dsm()
    - Formatted dump of a variable or context
    - dsm($node);

These are the roughly the extent of your debugging abilities while working remotely.

Local Development

- MAMP / Dev Desktop / Native LAMP
  - Code runs entirely locally
  - Added bonus of offline work, no internet required
- print_r(), dsm(), but also more
- xdebug

xdebug allows you to inspect the state of the entire application at a given breakpoint, not just the state of individual variables.
This is a great way to visualize the many, many loops and detours that Drupal takes from request to finished page.
IDEs are built around the concept of debugging, and most PHP IDEs have xdebug as a central feature.

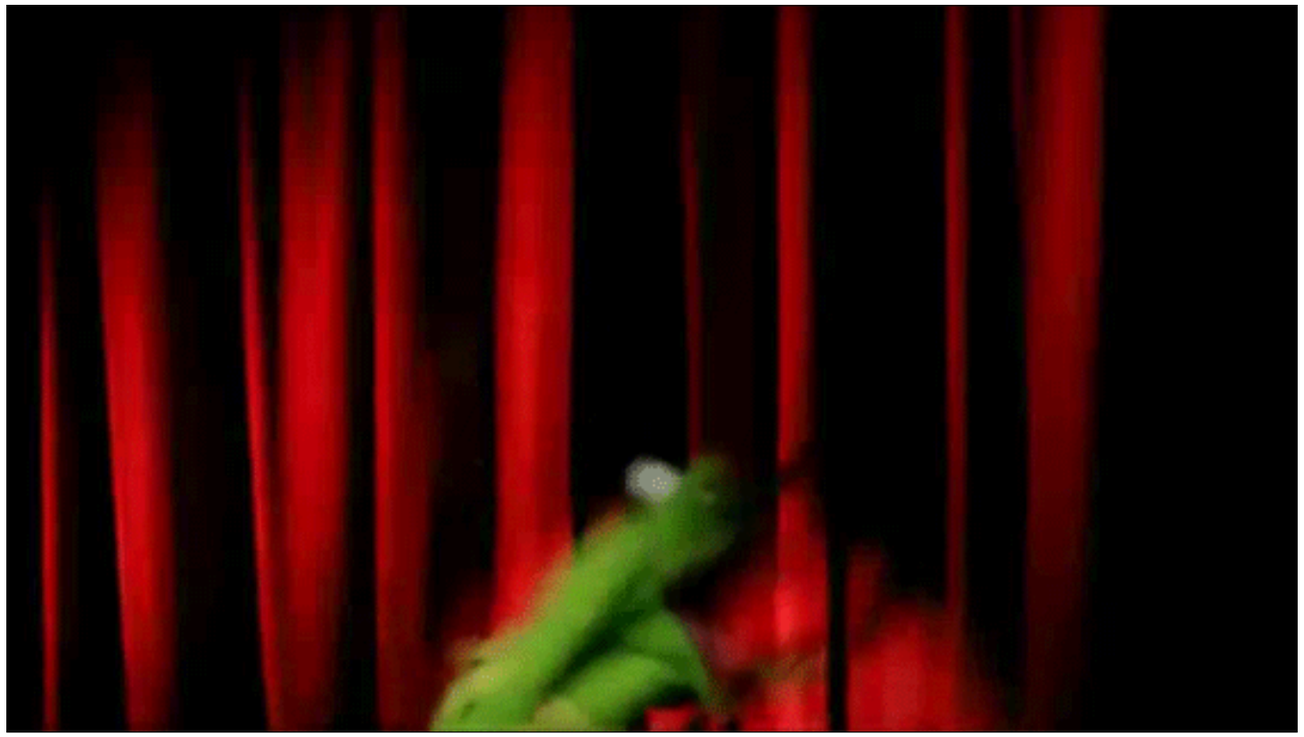
Local Development: The Next Generation

- Kalabox with Pantheon / Acquia integration
- Custom Vagrant boxes (DrupalVM)
- print_r(), dsm(), xdebug
- Very accurate performance profiling
- XHProf

This type of development gives you all of the advantages and shortcuts of local development while also giving you a mirror image of the final production environment. This allows you to do things like more accurately profile the memory usage of a module.

Something Is Broken!

I’m a support developer, so my default state is panic and terror.
I get the cranky, worried, angry emails and phone calls when the site isn’t working. It’s easy at this point to start getting cranky, worried or angry yourself, but the trick is:



The site is down!

Relax.

Relax. Stop and think about the situation.

Remember Cobble's Knot

Getting angry with the knot isn't going to help.
Getting frustrated with the knot isn't going to help.
The knot is still going to be the knot.

What's Broken?

- Is something not showing up?
  - New content - is it published?
  - Old content - are permissions set properly?
- Is something showing up that shouldn't?
  - Raw HTML and Javascript in a WYSIWYG field?

First you need to figure out what is broken. It doesn't matter where it's broken, we'll get to that.

Work From The Bottom Up

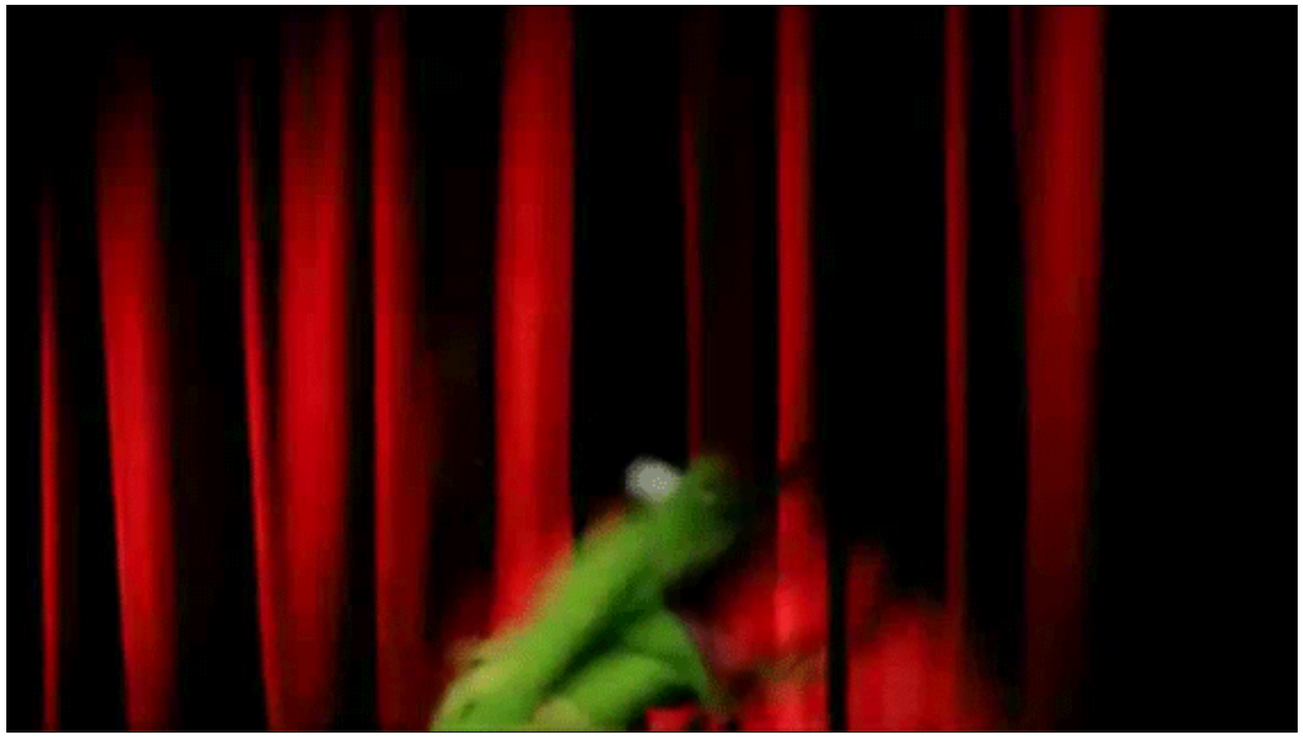
- Log Files
  - You should learn how to find your log files before you need your log files
- multitail
  - Linux / Mac utility to easily view logs with more options than tail

If you’re getting the old White Screen Of Death, most of your Drupal debugging techniques are out the window - for now.
multitail lets you output multiple log files together, so you can view Apache and mysql logs in the same stream.
Log errors will hopefully point you to a specific line on a specific file, but sometimes they can only give you something to Google for.

Where Is It Broken?

- Custom Module
- Theme template.php
- Theme Template

Hopefully by now you know _what_ is broken, but you still need to find _where_ it is broken.
For most of what we do in Drupal, it's going to be broken in three places. It can help to look at these from the bottom up as well.
The lowest level is likely to be in the custom module logic.
The next level where data is typically modified is in the theme's template.php. A quick and dirty way to see if the issue lies in a theme is to change the theme. If the problem disappears, you know the gremlin must be inside the theme.
By the time you are working in a theme template, you are working with the final processed data, so you can do some dirty things here, if necessary, to make the data match what you need. Drupal 8 is taking away that option, since templates will be fully PHP-less, but that isn't necessarily a bad thing, particularly for future support of a project.



I don’t know if you forgot, but the site is down. Again. We gotta relax.

The Scientific Method

Now that we know what we are working on and where we are working on it, it’s important that we slow down and apply a bit of scientific rigor.

Finally, Debugging

- Change ***one*** thing at a time
- Test that change
- Repeat

We need to do this methodically. Change one thing. Test it. Repeat.

git Is Your Friend

- Save your progress as you work
  - Recreate your Features
- Makes rabbit holes manageable

Recreate Features. Make branches. Use git stash. This way a 2 hour rabbit hole doesn’t become a black hole, just something you can easily back out of.

git diff Is Your Friend

- Remove your debug statements
- Ensure you only changed as much as you needed to change
- You only commit dsm('Butts'); to master ***once***

Before you call it a day, make sure you carefully go through your git diff. You only commit something embarrassing to master once. I hope.

git blame Finds Your Enemies

git blame ~~Finds Your Enemies~~

git blame Is Your Friend

- Can help you track down who wrote the offending code
  - This ***should not*** be a witch hunt
  - This ***should*** be a chance to find context for the issue

Make The Future Easier

Most of my time is dedicated to working in other people’s code, but once it’s in my hands, I try to make it easier to work with going forward

Proactive Debugging

- watchdog() (D7)
- \Drupal::logger() (D8)
- syslog Module
  - http://loggly.com
  - https://www.elastic.co/blog/heya-elastic-stack-and-x-pack
- Write a test!
  - Simpletest
  - Behat

Now that you've fixed the problem, it's time to take a minute and think about the future, make things better for you down the road.
Watchdog is great, but is database bound, and often truncates right before the data you need.
Syslog lets you write Drupal events to the system log instead, giving you access to more space and lets you integrate with standard logging tools like Loggly, which monitors and can alert you in real time of a logged issue.

@milsyobtaf

Thank you!