goto; conference aarhus

CONTINUOUS DELIVERY: THE DIRTY DETAILS

Mike Brittain

Etsy.com

@mikebrittain
mike@etsy.com

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com

a.k.a. “**Continuous Deployment**”

Etsy

www.Etsy.com

Etsy

GROSS MERCHANDISE SALES

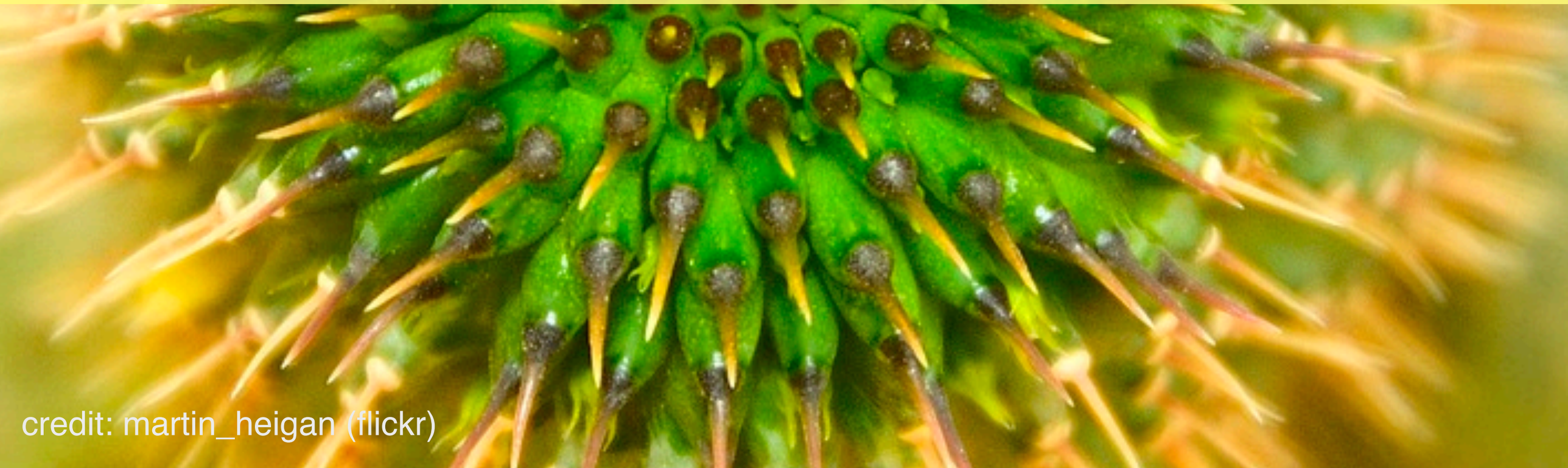
Etsy

AUGUST 2012

1.4 Billion page views
USD $76 Million in transactions
3.8 Million items sold

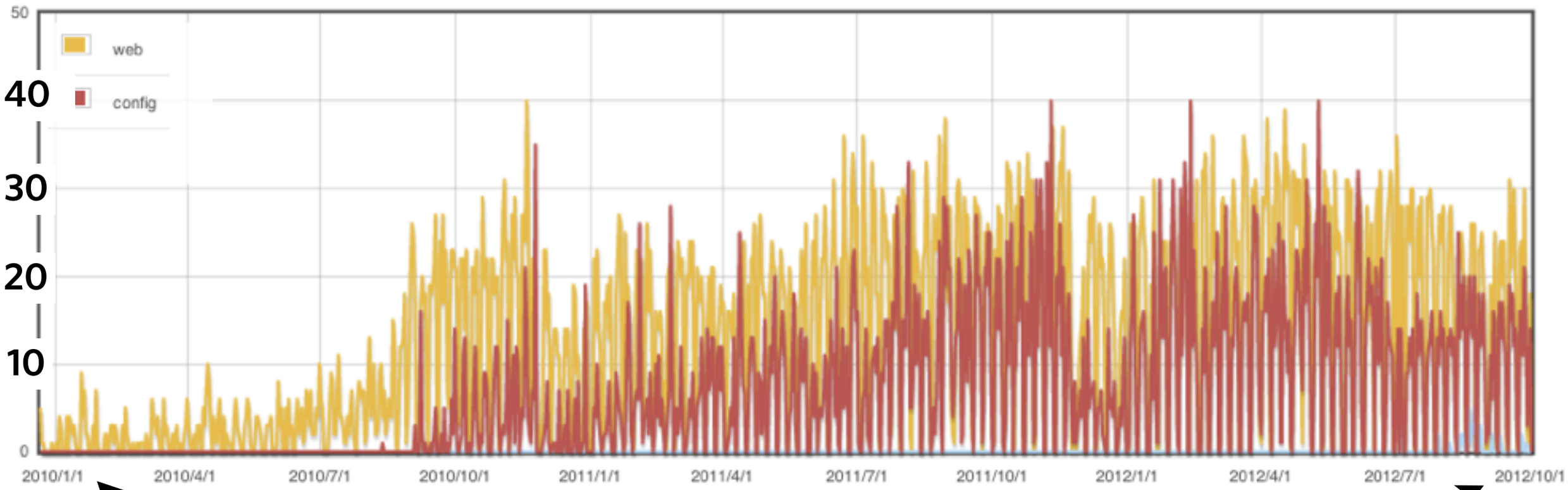
http://www.etsy.com/blog/news/2012/etsy-statistics-august-2012-weather-report/

~170 Committers, everyone deploys



credit: martin_heigan (flickr)

Deployments Per Day (US/Eastern)



Very end of 2009

Today

Continuous delivery is a pattern language in growing use in software development to improve the process of software delivery. Techniques such as **automated testing, continuous integration,** and **continuous deployment** allow software to be developed to a high standard and easily packaged and deployed to test environments, resulting in the ability to rapidly, reliably and **repeatedly push** out enhancements and bug fixes to customers at **low risk** and with **minimal manual overhead**. The technique was one of the assumptions of extreme programming but at an enterprise level has developed into a discipline of its own, with job descriptions for roles such as "buildmaster" calling for CD skills as mandatory.

~wikipedia

+ DevOps
+ Working on mainline, trunk, master
+ Feature flags
+ Branching in code

An Apology

An Apology

We build primarily in PHP.
Please don't run away!

The Dirty Details of...

"Continuous Deployment in Practice at Etsy"

Deploy to Production

Then

2009

Now

2010-today

Just before we started using CD

Then	Now
6-14 hours	15 mins
“Deployment Army”	1 person
Highly orchestrated and infrequent	Rapid release cycle

Then

Special event and highly disruptive

Now

Commonplace and happens so often we cannot keep up

Then

Blocked for
6-14 hours,
plus minimum of
6 hours to
redeploy

Now

Blocked for
15 minutes,
next deploy will
only take
15 minutes

Config flags <5 mins

Then

Release branch,
database schemas,
data transforms,
packaging,
rolling restarts,
cache purging,
scheduled downtime

Now

Mainline,
minimal linking
and building,
rsync,
site up

Then	Now
Slow	Fast
Complex	Simple
Special	Common

Deploying code is the very first thing engineers learn to do at Etsy.

1ST DAY

Add your photo to Etsy.com.

1ST DAY

Add your photo to Etsy.com.

2ND DAY

Complete tax, insurance, and benefits forms.

Deploy to Production

WARNING

Etsy

GROSS MERCHANDISE SALES

Continuous Deployment

Small, frequent changes.
Constantly integrating into production.
30 deploys per day.

“Wow… 30 deploys a day. How do you build features so quickly?”

Software Deploy ≠ Product Launch

Deploys frequently gated by **config flags**

("dark" releases)

```
$cfg[‘new_search’] = array('enabled' => 'off');
$cfg[‘sign_in’]    = array('enabled' => 'on');
$cfg[‘checkout’]   = array('enabled' => 'on');
$cfg[‘homepage’]   = array('enabled' => 'on');
```

```
$cfg[‘new_search’] = array('enabled' => 'off');
```

```
$cfg[‘new_search’] = array('enabled' => 'off');

// Meanwhile...



# old and boring search
$results = do_grep();
```

```php
$cfg[‘new_search’] = array('enabled' => 'off');

// Meanwhile...

if ($cfg[‘new_search’] == ‘on’) {
  # New and fancy search
  $results = do_solr();
} else {
  # old and boring search
  $results = do_grep();
}
```

```
$cfg[‘new_search’] = array('enabled' => 'on');

// or...

$cfg[‘new_search’] = array('enabled' => 'staff');

// or...

$cfg[‘new_search’] = array('enabled' => '1%');

// or...

$cfg[‘new_search’] = array('enabled' => 'users',
                           'user_list' => 'mike,john,kellan');
```

Validate in production, hidden from public.

What's in a deploy?

Small incremental changes to the application
New classes, methods, controllers
Graphics, stylesheets, templates
Copy/content changes

Turning flags on/off, or ramping up

Quickly Responding to issues

Security, bugs, traffic, load shedding, adding/removing infrastructure.

Tweaking config flags or releasing patches.

http://www.flickr.com/photos/flyforfun/2694158656/

http://www.flickr.com/photos/flyforfun/2694158656/

Summer's Glow - Mason Jar Chandelier Lighting Fixture

$130.00 USD Only 1 available

Add to Cart | Favorite

Shop

BootsNGus

A place where Handmade and Vintage come together.

192 items

Add shop to favorites

See who favorites this shop

Shop owner

Jeff and Mark

Ann Arbor and all over the Mitten State

Favorites

Circle

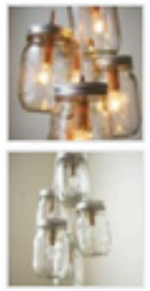
Feedback: 802, 100% pos.

Contact

Item

Item added to favorites

Summer's Glow - Mason Jar Chandelier Lighting Fixture



$130.00 USD Only 1 available

Add to Cart

Shop

BootsNGus

A place where Handmade and Vintage come together.

192 items

Shop owner

Jeff and Mark

Ann Arbor and all over the Mitten State

Circle

Feedback: 802, 100% pos.

Contact

Item

Tweet Pin it Like 32

64949 views

“How do you continuously deploy database schema changes?”

Code deploys: ~ every 15-20 minutes

Schema changes: Thursday

Our web application is largely monolithic.

Etsy.com, support tools, developer API, back-office, analytics

External “services” are not deployed with the main application.

Databases, Search, Photo storage

For every config flag, there are two states we can support — forward and backward.

Expose multiple versions in each service.

Expect multiple versions in the application.

Example: Changing a Database Schema

Prefer ADDs over ALTERs (“non-breaking expansions”)

Altering in-place requires coupling code and schema changes.

Merging “users” and “users_prefs”

1. Write to both versions
2. Backfill historical data
3. Read from new version
4. Cut-off writes to old version

0. Add new version to schema
1. Write to both versions
2. Backfill historical data
3. Read from new version
4. Cut-off writes to old version

0. Add new version to schema

Schema change to add prefs columns to “users” table.

“write_prefs_to_user_prefs_table” => “on”
“write_prefs_to_users_table” => “off”
“read_prefs_from_users_table” => “off”

1. Write to both versions

Write code for writing prefs to the “users” table.

“write_prefs_to_user_prefs_table” => “on”
“write_prefs_to_users_table” => “on”
“read_prefs_from_users_table” => “off”

2. Backfill historical data

Offline process to sync existing data from “user_prefs” to new columns in “users”

3. Read from new version

Data validation tests. Ensure consistency both internally and in production.

“write_prefs_to_user_prefs_table” => “on”
“write_prefs_to_users_table” => “on”
“read_prefs_from_users_table” => “staff”

3. Read from new version

Data validation tests. Ensure consistency both internally and in production.

“write_prefs_to_user_prefs_table” => “on”
“write_prefs_to_users_table” => “on”
“read_prefs_from_users_table” => “1%”

3. Read from new version

Data validation tests. Ensure consistency both internally and in production.

“write_prefs_to_user_prefs_table” => “on”
“write_prefs_to_users_table” => “on”
“read_prefs_from_users_table” => “5%”

3. Read from new version

Data validation tests. Ensure consistency both internally and in production.

“write_prefs_to_user_prefs_table” => “on”
“write_prefs_to_users_table” => “on”
“read_prefs_from_users_table” => “on”

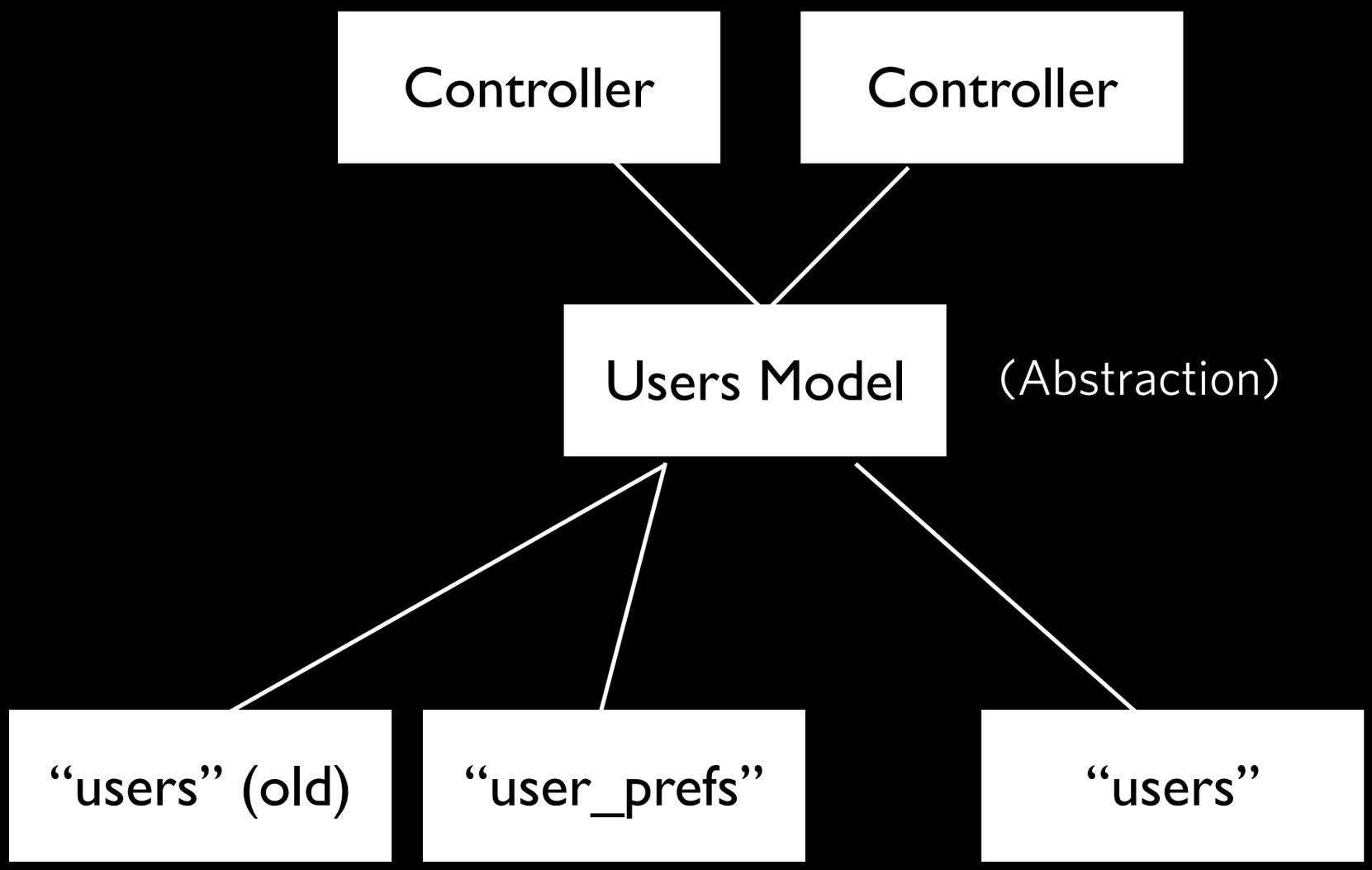
(“on” == “100%”)

4. Cut-off writes to old version

After running on the new table for a significant amount of time, we can cut off writes to the old table.

“write_prefs_to_user_prefs_table” => “off”
“write_prefs_to_users_table” => “on”
“read_prefs_from_users_table” => “on”

"Branch by Astraction"



http://paulhammant.com/blog/branch_by_abstraction.html
http://continuousdelivery.com/2011/05/make-large-scale-changes-incrementally-with-branch-by-abstraction/

"The Migration 4-Step"

1. Write to both versions
2. Backfill historical data
3. Read from new version
4. Cut-off writes to old version

"The Migration 4-Step"

1. Write to both versions
2. Backfill historical data
3. Read from new version
4. Cut-off writes to old version
5. Clean up flags, code, columns *(when?)*

Architecture and Process

Deploying is cheap.

Some philosophies on product development…

Gathering data should be cheap, too.

staff, opt-in prototypes, 1%

Treat first iterations as experiments.

Get into code as quickly as possible.

Architecture largely doesn't matter.

Kill things that don’t work.

“Terminate with extreme predjudice.”

Is the dumb solution enough to build a product?

How long will the dumb solution last?

Your assumptions will be wrong once you've scaled 10x.

"We don't optimize for being right. We optimize for quickly detecting when we're wrong."

~Kellan Elliott-McCrea, CTO

Become really good at changing your architecture.

Invest time in architecture by the 2nd or 3rd iteration.

Integration and Operations

Continuous Deployment

Small, frequent changes.

Constantly integrating into production.

30 deploys per day.

Code review before commit

Automated tests before deploy

Why Integrate with Production?

Dev ≠ Prod

Verify frequently and in small batches.

Integrating with production is a test in itself.

We do this *frequently and in small batches.*

"Production is truly the only place you can validate your code."

"Production is truly the only place you can validate your code."

~ Michael Nygard, about 40 min ago

More database servers in prod.
Bigger database hardware in prod.
More web servers.
Various replication schemes.
Different versions of server and OS software.
Schema changes applied at different times.
Physical hardware in prod.
More data in prod.
Legacy data (7 years of odd user states).
More traffic in prod.
Wait, I mean MUCH more traffic in prod.
Fewer elves.
Faster disks (SSDs) in prod.

Using a MySQL database to test an application that will eventually be deployed on Oracle:

Using a MySQL database to test an application that will eventually be deployed on Oracle: Priceless.

Verify frequently and in small batches.

Dev ≠ Prod

Dev → QA → Staging → Prod

Dev → QA → Staging → Prod

Dev → Pre-Prod → Prod

Test and integrate where you'll see value.

Config flags (again)

off, on, staff, opt-in prototypes, user list, 0-100%

Config flags (again)

off, on, staff, opt-in prototypes, user list, 0-100%

"canary pools"

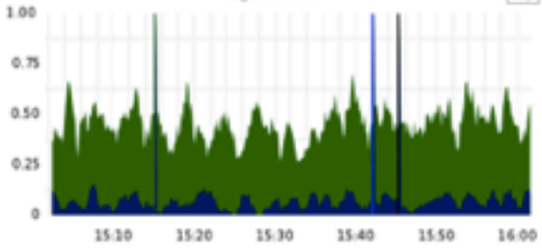
Automated tests **after deploy**

Real-time metrics and dashboards

Network & Servers, Application, Business

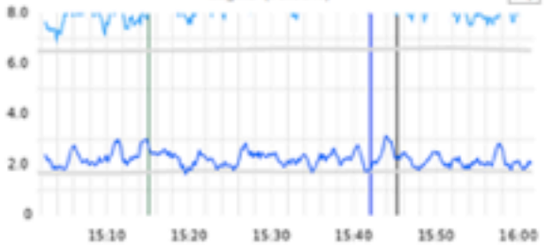
business graphs (1 hour)

Registrations



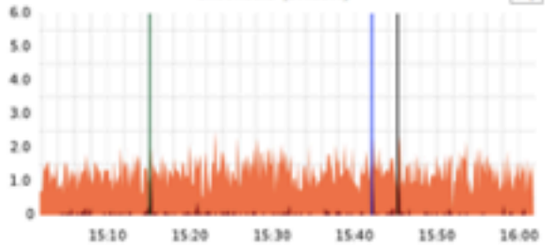
facebook · other

Logins (78.58%)



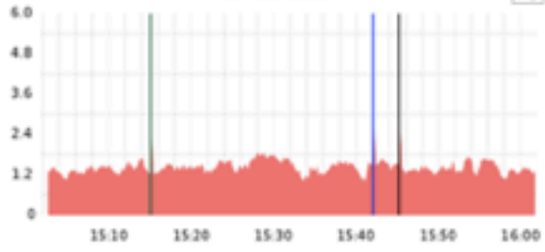
· logins · errors

Checkouts (97.26%)



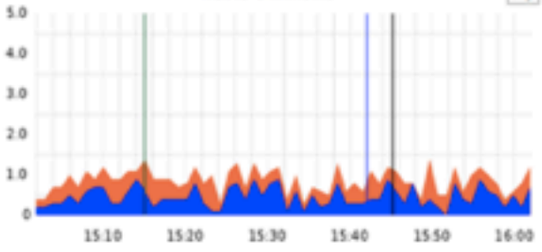
errors · checkouts

Post Checkouts



post checkouts

Mobile Checkouts



iOS · mobile web

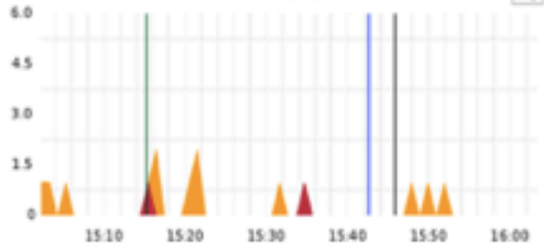
Listings Creation/Renewal

New & Copied · Renewed

Search Ads

Ad requests · Ad responses · Ads logged

Forum Posts



bugs forum · help forum

page performance (1 hour)

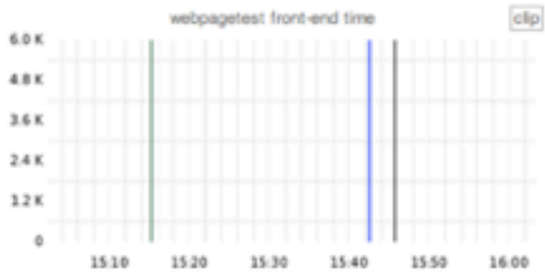
PHP time by site section (median)

view_listing · shop_home · view_profile · search · home

PHP time by site section (95th percentile)

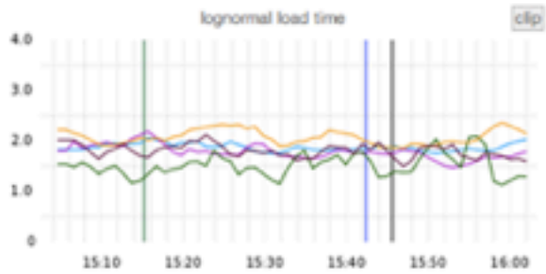
view_listing · shop_home · view_profile · search · home

webpagetest front-end time



view_listing · shop_home · view_profile · search · home

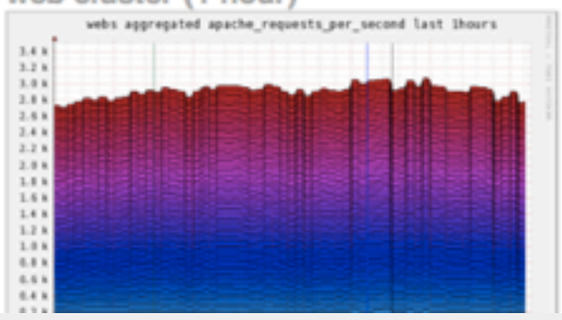
lognormal load time



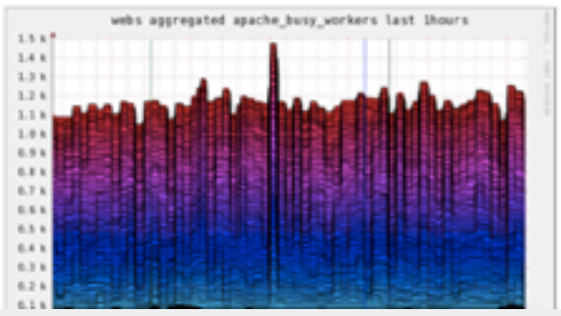
view_listing · shop_home · view_profile · search · home

web cluster (1 hour)

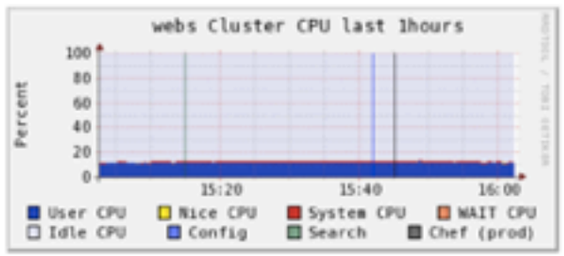
webs aggregated apache_requests_per_second last 1hours



webs aggregated apache_busy_workers last 1hours



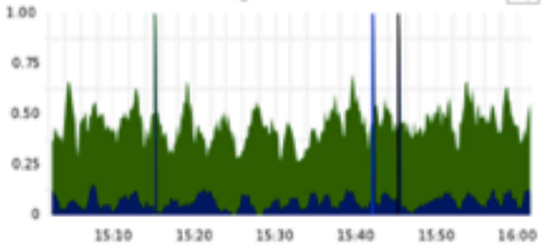
webs Cluster CPU last 1hours



Release Managers: 0

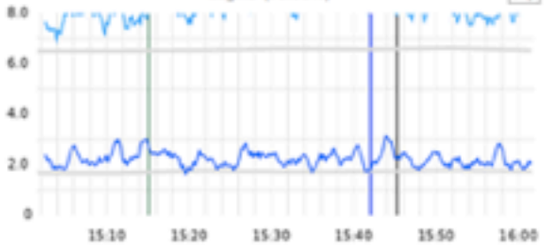
business graphs (1 hour)

Registrations



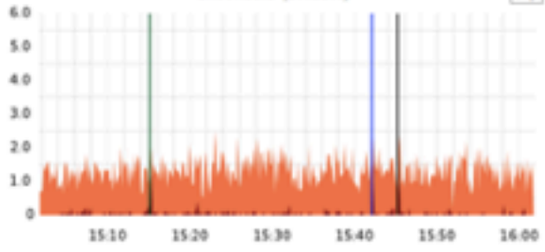
facebook · other

Logins (78.58%)



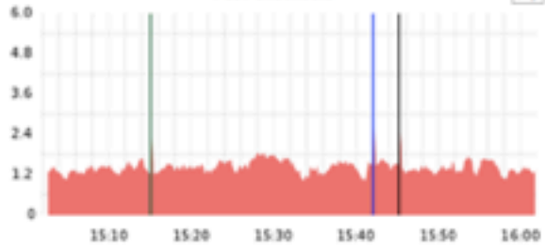
· logins · errors

Checkouts (97.26%)



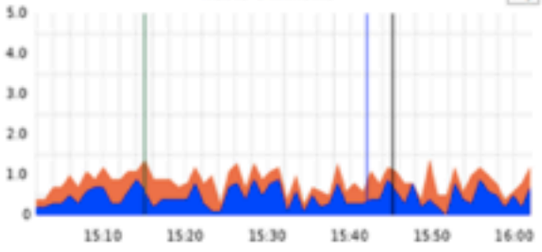
errors · checkouts

Post Checkouts



post checkouts

Mobile Checkouts



iOS · mobile web

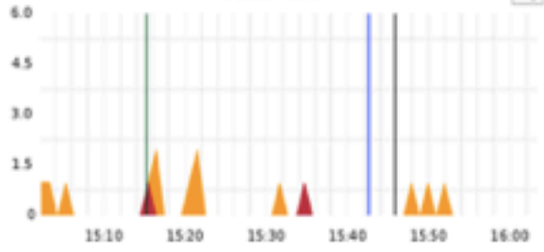
Listings Creation/Renewal

New & Copied · Renewed

Search Ads

Ad requests · Ad responses · Ads logged

Forum Posts



bugs forum · help forum

page performance (1 hour)

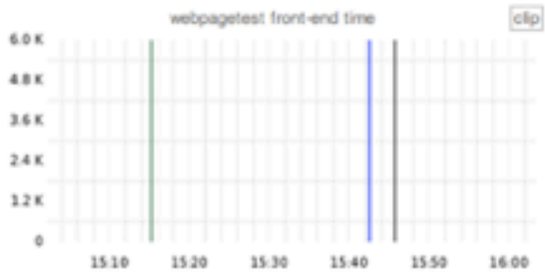
PHP time by site section (median)

view_listing · shop_home · view_profile · search · home

PHP time by site section (95th percentile)

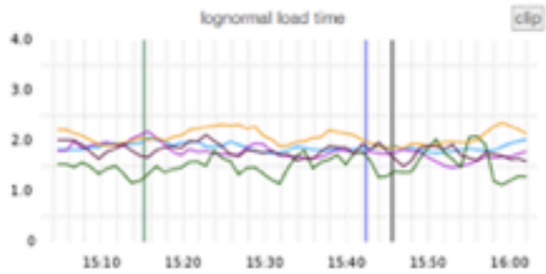
view_listing · shop_home · view_profile · search · home

webpagetest front-end time



view_listing · shop_home · view_profile · search · home

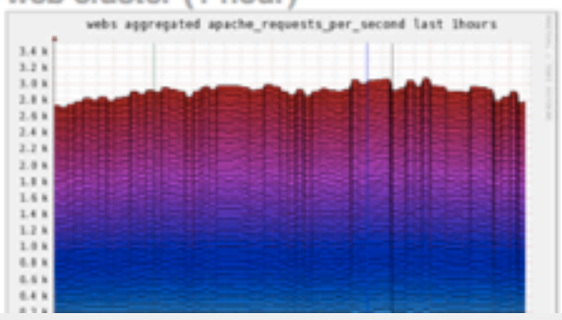
lognormal load time



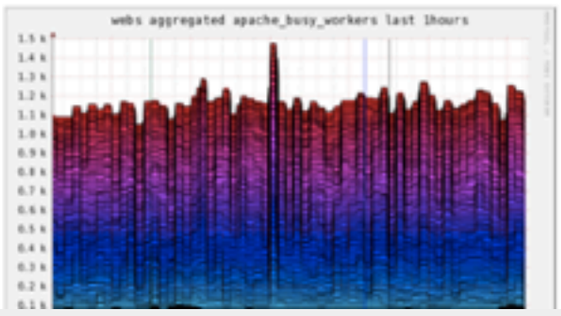
view_listing · shop_home · view_profile · search · home

web cluster (1 hour)

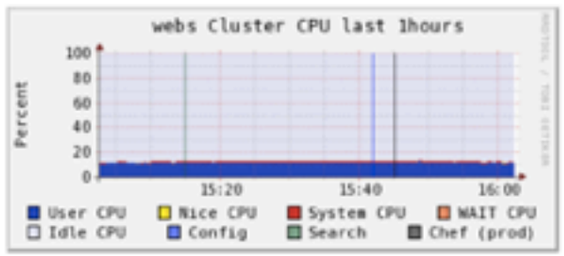
webs aggregated apache_requests_per_second last 1hours



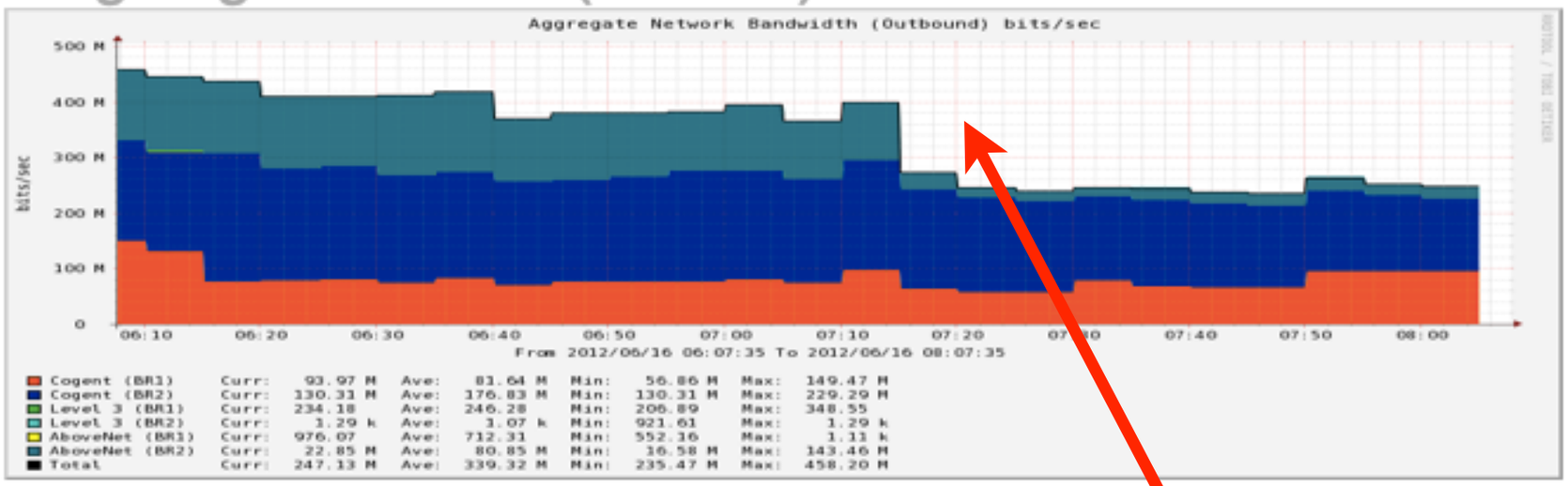
webs aggregated apache_busy_workers last 1hours



webs Cluster CPU last 1hours



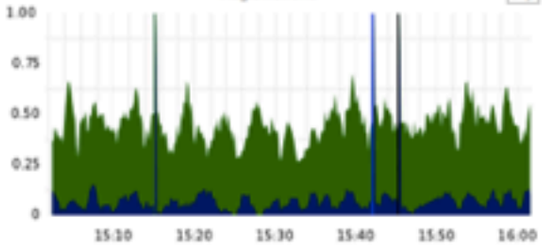
outgoing bandwidth (2 hours)



Is it Broken?
Or , is it just better?

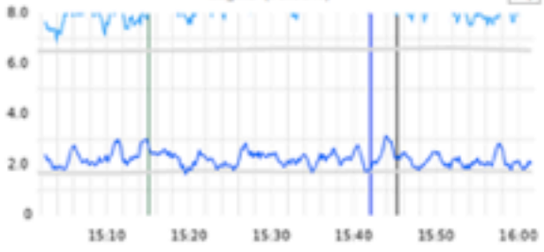
business graphs (1 hour)

Registrations



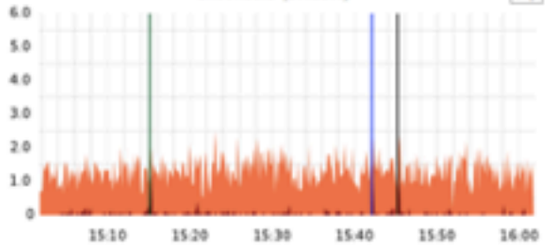
facebook · other

Logins (78.58%)



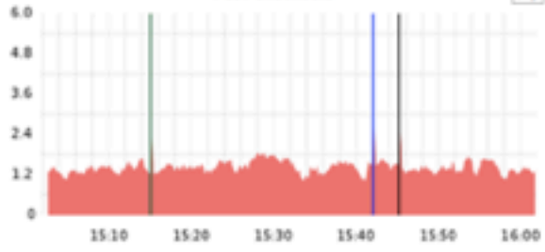
· logins · errors

Checkouts (97.26%)



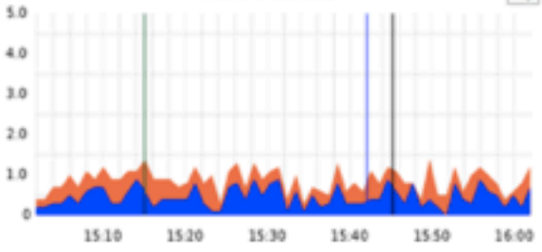
errors · checkouts

Post Checkouts



post checkouts

Mobile Checkouts



iOS · mobile web

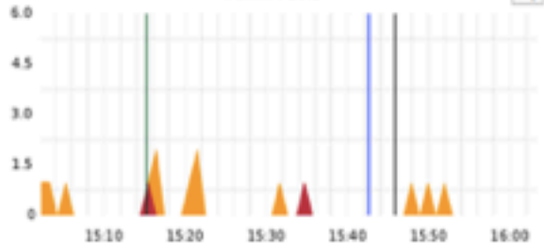
Listings Creation/Renewal

New & Copied · Renewed

Search Ads

Ad requests · Ad responses · Ads logged

Forum Posts



bugs forum · help forum

page performance (1 hour)

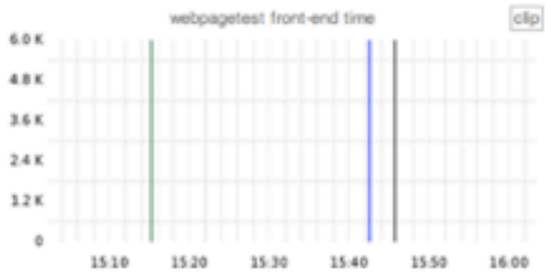
PHP time by site section (median)

view_listing · shop_home · view_profile · search · home

PHP time by site section (95th percentile)

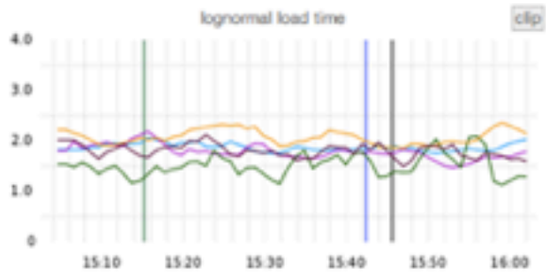
view_listing · shop_home · view_profile · search · home

webpagetest front-end time



view_listing · shop_home · view_profile · search · home

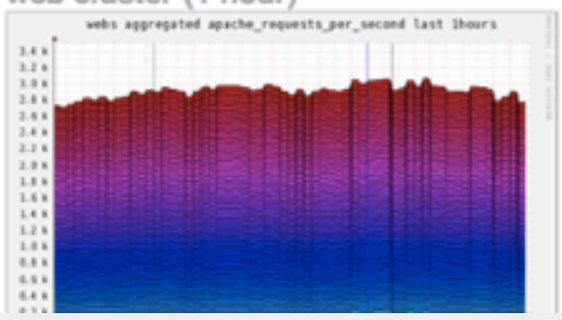
lognormal load time



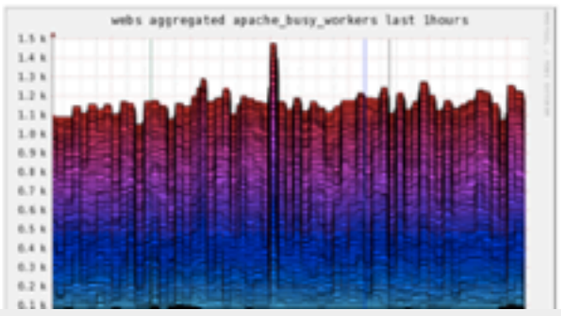
view_listing · shop_home · view_profile · search · home

web cluster (1 hour)

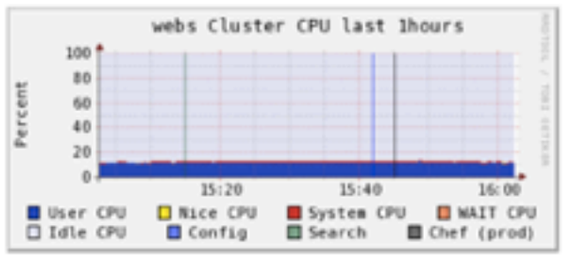
webs aggregated apache_requests_per_second last 1hours

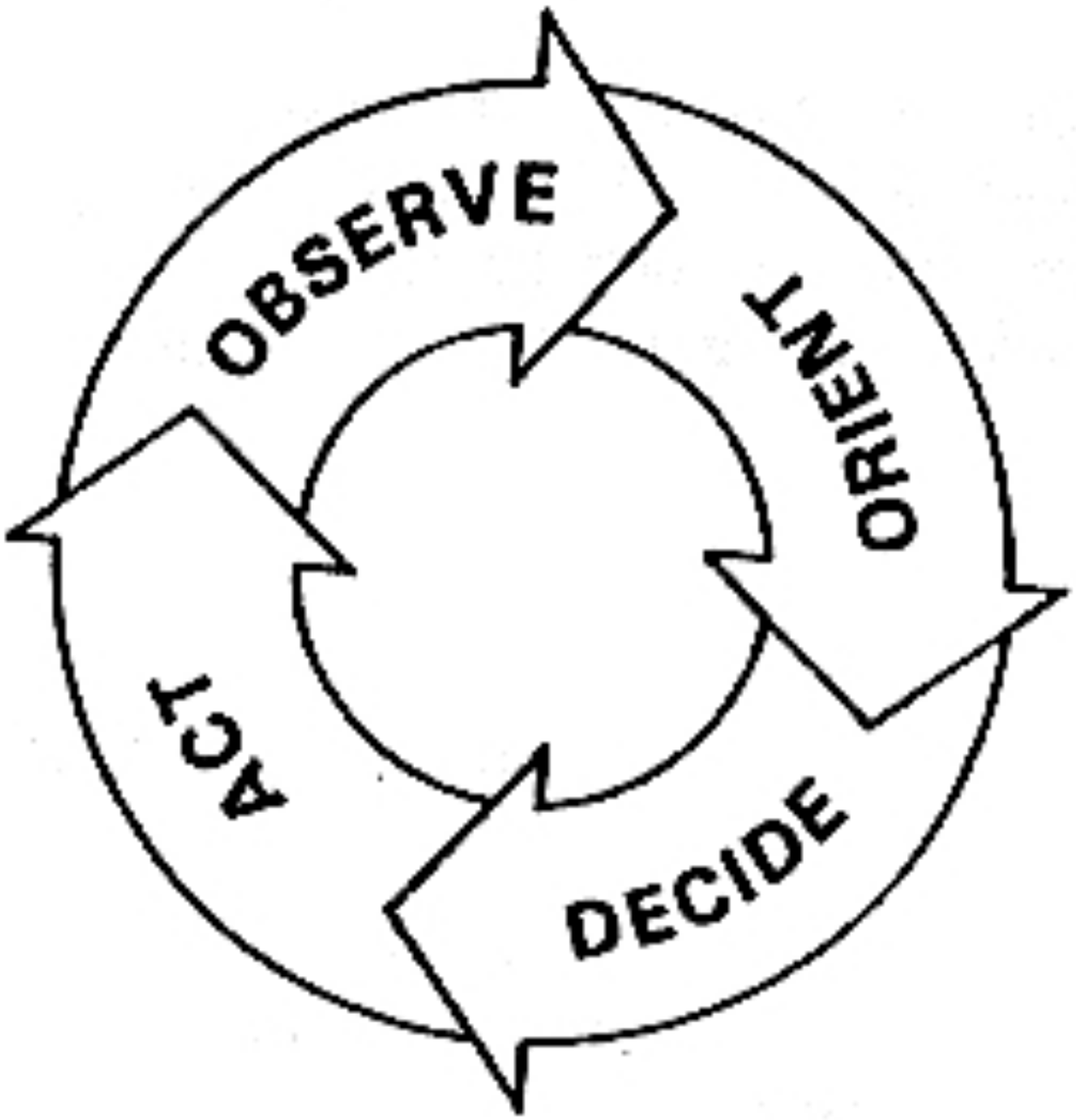


webs aggregated apache_busy_workers last 1hours



webs Cluster CPU last 1hours

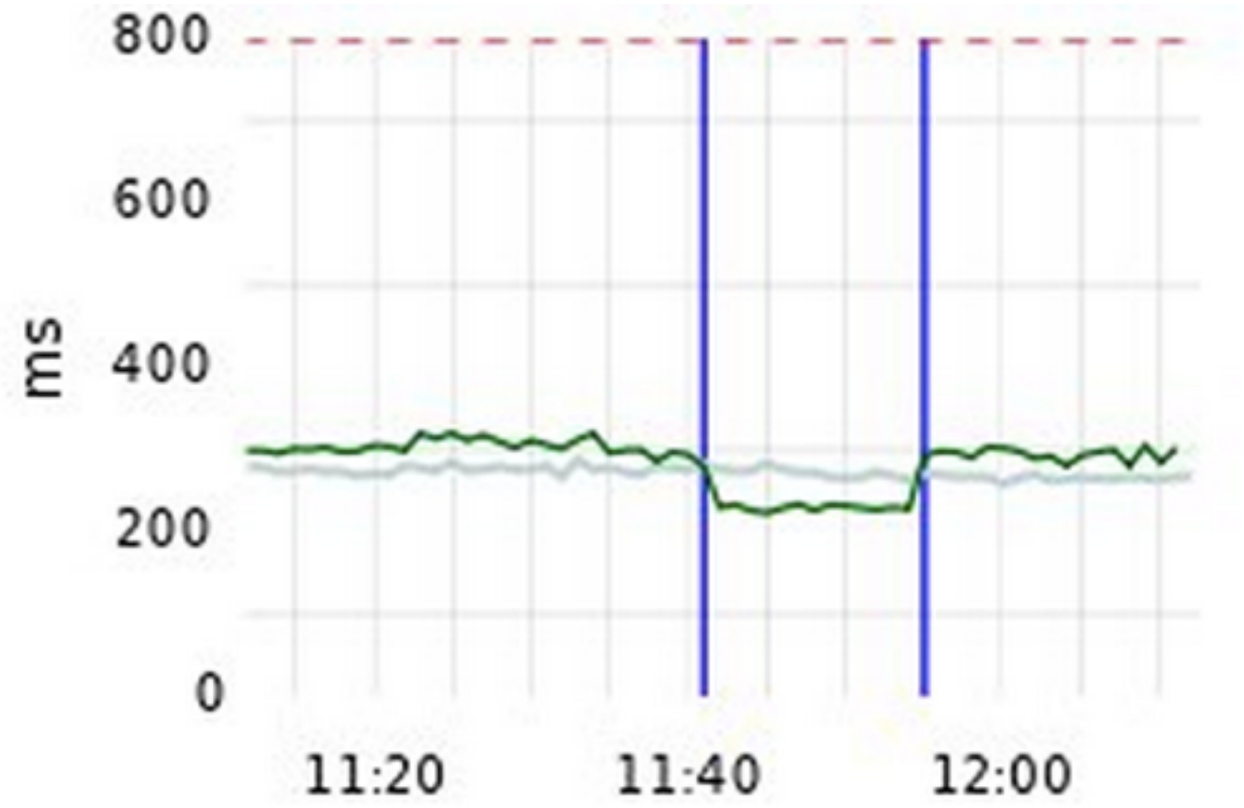




Metrics + Configs → OODA Loop

“Theoretical” vs. “Practical”

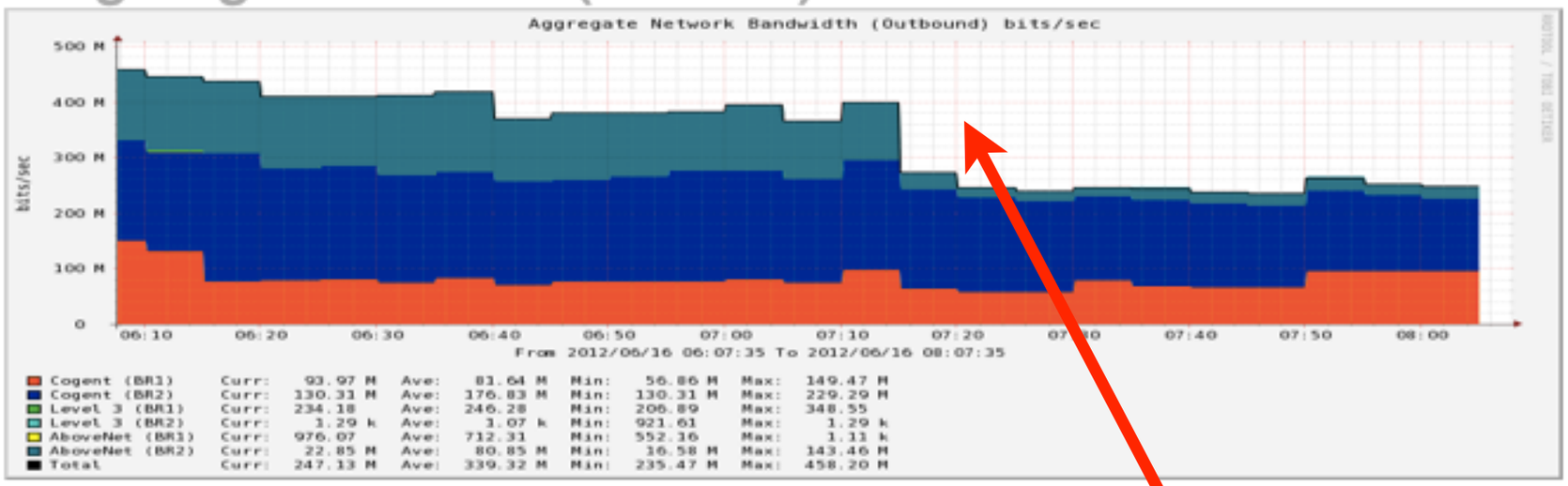
Homepage (95th perc.)



Surprise!!!

Turning off multi-language support improves our page generation times by up to 25%.

outgoing bandwidth (2 hours)



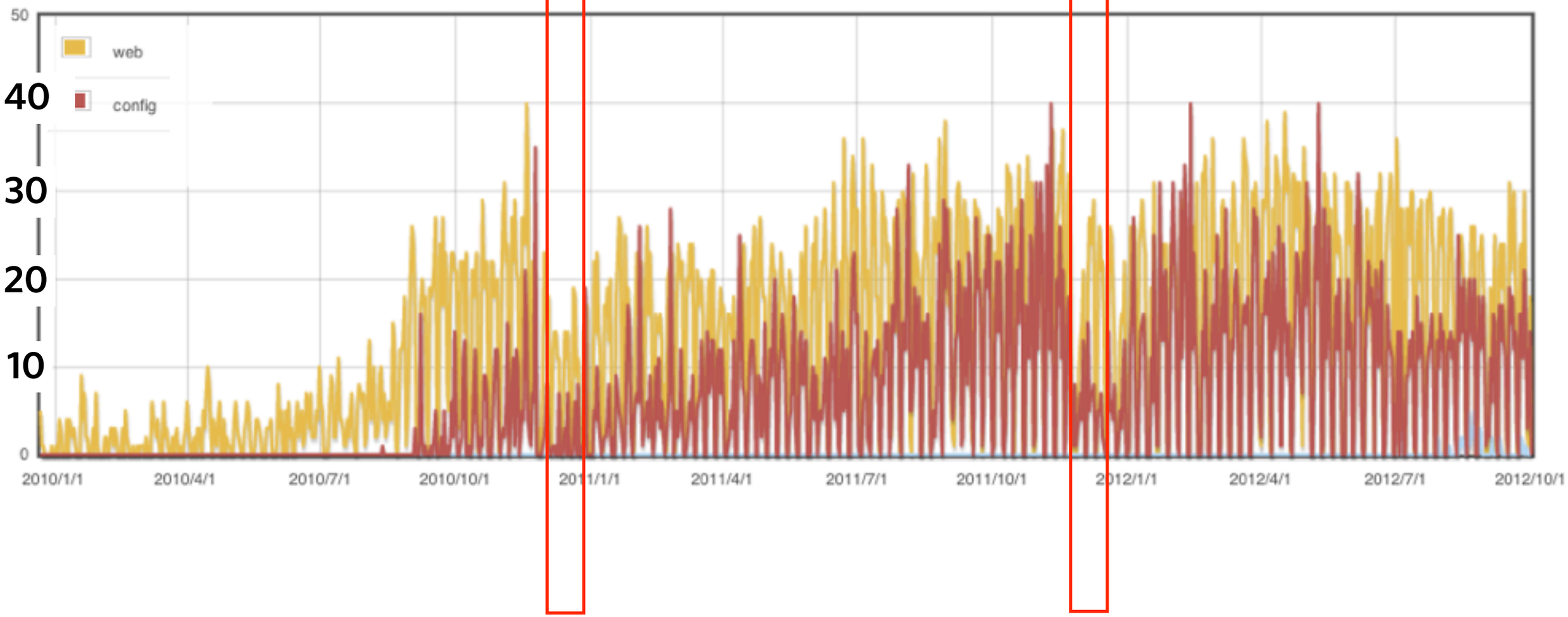
Nope. It's really broken.

http://www.flickr.com/photos/flyforfun/2694158656/

Thursday, Nov 22 - Thanksgiving
Friday, Nov 23 - “Black Friday”
Monday, Nov 26 - “Cyber Monday”

~30 days out from Christmas

Deployments Per Day (US/Eastern)



Thank you.

Mike Brittain

mike@etsy.com
@mikebrittain