



# Looking Under the Hood of New hm Deploys



A technical talk // from michael

2022-05-19

# the plan

- deploy, what is it
- deploy, the before times
- deploy, the in-between times
- deploy, the current world
- and yes, some git arcana 🌟

deploy, what is it

# deploy, what is it

- robn, "How does deployment?" (2020-02-07, TFCon #4)
- much of this is still true and relevant!
- (except for the parts I rewrote entirely)

# rob says

- deployment is two parts
  - get the change\* onto servers
  - activate the changes
- this is all (mostly) still true!

rob says

uncomfortable  
truths

# uncomfortable truths

- deploy is by branch, not by MR
  - possible to accidentally deploy things
- extremely difficult to work out how to make your change live
  - mostly you just have to know for whatever thing you're changing
- there's few protections in the system

~~uncomfortable truths~~

solved\* problems 🎉

# rob says

- rolloutsa
- rolloutdns
- FailOverCyrus.pl
  - I *also* rewrote this entirely since rob's talk
  - ...but I'm not gonna talk about it now

# refreshers and reminders

- every machine has a clone of hm.git
- every machine has a "home branch"
  - managed by branchio
- deploy is effectively `git pull` × 108 (🤔)

# refreshers and reminders

- activating your changes
  - do nothing
  - install some config
  - restart services
  - recompile something
  - all of the above [sic]

deploy, the before times

# deploy, the ancient times

- bort's birthday, 2015-06-03
- v1 already knew how to rollout

10:14 **bort** BOT

**rollout finished**

rollout fastmailbeta finished, 4a22e2a -> 95107e7

10:14 ☆ **trob** BOT **bort: www : Tests started**

10:14 **bort** BOT trob: 🤔 I can't!

10:14 **trob** BOT bort: 🤔 I could try, but it's probably not going to work.

10:14 **bort** BOT trob: 🤔 Sorry, what's that?

10:14 **trob** BOT bort: 🤔 No, you do it.

10:14 **bort** BOT trob: 🤔 I got nothing.

10:14 **trob** BOT bort: 🤔 Nope.

10:14 **bort** BOT trob: 🤔 I got nothing.

10:14 **trob** BOT bort: 🤔 Yeah, nah.

10:14 **bort** BOT trob: 🤔 Hmm?

10:14 **trob** BOT bort: 🤔 It's too hard!

10:14 **bort** BOT trob: 🤔 No, you do it.

10:14 **trob** BOT bort: 🤔 But it's cold outside and I'm frightened!

10:14 **bort** BOT trob: 🤔 Nope.

10:14 **trob** BOT bort: 🤔 Wat?

10:14 **bort** BOT trob: 🤔 Yeah, nah.

10:14 **trob** BOT bort: 🤔 No, you do it.

10:14 **bort** BOT trob: 🤔 I can't!

10:14 **trob** BOT bort: 🤔 I got nothing.

10:14 **bort** BOT trob: 🤔 Yeah, nah.

10:14 **trob** BOT bort: 🤔 I can't!



bort: Result for cyrus : ✅ SUCCESS [Link to results](#) Tests started 737 minutes ago.

10:14 **bort** BOT trob: 🤔 But it's cold outside and I'm frightened!

# deploy, the ancient times



# deploy, the before times

- bort deploy fastmail/master /r hm!4805 a=woods r=cmorgan
- 
- bort rollout fastmail
- 



**bort** APP 10:41

### deploy finished

[2038229] I: DEPLOY of fastmail/beta: root - Deploy by woods (reason: [none provided]); No approval required

[2038229] I: deploying

fastmail/beta@58dd125e80659115a74cb89935532a87dfac4fc6

[2038229] I: successful deployment to 3 hosts

[2038229] betautility1 betaweb1 betaweb2

### rollout finished

[3043746] E: fastmail/beta is at c765e1d509180541aa9e1a80d138da291c0cc2e1, but last deploy was 58dd125e80659115a74cb89935532a87dfac4fc6, won't build

### deploy finished

[2038701] I: DEPLOY of fastmail/master: root - deploy by woods (reason: hm!4805 a=woods r=cmorgan); Approved by michael

[2038701] I: deploying

fastmail/master@0c0bfeb2931b96edcd8a033db66a4cbcde3d54ff

[2038701] W: Awaiting response from...

[2038701] imap35

[2038701] for 30.00.

[Show more](#)



**rollout** APP 10:49

fastmail rolled out, 61157d6 => 0c0bfeb ([compare](#))

### Adds commits

4b86603 Audit logger: include unpending deleted report

868e129 Option to ignore ticket keys when processing email

55a1de8 safe /before/ commit

578c3af Null key warning silence, Pobox category

09e9166 Add pobox processing to helpspot script

da2c053 Fix mobile display issues

98bf51f Change alignment of pricing table

79da654 Signup form tweaks (mostly mobile)

# deploy, the before times

- ci-rebuild-branches
- rebuilt beta/dogfood when any MR was updated
- mostly worked

+ Add a bookmark



GitLab APP 21:39

February 3rd, 2020 ▾



Richard Lovejoy (rjlov)

Pipeline #49933 has failed in 00:22

Branch

[master](#)

Failed stage

[rebuild](#)

hm | Feb 3rd, 2020

Commit

[Merge branch 'avalara-db' into 'master'](#)

Failed job

[rebuild-branches](#)



Fastmail (fm)

Pipeline #49934 has failed in 00:19

Branch

[master](#)

Failed stage

[rebuild](#)

hm | Feb 3rd, 2020

Commit

[Merge branch 'avalara-db' into 'master'](#)

Failed job

[rebuild-branches](#)



Fastmail (fm)

Pipeline #49935 has failed in 00:19

Branch

[master](#)

Failed stage

[rebuild](#)

hm | Feb 3rd, 2020

Commit

[Merge branch 'avalara-db' into 'master'](#)

Failed job

[rebuild-branches](#)



GitLab APP 21:58



Fastmail (fm)

Pipeline #49936 has failed in 00:16

Branch

[master](#)

Failed stage

[rebuild](#)

hm | Feb 3rd, 2020

Commit

[Merge branch 'avalara-db' into 'master'](#)

Failed job

[rebuild-branches](#)



Fastmail (fm)

Pipeline #49937 has failed in 00:15

Branch

[master](#)

Failed stage

[rebuild](#)

hm | Feb 3rd, 2020

Commit

[Merge branch 'avalara-db' into 'master'](#)

Failed job

[rebuild-branches](#)

# deploy, the before times

- goofy problems:
- beta didn't contain your just-pushed changes
- beta rebuild between deploy and rollout
- E: fastmail/beta is at 2291eb5a, but last deploy was 159de375, won't build

deploy, the in-between times



deploy, the in-between times



# deploy, the in-between times

- mint-tag introduced on an unsuspecting world (2020-05-22)
- bort's brain transplant (2020-06-08)
- bort learns `deploy /rollout` (2020-06-18)
- death to `ci-rebuild-branches` (2020-07-16)

# deploy, the in-between times

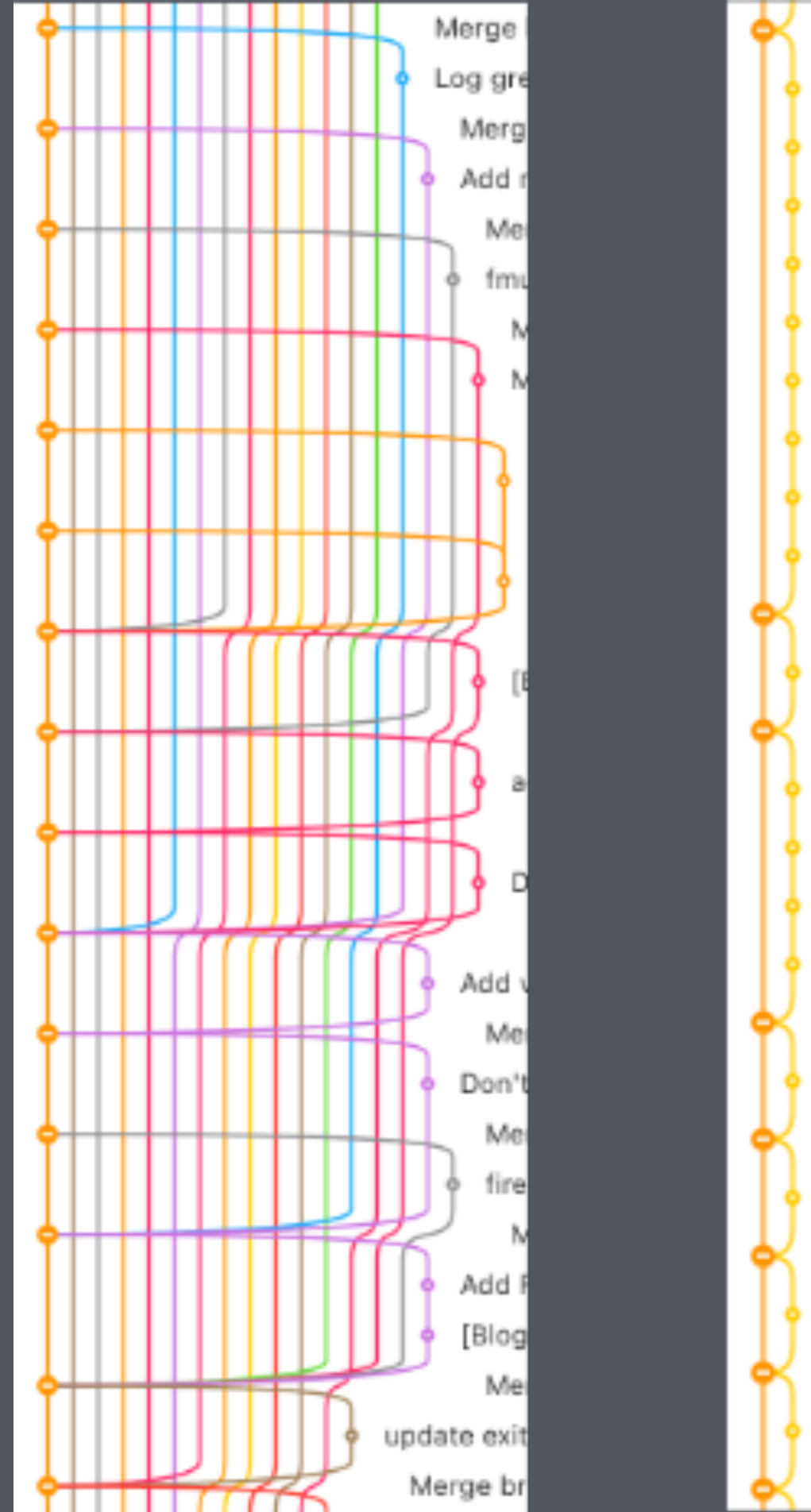
- mint-tag is great
-  Minting Tags and Git Arcana  (TFCon, 2020-08-06)
- totally eliminated some classes of errors

why bother?



# why bother?

- 2021-07-22: 🚀 Moving to GitHub has launched!
- 2021-08-11: Michael says "Wellll...."
- 2021-09-13: 🚀 hm.git Deployment Updates



a brief aside on git history

# a brief aside on git history

- I care a *lot* about git history
- and I suggest that maybe you should too
- and that you should start by writing better commit messages

# good commit messages

- Victoria Dye - "Writing Commits For You, Your Friends, And Your Future Self" ([on YouTube](#))
- emails to your future self
- should contain
  - *what* - high-level intent of commit
  - *why* - context for the implementation

fmvars: remove host partition counts

Long ago, these were important because it wasn't uncommon for a disk controller to fail and a volume just disappear, but also, it was possible for us to start up and find that something hadn't been mounted, and we didn't notice until the root disk failed.

These days that's basically a non-issue. Linux doesn't routinely drop volumes, even if they're dead underneath (other stuff fails, but they don't disappear). We have checks in Cyrus startup to prevent start if the underlying partitions aren't there, and with more stuff moving to ZFS these checks don't even make sense more and more of the time.

But! When we do something with disks, adding or removing mounts, even temporarily, this will start sending email. And we sigh, and adjust the number in fmvars just to shut it up and nothing more.

It's useless. Lets kill it.

5 files changed, 31 insertions(+), 99 deletions(-)

Initial JMAP endpoint

213 files changed, 14903 insertions(+), 4943 deletions(-)

mint-tag: do not use semilinear merge on QA

Right now, if you deploy two MRs at the same time to QA, you can wind up in a potentially dangerous situation, because we were using semilinear merges there.

Imagine a situation like this:

- We have two branches; branch X (MR #123) and branch Y (MR #456)
- Someone asks bort to deploy MRs 123 and 456 to QA
- mint-tag fetches the MRs, then rebases+merges X on master, then rebases Y on top of that (master + X) and merges it in.
- mint-tag then force-pushes the new head of X back to 123, and the new head of Y back to 456

The last step here is wrong, because branch Y will also include the commits from X!

This is acceptable and is what we want for master deploys, because "rebase Y on master + X" is safe because we've already committed to merging X into master. In the QA case, though, it's not safe, because then it's possible that someone could say "oh, Y is good but X is not" and decide to deploy Y to master, unknowingly deploying the changes from X, which are now in the branch for Y!

We can fix this by just turning off semilinear merges for QA, and turning on rebases instead. With this mint-tag config, the situation above turns into:

- Someone asks bort to deploy MRs 123 and 456 to QA
- mint-tag does the fetches, then rebases both X and Y on current master, and does an octopus merge of the two of them to form a QA branch. (This is exactly what we do for dogfood and beta, with the addition of a rebase on master.)
- mint-tag then force-pushes the new head of X back to 123, and the new head of Y back to 456. This is fine, because all we've done is a rebase on current master; it's as though you'd just clicked the Rebase button in the GitLab UI.

1 file changed, 1 insertion(+), 1 deletion(-)

anyway, deploy

# why bother?



**bort** APP 18:30



I fainted during deploy **20211206.022** (started by neilj in #plumbing, 11 minutes and 4 seconds ago), so I'm not sure what state that's in. Sorry!

# why bother?

- deploy state was in memory in bort
- bort can reboot himself when he can't get a connection to Slack
- you had to log-dive to see what was going on
- could be dangerous! after disconnect, we dropped the lock

# why bother

# deploys – Dec 13th, 2021



**bort** APP 13:59



rollout 20211213.020 (fastmail) finished: rollout succeeded; all done!

started by woods in #plumbing, 24 minutes and 39 seconds ago

deploy, the current world

# deploy, the current world

- deploys are now run via the job queue
- logic now lives in ME::Deploy, not in bort
- rollouts are now both smarter and (consequently) faster

# deploys via job queue

- we have a job queue (landed April 2019)
- used for migrations, fixaccount, sending welcome messages
- and now, deploys

# deploys via job queue

- job queue got some improvements along the way
- it now restarts safely
- and runs in more places (webs, build1)
- these are plumbing details, not interesting for this
- ...but the commit messages tell the story 🤗

# deploys via job queue

```
sub execute ($self) {  
    my $deploy = ME::Deploy->retrieve($self->deploy_id);  
  
    eval { $deploy->execute };  
    if ($@) {  
        $Logger->log([ "deploy failed: %s", $@ ]);  
        return $self->cancel;  
    }  
  
    $self->mark_done;  
}
```

```
sub execute ($self) {  
    if ($self->is_done) {  
        $Logger->log_fatal([ "cannot re-execute completed deploy (%s)", $self->id ]);  
    }  
  
    try {  
        $self->prepare_branch;  
        $self->deploy_branch;  
        $self->do_rollout;  
    } catch {  
        my $err = $_;  
        $self->_handle_exception($err);  
    };  
  
    return $self->mark_done;  
}
```

# deploys via job queue

- bort can now recover from a slack reconnection
- via a convoluted process involving Consul, IO::Async, manual file descriptor handling, and more
- this is very interesting, but I will mostly not talk about it here

=head1 WHAT EVEN IS THIS NONSENSE

The purpose of this class is to run in a forked process and listen for consul events. It needs to run in a subprocess because the consul event API works by making blocking HTTP calls: i.e., to listen for an event, you make an HTTP call with a timeout, and it returns when either there an event arrives or until the timeout has expired. We could use an async consul, which would run a callback on the completion of the HTTP call, but really we need to do this in a loop, which means we'd wind up in callback hell pretty quickly.

But there is another wrinkle: because this code is running in a subprocess, we've closed down all the IO handles before fork, including notably, our connection to Slack. That means we can't just call methods on the slack reactor, because they'll never get sent. So instead, our parent sets up a pipe and passes a file descriptor to us, which we open here. Data we want to send to slack gets JSON-encoded and piped back to our parent via `->write_handle` here, which our parent decodes and dispatches.

The remaining fiddliness of this code is because consul events are not guaranteed to arrive in any particular order (or at all), so we need to be resilient and be able to cope with that. This has a couple of knock-on effects, namely that we poll more often than we should really ever need to, and also that we have to do a bunch of housekeeping to make sure the instigating event has the correct `reactji` at all times.

# deploys via job queue

- ME::Deploy is mostly coordination logic
- under the hood, it runs the same thing it always has

# deploys via job queue

- ME::Deploy is mostly coordination logic
- under the hood, it runs the same thing it always has\*

```
$self->prepare_branch;    # runs mint-tag  
$self->deploy_branch;     # runs fm-deployo  
$self->do_rollout;        # runs Rollout.pl
```

# mint-tag

# mint-tag

- builds branches from labeled merge requests
- gained some new features as part of this work
- notably: how to do semi-linear merges
- I'll spare you the details (it's all in the commit messages)

# mint-tag

- ME::Deploy runs:

```
$ mint-release-branch -c /etc/minttag/master.toml --mr 10682 --mr 10713
```

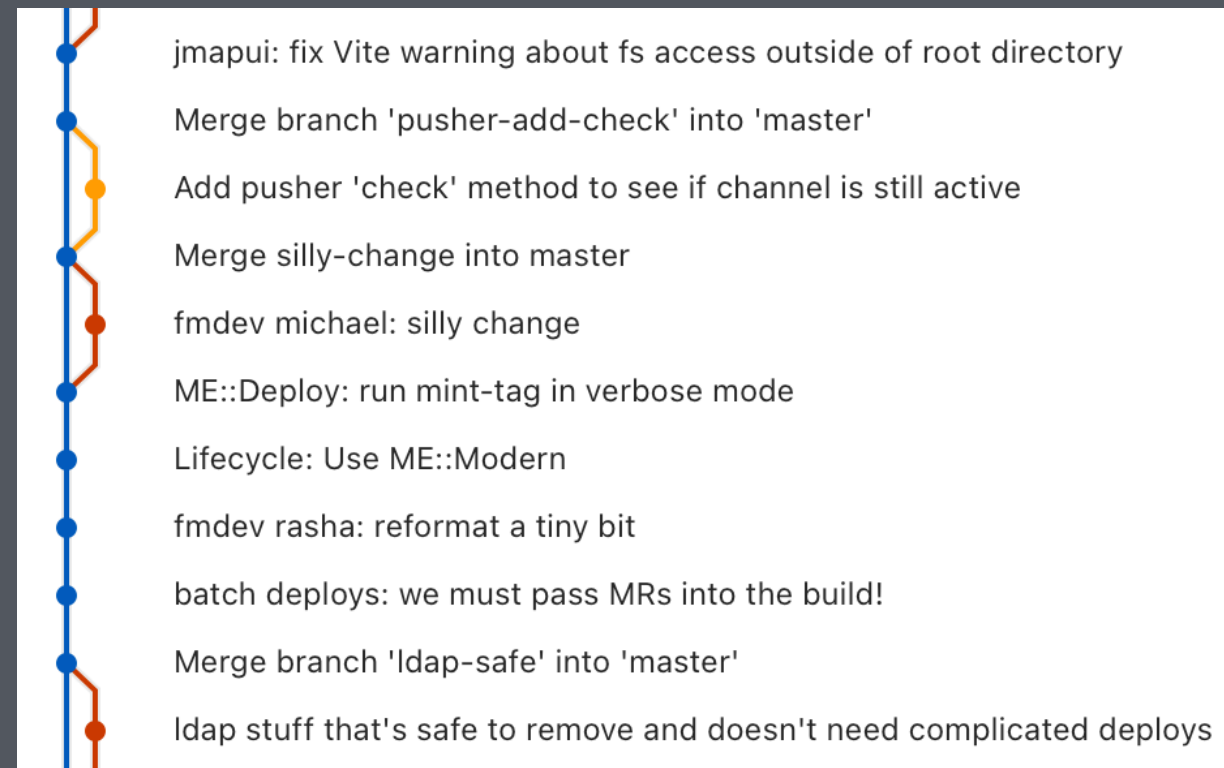
- which, for every MR:
  - fetches, rebases, merges it, then force-pushes it back to branch
- then pushes to master

# mint-tag

- this all works a treat
- except for...



# mint-tag

- this all works a treat
- except for...



a git bug! 🐛

a git bug! 

- which I am going to tell you about
- you knew what this was 
- sorry not-sorry 

# a git bug! 🐛

- mint-tag operates with shas directly, not with ref names
  - i.e. it says 94a46e732e8bf1cea62 instead of michael/movemate
- git rebase main topic is shorthand for git checkout topic; git rebase main
- after this, you're on branch topic, freshly rebased
- mint-tag uses this shorthand

# a git bug!

- `$ git rebase main ba4a4a5`
- what should happen
  - `git checkout ba4a4a5; git rebase main`
  - you're left with detached HEAD, with topic freshly rebased

# a git bug!

- this was *broken* in git!
  - if ba4a4a5 needed rebasing, everything was fine
  - if ba4a4a5 was a fast-forward operation (i.e., was already up to date), main was *fast-forwarded*
- when mint-tag said `checkout main && merge --no-ff ba4a4a5`, main had moved out from under it!

# a git bug! 🐛

- fixed in git v2.36

```
[~/code/src/git] $ git log --oneline --grep McClimon  
bdff97a3 rebase: set REF_HEAD_DETACH in  
checkout_up_to_date()
```

# achievement unlocked

The latest feature release Git v2.36.0 is now available at the usual places. It is comprised of 717 non-merge commits since v2.35.0, contributed by 96 people, 26 of which are new faces [\*].

New contributors whose contributions weren't in v2.35.0 are as follows. Welcome to the Git development community!

..., Michael McClimon, ...

what was I talking about again?


# deploys via job queue


- ME::Deploy is mostly coordination logic
- under the hood, it runs the same thing it always has\*

```
$self->prepare_branch;    # runs mint-tag  
$self->deploy_branch;     # runs fm-deployo  
$self->do_rollout;        # runs Rollout.pl
```

# rollout improvements

- oh yeah I rewrote rollout entirely\*
- it's way faster than it was

 **bort** APP 14:38  
✅ rollout 20220511.029 (fastmail) finished: rollout succeeded; all done!  
started by woods in #plumbing, 6 minutes and 33 seconds ago

 **bort** APP 14:57  
✅ rollout 20220511.030 (fastmail) finished: rollout succeeded; all done!  
started by rasha in #plumbing, 2 minutes and 54 seconds ago

# rollout, before

- BuildHtdocs.pl
- rebuild the frontend (zolasite, jmapui, l10n, etc.)
- rsync them to the mirrors
- rsync from there to the frontends
- on each web, stop apache, rsync the frontend in, start apache

# rollout, now

```
my $rollout = ME::Rollout->new({  
    target          => $target,  
    verbose         => $opt->verbose,  
    should_sync     => $opt->sync,  
    defined_kv(restart_apache => $opt->restart_apache),  
    defined_kv(rebuild_js    => $opt->rebuild_js),  
});  
  
$rollout->execute;
```

# rollout, now

- still does all the same things
- ...but only if it needs to

```
sub execute ($self) {  
    $self->establish_lock;           # 1  
    $self->prepare_build_dir;        # 2  
  
    $self->build_l10n_files;          # 3  
    $self->build_htdocs;              # 4  
  
    return unless $self->should_sync;  
  
    $self->sync_to_mirror;             # 5  
    $self->sync_to_frontends;          # 6  
    $self->sync_to_webs;               # 7  
  
    my $rev_after = $self->store_rollout_sha_in_consul;  
    $self->finalize($self->previous_rollout_sha, $rev_after);  
}
```

# rollout, now: setup

- 1) we establish a lock, to make sure you can't rollout twice at the same time
- 2) update our working dir (on build1) to the right commit
  - by this point, mint-tag has pushed the branch and the deploy has happened

# rollout, now: build frontend

- 3) build the localization files
  - is this even still necessary? (Probably, for now)
- 4) build the necessary JS and HTML
  - this just calls make to do the work

# rollout, now: conditional builds

```
sub build_l10n_files ($self) {  
    return unless $self->should_build_js;  
    ...  
}
```

```
sub build_htdocs ($self) {  
    return unless $self->should_build_js;  
    ...  
}
```

# rollout, now: conditional builds

```
has should_build_js => (  
  is => 'ro',  
  init_arg => 'rebuild_js',  
  lazy => 1,  
  default => sub ($self) {  
    my $files = $self->files_touched_since_last_rollout;  
  
    return 1 unless $files; # dunno what changed? default to build  
  
    return any {; /^(htdocs|localisation)/n } @$files;  
  },  
);
```

# rollout, now: conditional builds

- `files_touched_since_last_rollout`
- we store rollout shas in consul, so we know what has changed
- and we can take action on it!
  - if we haven't touched JS, don't rebuild JS
  - if we haven't touched perl, don't restart apache
  - maybe later, other smarts

# rollout, now: sync to and fro

- we still must sync to mirror (5) and then out to the frontends (6; for static assets) and backends (7; needed by JMAPApp)
- syncing frontends is now much faster
  - we now do it from the local mirror (thanks Joe!)
  - we now do it in parallel

# rollout, now: no-restart JS changes

- JMAPApp no longer needs restart for JS changes
- Rik and I took a pass at this in July '21 and failed
- now it works

# rollout, now: no-restart JS changes

- there are two files, bootstrap.html and bootstrap.hash
- before, the JS build copied them into the root of jmapui/
- which required a restart of apache to pick them up
  - because rsync might copy the bootstrap files before everything else was ready

# rollout, now: no-restart JS changes

- now, JS build leaves them inside the build directory (not in the root)
- and rollout symlinks them into place after the rsync
- JMAPApp watches for the inode to change and reloads the bootstrap files
- this is a *big win* for speed

# rollout, now: no-JS apache restarts

- the opposite way is faster too!
- if the JS files haven't changed, we don't need to re-bundle the frontend!
- so we can skip the syncing entirely and just do the restarts
- this is also a big win for speed!

# rollout, now:

- it's fast
- it's maintainable (BuildHtdocs.pl had effectively no subroutines)
- it does what you want more of the time

# other new toys

- deploys can now be manually locked
- `deploy fastmail/all`
- `deploy /mr 123`

deploy fastmail/all

**LP 62992101**

**bort: add deploy /all**

*created by robn on 5/17/2021, 9:05:43 PM*

Never marked as done

# deploy fastmail/all

- made possible by moving deploys into job queue
- implemented by ME::Deploy::Batch

# deploy fastmail/all

- when you deploy `fastmail/all /mr 123`,
- bort takes a lock and inserts a batch-deploy into the job queue
- when that runs, it creates 4 more jobs
- which all run in sequence
- and bort watches the batch to wait for its completion
- simple matter of programming!

# deploy /mr 123

- one of the uncomfortable truths; deploy is unsafe!
- you want to deploy your trivial MR, !123 to all four environments
- you rebase it and click the merge button in GitLab
- before you deploy master, someone merges !125
- you deploy master for !123, unknowingly sending !125 along for the ride !

# deploy /mr 123

- the biggest change, and the most important
- with fastmail/all, *significantly* increases safety of deploys
- you can now do a final test on QA *without* merging to master
- prepares the way for an eventual move to GitHub
- don't worry: robn has already found new things to ask me for

# in conclusion

- write good commit messages
- test your changes
- get your changes onto servers
- make your changes actually go
- it's safe and fast and good

