Feature Toggles and Graphs

Tim Retout
London Perl Workshop, 2014-11-08
Who am I?

- Tim Retout <tim@retout.co.uk>
- Development Manager at CV-Library
- CV-Library has lots of users, and releases lots of features
- Yes, we’re hiring
tl;dr: Improve your release process

Releasing features to all users at once is risky.

Do it a bit at a time; and get feedback.
1. Feature Toggles

Do it a bit at a time...
Feature Toggles

- a.k.a. flippers, switches, etc.
- Lots of big tech companies use them
- Make a decision at runtime about whether to enable a feature - and don’t embed the decision in the code
- Talk to a database (e.g. Redis) which stores information about when to turn on a feature
Downsides of Feature Toggles

- Some more complexity
- Code duplication if comparing old/new code
- Testing impact - combinatorial explosion
- Runtime performance impact (be careful)
- Must remember to go in and clean up old code once toggle is no longer needed
Upsides of Feature Toggles

● If used correctly, we can reduce deployment risks:
  ○ Avoid complex merges
  ○ Roll out code to a percentage of users
  ○ Performance testing on real hardware
  ○ Quick rollback if there are problems

● More flexible than alternatives involving releasing to a percentage of machines
Toggle (on CPAN)

● Written and used at CV-Library
● Stolen/Borrowed from Ruby (rollout)
In your initialization code

Give Toggle a data store:

my $redis = Redis->new();
my $toggle = Toggle->new( store => $redis );

(You do use dependency injection, right?)
Data store

Implement key/value API like Redis. Stored internally as percentage/user/group:

```
chat => "10|tim,bob|staff"
```

Looks a bit odd, but requires only one row lookup per feature, which keeps it fast.
UI for toggling

Currently no pretty UI, although there's a Ruby one available for porting.

Toggle implements the required methods for making a user interface, however.
Rollout UI
Around your feature code

if ( $toggle->is_enabled('chat') ) {
    # Code for cool new chat feature
}

This is probably not useful enough - it gets more interesting if you can toggle per user.
if ( $toggle->is_enabled('chat', $user) ) {
    # Code for cool new chat feature
}

Note that no decision about $user is made here - just use the result from Toggle.
Use Case: staff testing

```perl
$toggle->add_group( staff => sub {
    my $user = shift;
    return $user->admin == 1;
});
```

Then add "staff" group in DB.
Use Case: incremental rollout

- Enable for 1% of users in DB, then increase percentage as you gain confidence.
- Uses a hash of $user->id to control percentage; once a user gets a feature, they keep the feature.
Use Case: quick rollback

- Just set percentage to 0 in DB.
- Takes effect instantly; no deployment required.
Use Case: “Labs” experiments

- No reason to use Toggle here, unless you are planning to release to all users later?
- But if you do, create a group as before, and check whether user has opted in.
- Toggle supports adding user ids directly in its store, but this won't scale well.
2. Graphs

...and get feedback
Graphs

“You need more monitoring”
-- Me, repeatedly, since 2010 or so

- Near-realtime
- 10s resolution
- Application-level and server-level
Statsd and Graphite

- Add code in your application to send UDP packets to statsd
- Statsd aggregates packets and updates Graphite (over TCP) every 10s
- Graphite lets you easily graph all the data
Using statsd

- Grab a CPAN module like Net::Statsd
- Sprinkle counters and timing all over code
- Suddenly you can see what’s happening on your platform!

```perl
# generate name at runtime
inc("foo.bar.$baz");
```
Delete cron script that restarted memcached every 2 hours

Vastly speed up template processing

Homepage median server response time
What's this? No idea, but at least we know it happened!
job view - 90th percentile load time

Looks even worse at 90th percentile
Median is under 150ms
Same time period has spikes at 90th percentile
job view - with deployment times
Use Case: A/B testing

- Toggle 0.02 added variant support
- Plot relevant graphs for A/B
Use Case: performance testing

- Difficult to get realistic performance numbers until running on production
- Put a small percentage of users through new code path, and measure impact
Use Case: degrade under load

- Measure error rate from non-critical services
- If rate exceeds a threshold, programmatically disable that feature

https://github.com/jamesgolick/degrade
Use Case: dat-science

- Rewriting a code path?
- For a small percentage of users, try both paths and throw the new one away
- Graph timings and count result differences

https://github.com/github/dat-science
Conclusions

Think about risks; work to reduce risks in your development processes.

Graphs are cool.

Everyone should use Toggle.
Questions?

- Do you already do something better? :)
- Alternatively, chat to me at one of the coffee breaks, or ask via email.
- Source is on CPAN and Github.