Talisman.
We lower the cost of recruitment through technology

Modernising Legacy Perl

For Commercial Financial/Recruitment Company Software

Peter Edwards, Talisman Technology
https://www.talisman.tech/
https://linkedin.com/in/peteredwards/
http://dragonstaff.co.uk/perl/

PerlKohaCon, Helsinki Aug 15th 2023

Peter - Introduction

Speaking at first Kiev Perl Mova Conference in 2008

Been using perl since it was first released, and using Unix since 1984. Worked in Commercial, Education, Media, International Public Relations:

Open University, BBC Future Media and World Service, British Council.

Procured Koha LMS for British Council library, Lahore, Pakistan, 2016.

British Council, Beijing 2013

Spent a lot of time at Talisman Tech writing perl CGI web programs in early 2000s.

Rejoined in 2017 as CTO to modernise the company, codebase and products.

Talk - What we'll cover

We'll look at crufty* old perl, and see how it can be modernised
in a real world case.

How can I go from apache 1/mod_perl 1 web server with CGI and perl 5.10.1

to Plack with PSGI and perl 5.36 and a modern Web site service environment?

It can be done!

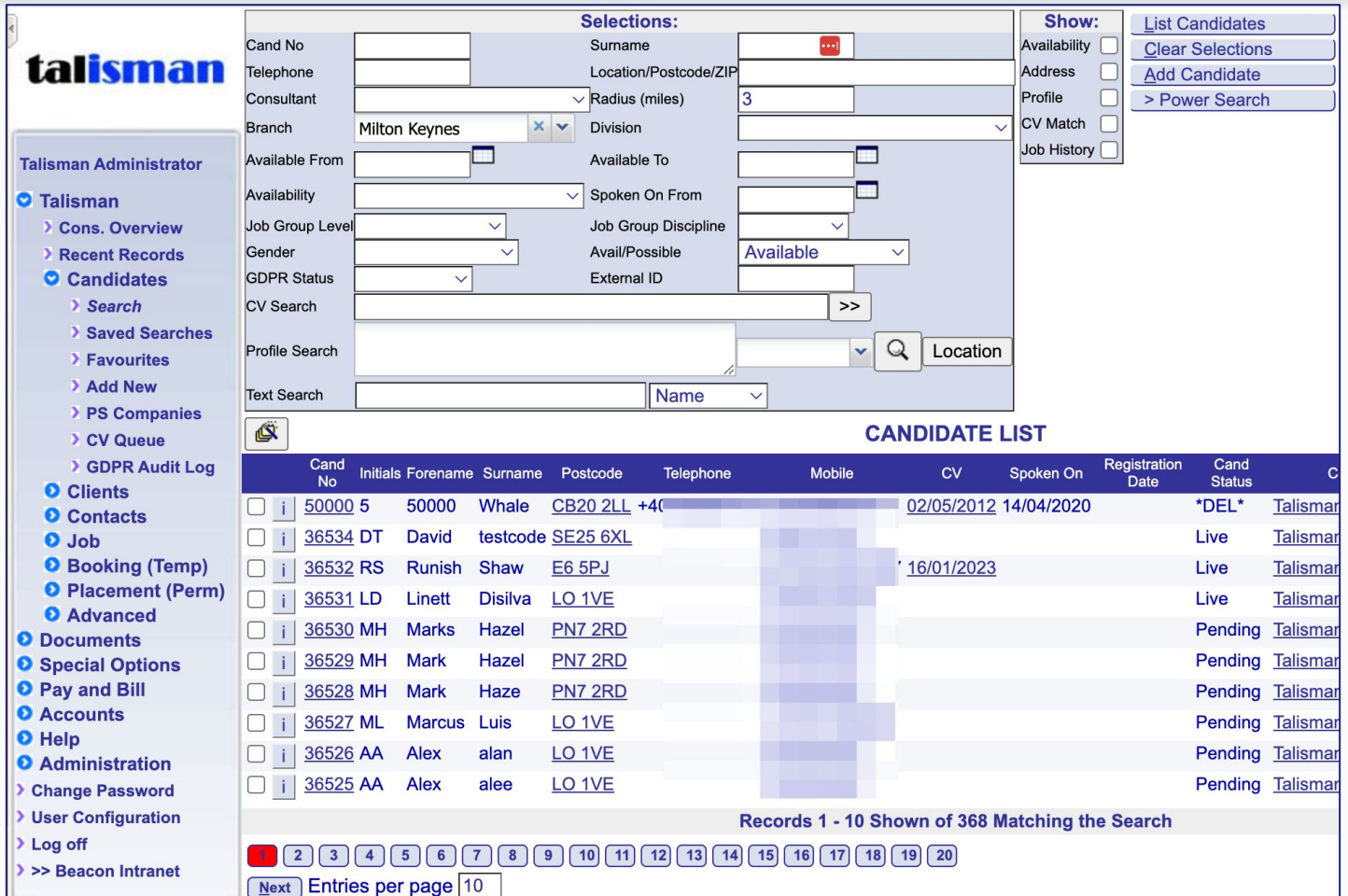
* Cruft is a jargon word for anything that is left over, redundant and getting in the way. It is used particularly for defective, superseded, useless, superfluous, or dysfunctional elements in computer software. https://en.wikipedia.org/wiki/Cruft

What have we got in 2019?

RS-232 plug box for wiring VDU green screen terminals to Unix servers. Finally removed from our machine room last week (2023-08-06)

Talisman Web
Recruitment CRM/Pay and Bill/Accounts

- Red Hat Enterprise Linux 7
- Round trip HTML 4. Post a form, whole page of HTML comes back.
- CGI protocol from apache web server
- Frameset and frames (!)
- Apache 2 system perl 5.16
- Apache 1 mod_perl 5.10.1



2019

- Proxy via Apache 2 front to Apache 1 mod_perl backend perl 5.10.1
- Written with perl 5.005 then perl 5.008, and MySQL 3 database.
- Mixture of legacy perl, some okay with ‘use warnings’, some written like C, or BASIC and not. Some “green screen” 4GL code and C-ISAM database.
- A decent business database object layer. We can build on that.
- Some badly behaved scripts have to run in perl exec CGI mode so Perl exit() does the garbage collection :-(Runs from Apache 2
- Majority of code runs under mod_perl1 compiled with apache1 web server.
- On-premise servers

2019

- Proxy via Apache 2 front to Apache 1 mod_perl backend perl 5.10.1
- Written with perl 5.005 then perl 5.008, and MySQL 3 database.
- Mixture of legacy perl, some okay with ‘use warnings’, some written like C, or BASIC and not. Some “green screen” 4GL code and C-ISAM database.
- A decent business database object layer. We can build on that.
- Some badly behaved scripts have to run in perl exec CGI mode so Perl exit() does the garbage collection :-(Runs from Apache 2
- Majority of code runs under mod_perl1 compiled with apache1 web server.
- On-premise servers

Where would we like to get to?
Customer wants

- Modern design and UX: ReactJS
- Get rid of the old green screen 4GL programs
- Scalable service
- Mixture of on-premise and cloud computing options with integrations to third parties
- Database replication and at rest encryption
- Single Sign On

Where would we like to get to?
Talisman company wants

- Supportable
- Doesn’t break existing production quality service
- Modern architecture:
  ReactJS front end and microservice API backend
- Unit tests, quality tools
- Documentation
- Automated CI/CD
- Dependency analysis, regression control
- Cloud computing - docker, VMs, standard dev environments

Where would we like to get to?
Talisman company wants

- Supportable
- Doesn’t break existing production quality service
- Modern architecture:
  ReactJS front end and microservice API backend
- Unit tests, quality tools
- Documentation
- Automated CI/CD
- Dependency analysis, regression control
- Cloud computing - docker, VMs, standard dev environments

First steps - inventory and standardise

Made an inventory and archive of all software and product assets.

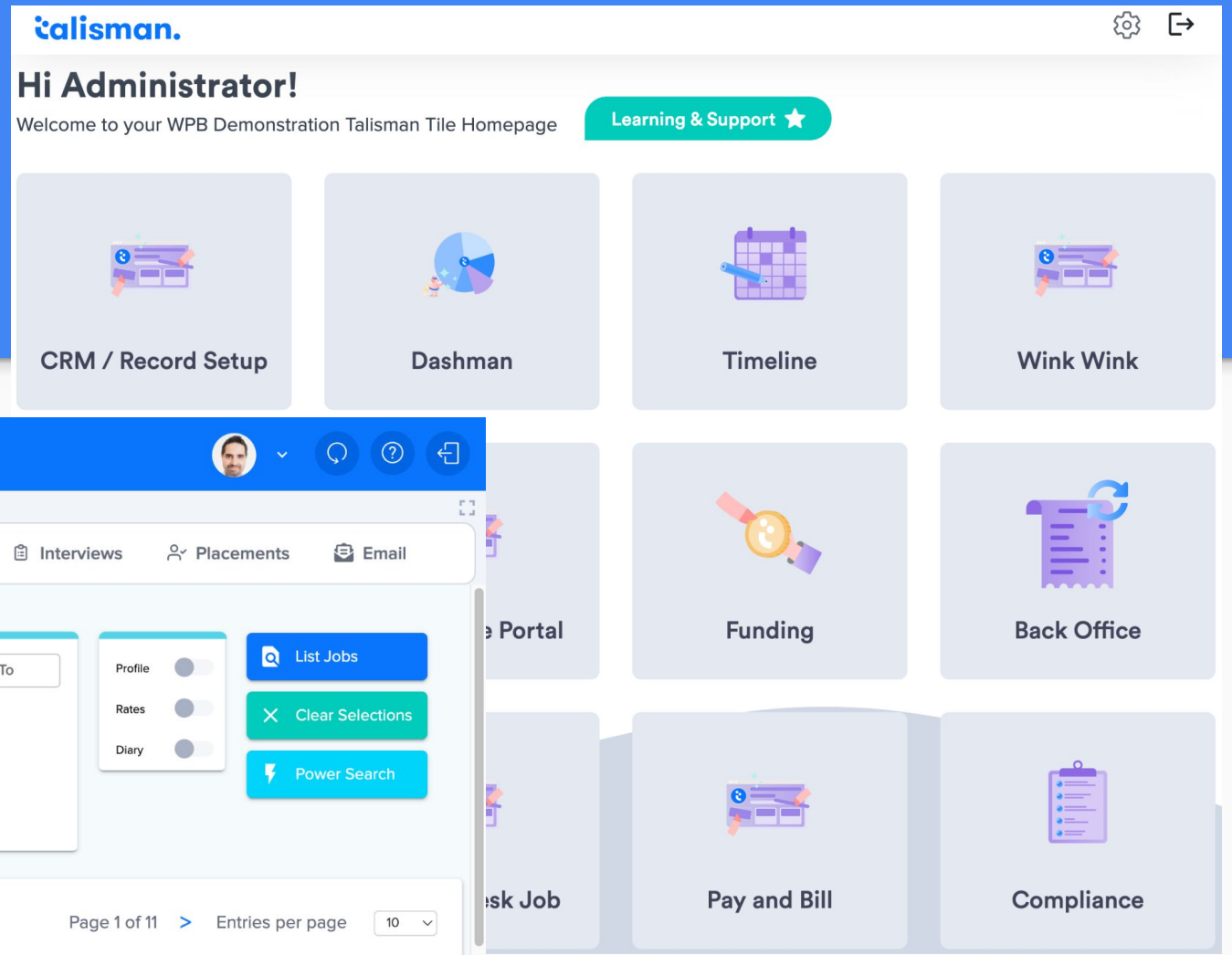
Standardised on RHEL 7.3, MySQL 5.x, perl 5.16.3.

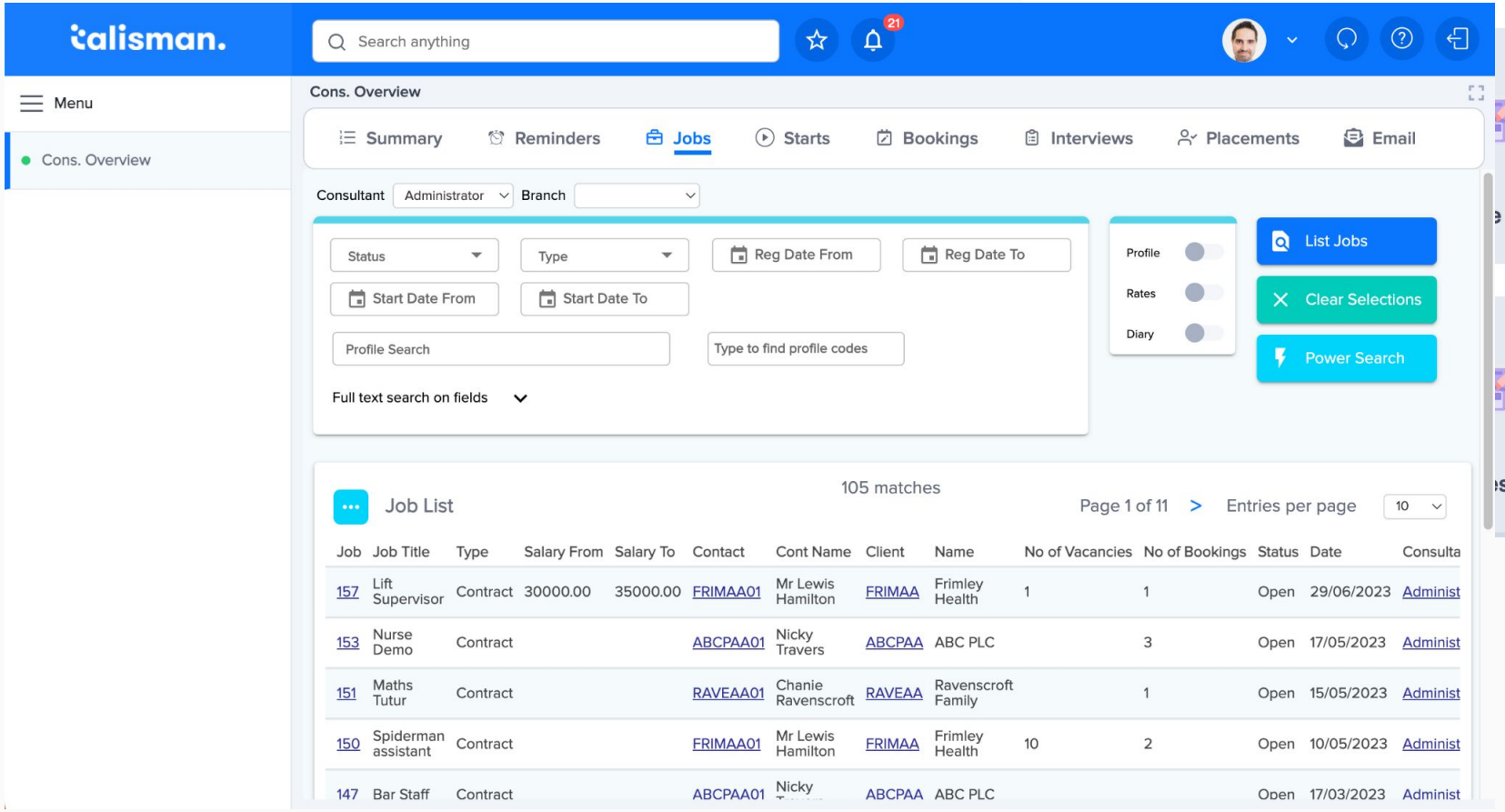
Wrote ansible scripts to automate deployment of standard server.

Used these to create AWS Virtual Machine servers
and a standard virtualbox local development VMs under vagrant.
Dev replicates what is on production.

Introduce Jira ticketing, Slack communication, github.com for code workflow, and Agile processes.

We got there


talisman.
Hi Administrator!
Welcome to your WPB Demonstration Talisman Tile Homepage
Learning & Support
CRM / Record Setup
Dashman
Timeline
Wink Wink
Portal
Funding
Back Office
sk Job
Pay and Bill
Compliance


talisman.
Search anything
Menu
Cons. Overview
Cons. Overview
Summary
Reminders
Jobs
Starts
Bookings
Interviews
Placements
Email
Consultant
Administrator
Branch
Status
Type
Reg Date From
Reg Date To
Start Date From
Start Date To
Profile Search
Type to find profile codes
Full text search on fields
Profile
Rates
Diary
List Jobs
Clear Selections
Power Search
Job List
105 matches
Page 1 of 11
Entries per page
10
Job Job Title Type Salary From Salary To Contact Cont Name Client Name No of Vacancies No of Bookings Status Date Consulta
157 Lift Supervisor Contract 30000.00 35000.00 FRIMAA01 Mr Lewis Hamilton FRIMAA Frimley Health 1 1 Open 29/06/2023 Administ
153 Nurse Demo Contract ABCPAA01 Nicky Travers ABCPAA ABC PLC 3 Open 17/05/2023 Administ
151 Maths Tutur Contract RAVEAA01 Chanie Ravenscroft RAVEAA Ravenscroft Family 1 Open 15/05/2023 Administ
150 Spiderman assistant Contract FRIMAA01 Mr Lewis Hamilton FRIMAA Frimley Health 10 2 Open 10/05/2023 Administ
147 Bar Staff Contract ABCPAA01 Nicky ABCPAA ABC PLC Open 17/03/2023 Administ

How did we get there?

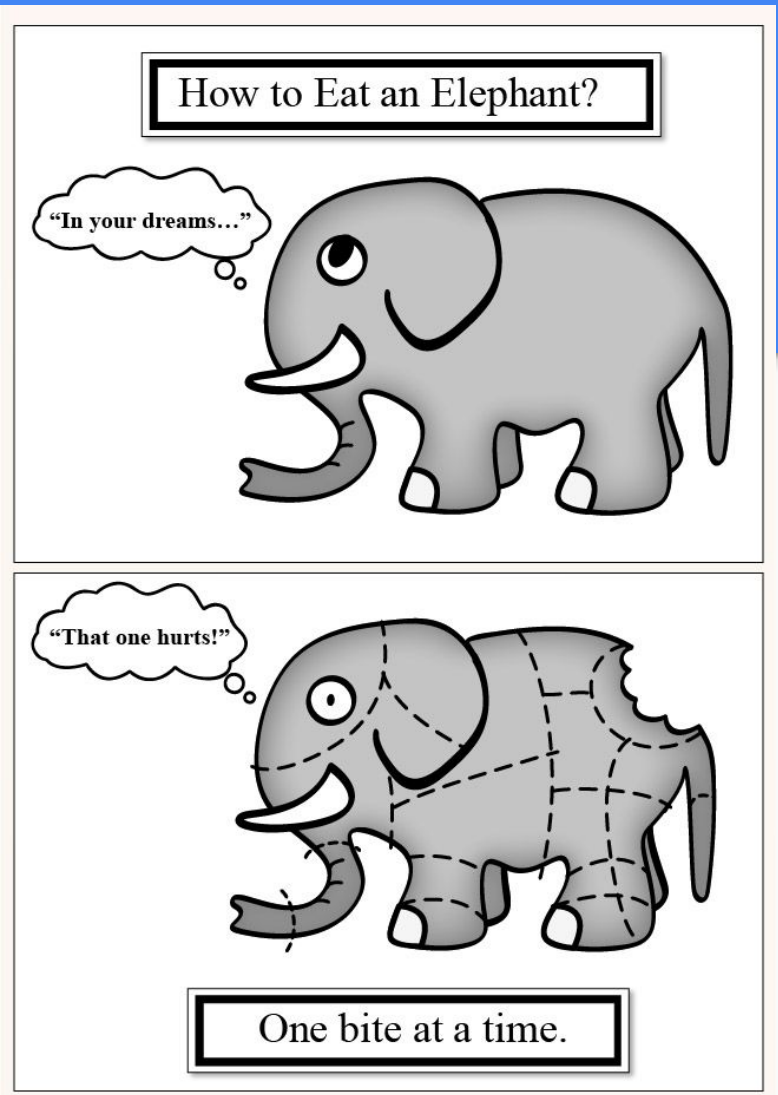
Question: How do I eat an elephant?

Answer: one bite at a time.

But

```
 12:27 am:~/localrepos/tal/cgi $ find . -name \*.cgi | wc -l
     559
 12:27 am:~/localrepos/tal/cgi $ find . -name \*.pl | wc -l
     129
```

Squeak!



Source: https://www.ihavetoomuchstuff.com/blog/secret-decluttering-home/

Naive approach

Do It All In One Go. It'll be easy.

The code will be clean. It will be perfect.

We will have fun. We'll write lots of code!

Definition of 'counsel of perfection'

counsel of perfection

in British English

excellent but unrealizable advice

See full dictionary entry for counsel

What happens

Do It All In One Go.
Takes a large team 3 years to do it all and get it properly stable. Costs a lot.

1. “Rewrite it all in Python. It’ll be easy.”
   And now two tech stacks to support because of existing customers.

2. Rewrite it all in latest framework because it’s great
   [Catalyst, Mojolicious, Dancer, Dancer2, Raisin…]
   You will note from this list that the latest framework will have changed by the time you finally get there… and it still takes a long time for it to be stable.

I have seen multiple projects fail with this approach. “Big Bang” project often == Big Bang

What happens

Do It All In One Go.
Takes a large team 3 years to do it all and get it properly stable. Costs a lot.

1. “Rewrite it all in Python. It’ll be easy.”
   And now two tech stacks to support because of existing customers.

2. Rewrite it all in latest framework because it’s great
   [Catalyst, Mojolicious, Dancer, Dancer2, Raisin…]
   You will note from this list that the latest framework will have changed by the time you finally get there… and it still takes a long time for it to be stable.

I have seen multiple projects fail with this approach. “Big Bang” project often == Big Bang

Gradual approach - expert led

Get an expert in - a few examples, plenty of others! Ask around
Gabor Szabo https://perlmaven.com/ Dave Cross https://davorg.dev/ Andrei Shitov https://andrewshitov.com/ Curtis “Ovid” Poe https://ovid.github.io/ John Napiorkorwski https://www.linkedin.com/in/jnapiorkowski/

Ask them to

- Itemise and assess what you have
- Advise on best new tech to use
- Figure out how to leverage your existing production quality code

What we did

I went via Matt Geppert https://www.linkedin.com/in/mattgeppert/ a UK recruiter I've used often, who runs Open Select Recruitment https://www.osrecruit.com/

And hired Dave Lambley https://www.linkedin.com/in/dave-lambley-992a263/
Dave helped us sort it all out.
Here's one of his talks. Don't know who he's mentioning.
https://archive.fosdem.org/2018/schedule/event/software_necromancy/

If you find yourself in a hole, stop digging

New tech stack:

- Microservice API backends using **Dancer2** https://metacpan.org/pod/dancer2 or **Raisin** https://metacpan.org/pod/Raisin
- New UI using ReactJS and node.js

All major future new development uses new stack. Only **maintenance and small changes** with old stack.

An excavator that is in a hole, and per the Law of Holes, has stopped digging.

https://en.wikipedia.org/wiki/Law_of_holes

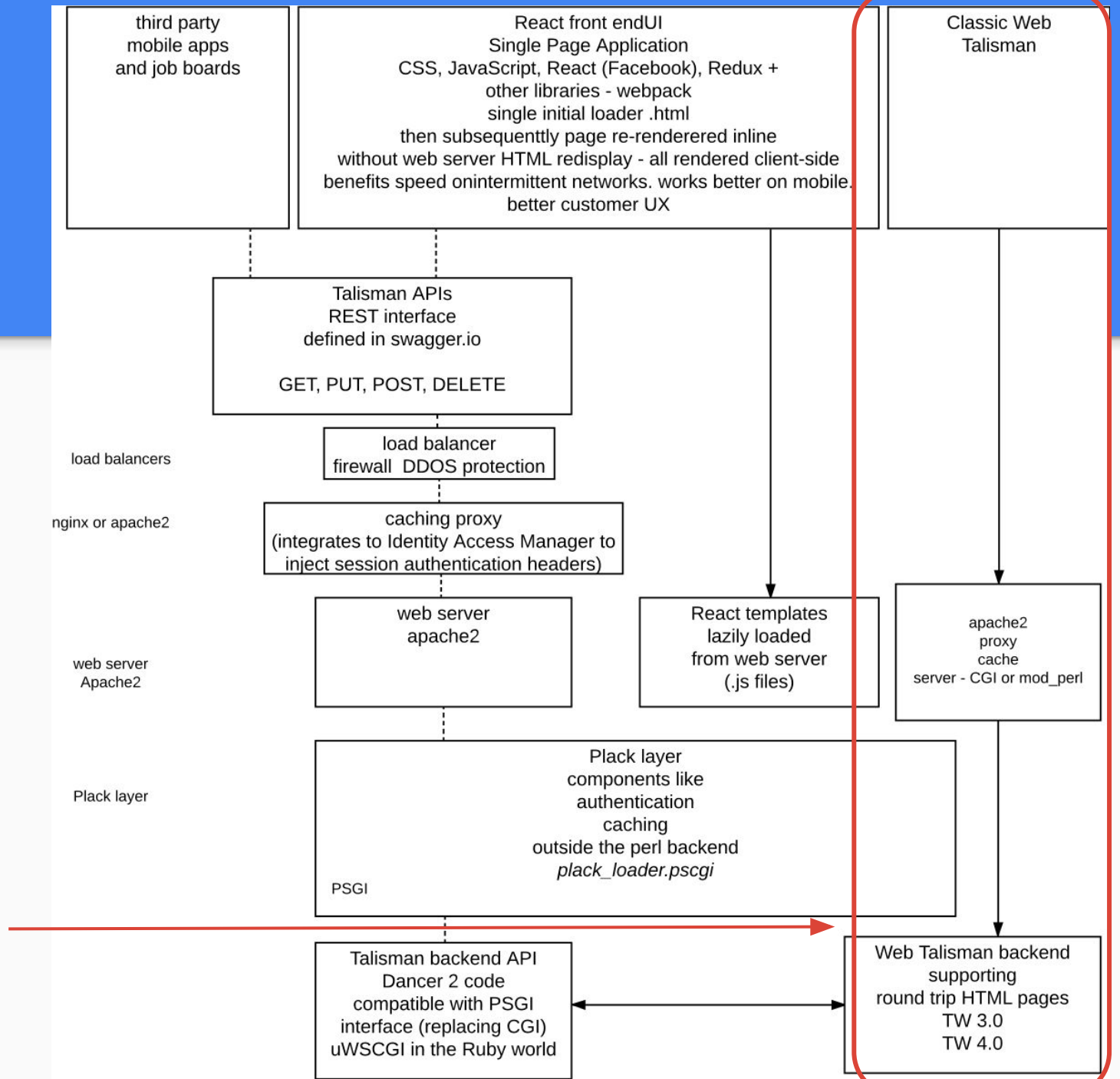
Using Modern Perl - some resources

- The Modern Perl book by Chromatic:
  http://www.modernperlbooks.com/books/modern_perl_2016/index.html
  https://github.com/chromatic/modern_perl_book/
- Perl Maven https://perlmaven.com/ by Gabor Szabo
- Resources at https://learn.perl.org/
- Perl Best Practices as reflected in perlcritic
  https://metacpan.org/pod/perlcritic
- Recommended modules for Enlightened Perl
  https://metacpan.org/pod/Task::Kensho

But what about the deprecated old stack?

A system architecture diagram I created to help visualise the “to be” state.

The red circled part is what we’ll look at today:
Modernising the legacy CGI to run on Plack and remove apache 1 + mod_perl 1



What do we mean by old perl?

Versions shipped by Red Hat: we used RH 7, 9 then, later RHEL 2, 3, 5, 7, Rocky Linux 8

1999 perl 5.005, legacy C-ISAM database
2002 perl 5.8.0 was problematic with utf-8
2003 perl 5.8.1 **this is where a lot of our code got "parked",** legacy + MySQL 3.x
2008 perl 5.8.8
2014 perl 5.16.3 ships with RHEL7, legacy + MySQL 5.x

2003, rewrote much early code to run under mod_perl1 with Apache::PerlRun

Mod_perl1 porting guidelines for CGI code https://perl.apache.org/docs/1.0/guide/

- use strict; use warnings; # mea culpa, mea maxima culpa
  update the header
- Local variables my $x -> our $x
- Use braces to create scope to remove code from scope after execution
  {
  … old code goes here, will now get run on next invocation
  }
- Replace globals and procedural code with objects
  - but it may be impractical if there is too much
- Garbage collection, mod_perl 1: end of execution hook: release locks and other resources

Mod_perl1 porting guidelines for CGI code https://perl.apache.org/docs/1.0/guide/

- use strict; use warnings; # mea culpa, mea maxima culpa
  update the header
- Local variables my $x -> our $x
- Use braces to create scope to remove code from scope after execution
  {
  … old code goes here, will now get run on next invocation
  }
- Replace globals and procedural code with objects
  - but it may be impractical if there is too much
- Garbage collection, mod_perl 1: end of execution hook: release locks and other resources

How did apache 1 work?

/etc/httpd/conf/httpd.conf

ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"

apache sees a file with .cgi extension and executes it
this means many entry points

/var/www/cgi-bin/tal_dev4/cgi/

bookings/list.cgi, bookings/view.cgi, bookings/amend.cgi, …
candidate/list.cgi, candidate/view.cgi, …

600 scripts later

mod_perl1

This was the perl interpreter compiled with the apache1 C code into a single binary to execute scripts without the perl binary load time overhead.

We already had a lot of CGI scripts and not enough time to rewrite to use Apache::Registry. We used Apache::PerlRun to read the scripts into memory and fake a CGI environment for them to run in.
We put a system apache 2 as front end proxy in front.

It ran a lot faster and was solid for the past 20 years. We couldn't justify the cost of rewriting it to mod_perl2 in 2005… another compromise.

mod_perl1

This was the perl interpreter compiled with the apache1 C code into a single binary to execute scripts without the perl binary load time overhead.

We already had a lot of CGI scripts and not enough time to rewrite to use Apache::Registry. We used Apache::PerlRun to read the scripts into memory and fake a CGI environment for them to run in.
We put a system apache 2 as front end proxy in front.

It ran a lot faster and was solid for the past 20 years. We couldn't justify the cost of rewriting it to mod_perl2 in 2005… another compromise.

Obsolescence
(Not old programmers :-)

CGI.pm HAS BEEN REMOVED FROM THE PERL CORE

http://perl5.git.perl.org/perl.git/commitdiff/e9fa5a80

If you upgrade to a new version of perl or if you rely on a system or vendor perl and get an updated version of perl through a system update, then you will have to install CGI.pm yourself with cpan/cpanm/a vendor package/manually. To make this a little easier the CGI::Fast module has been split into its own distribution, meaning you do not need access to a compiler to install CGI.pm

The rationale for this decision is that CGI.pm is no longer considered good practice

CGI.pm is discouraged from use

and getting apache1 + mod_perl1 to compile on Red Hat Enterprise Linux 7

I had to modify the apache1 C code. Getting it to compile and run without segmentation violations **was “quite tricky”**.

We decided it had to go and be replaced with Plack, before we moved to Rocky Linux 8, in case Bad Things Happened.

Moving CGI to Plack

With a modern framework, you have a single persistent entry point on a master server that routes web requests and calls the correct class and method to process it.

We can fake that, without rewriting all those old scripts, by using Plack::App::CGIBin to map the web request URL path to script files.
This reads them into memory and then eval()s on each web request.

Using SSO and SSO proxy instead of apache 2 front end proxy

We added on the front end Gluu https://gluu.org/single-sign-on/ SSO for people to login (replacing the old perl login web form, though that remained as an option for when directly logging in to the server not via SSO).

We front it with nginx as a master router.
Gluu SSO proxy replaces old apache 2 front end to route authorised requests.
Gluu uses apache 2 as its proxy server inside a docker container.

In the old world we used cookies for user and session for authentication.
In the new world, it is different..

SSO proxy configuration - authentication header

We pass web headers OIDC-CLAIM-EMAIL and OIDC-CLAIM-SUB to the backend as auth. Changed the framework to allow auto-login for matching user if IP origin was SSO proxy.

/opt/iam/psgis/apache-docker-tal-qa-psgi/httpd/conf/extra/httpd-ssl.conf

<VirtualHost _default_:443>

...

ReWriteEngine On
RequestHeader **set OIDC-CLAIM-EMAIL** "expr=%{ENV:OIDC_CLAIM_email}"
RequestHeader **set OIDC-CLAIM-SUB** "expr=%{ENV:OIDC_CLAIM_sub}"

SSO proxy configuration - routing

We route /cgi-{bin,perl}/ requests to the Plack backend legacy CGI service, here port 24001

/opt/iam/psgis/apache-docker-tal-qa-psgi/httpd/conf/extra/httpd-ssl.conf

```
# redirect / to default cgi/home.cgi
ReWriteRule ^/$ http://52.56.93.98:24001/cgi-bin/tal_qa/cgi/home.cgi [P]

# proxy /cgi-bin/ and /cgi-perl/ requests to Plack backend service on qa server
# to run older code
ReWriteRule ^/cgi-bin/(.*$) http://52.56.93.98:24001/cgi-bin/$1 [P]
ReWriteRule ^/cgi-perl/(.*$) http://52.56.93.98:24001/cgi-bin/$1 [P]
```

SSO proxy configuration - routing 2

Static assets remain served by apache2 port 80 on same server as backend

/opt/iam/psgis/apache-docker-tal-qa-psgi/httpd/conf/extra/httpd-ssl.conf

```
# static assets served by apache 2 on qa server
ReWriteRule ^/tal/(.*$) http://52.56.93.98/tal/$1 [P]
ReWriteRule ^/icons/(.*$) http://52.56.93.98/icons/$1 [P]
```

SSO proxy configuration - routing 3

New backend API is routed to TWAPI service, here port 17339

/opt/iam/psgis/apache-docker-tal-qa-psgi/httpd/conf/extra/httpd-ssl.conf

```
  # TWAPI - Raisin API service to run newer code, also under Plack
  <Location /apiclient/twapi/tal-qa/>
    AuthType auth-openid
    OIDCUnAuthAction 401
    ProxyPass http://52.56.93.98:17339/
    ProxyPassReverse http://52.56.93.98:17339/
  </Location>

  ProxyPassReverse / http://52.56.93.98/
```

Plack psgi loader

This provides the service point referenced above

http://52.56.93.98:24001/cgi-bin/$1

We have a main entry point

{Application root}/cgi/plack_loader.psgi

How we run it - manually

We can run it as a development standalone server
$ cd /var/www/cgi-bin/tal_qa/cgi
$ **plackup plack_loader.psgi**
(defaults to listening at localhost port 5000)

to run with the perl debugger you need invoke plackup, not the .psgi
$ perl -d **/usr/local/bin/plackup** plack_loader.psgi
You can also start the debugger from VSCode by passing psgi filename arg
https://marketplace.visualstudio.com/items?itemName=richterger.perl

How we run it - from systemd

Normally we start it as a service using a systemd unit to run under Starman PSGI web server

$ **systemctl status tal-psgi@tal_qa.service**
● **tal-psgi@tal_qa.service** - Start Talisman PSGI service for tal_qa
Loaded: loaded (/usr/lib/systemd/system/tal-psgi@.service; disabled; vendor preset: disabled)
Active: active (running) since Mon 2023-08-14 02:03:03 BST; 10h ago
Process: 22044 ExecStart=**/usr/local/bin/plackup --port $PORT --workers $WORKERS --daemonize --access-log=${LOGPREFIX}_access.log --error-log=${LOGPREFIX}_error.log --pid=${LOGPREFIX}_server.pid -a ${BASEDIR}/%i/cgi/plack_loader.psgi** (code=exited, status=0/SUCCESS)
Main PID: 22045 (starman master)
CGroup: /system.slice/system-tal\x2dpsgi.slice/tal-psgi@tal_qa.service
├─22045 starman master
├─22046 starman worker
├─22047 starman worker …

Systemd environment file

$ cat /etc/sysconfig/tal-psgi/tal_qa

BASEDIR=/var/www/cgi-bin
PORT=24001
WORKERS=5
PLACK_SERVER=Starman
PLACK_ENV=production
AREA=tal_qa
LOGPREFIX=/beacon/logs/tal-psgi/tal_qa

Systemd unit

$ cat /usr/lib/systemd/system/tal-psgi@.service

```
[Unit]
Description=Start Talisman PSGI service for %i
After=syslog.target network-online.target
Wants=network-online.target

[Service]
Type=forking
Restart=on-failure
RestartSec=5
EnvironmentFile=/etc/sysconfig/tal-psgi/%i
ExecStart=/usr/local/bin/plackup --port $PORT --workers $WORKERS --daemonize
--access-log=${LOGPREFIX}_access.log --error-log=${LOGPREFIX}_error.log --pid=${LOGPREFIX}_server.pid -a
${BASEDIR}/%i/cgi/plack_loader.psgi

[Install]
WantedBy=multi-user.target
```

In the plack_loader.psgi script

We use these to provide the legacy CGI web service:
- Plack PSGI toolkit
- Plack::App::CGIBin as a cgi-bin replacement
uses CGI::Compile to read and cache perl scripts into subs (like ModPerl::Registry)
and then run it as a persistent application using CGI::Emulate::PSGI
- Plack::App::MCCS to serve static assets
- Plack::Builder
Inside a builder block we map an apache ExecCGI **/cgi-bin/**AREA path to Plack::App::CGIBin for later execution.

plack_loader.psgi

```
use strict;
use warnings;
use Beacon::TalPath qw/TalPath/;
use Plack;
use CGI::Compile;
use Plack::Builder;
use Plack::App::MCCS;
use Plack::App::CGIBin;
use File::Basename qw();
```

plack_loader.psgi - handle non-safe scripts

```
sub should_run_with_child_exec_perl {
   my $file = shift // '';
   $ENV{SCRIPT_FILENAME} = $file; # so Talsettings::calc_appfile_path() can find cgi
root directory
   my $scriptdir = File::Basename::dirname($file);
   chdir($scriptdir) || die "cannot change to CGI script directory $scriptdir: $!";
   ... returns boolean whether we need to fork a new perl child each time
}
```

We will see how this works later

Legacy framework expects apache environment variable

plack_loader.psgi - if you need an init hook

```
# EXAMPLE OF an init hook before Talisman code
# {
#  my $orig = *CGI::initialize_globals{CODE};
#  no warnings 'redefine';
#  *CGI::initialize_globals = sub {
#     print STDERR "CGI::initialize_globals()\n";
#     <<do something here>>
#     &$orig;
#  };
# }
```

plack_loader.psgi - end hook for cleanup

```
# CGI::Compile does not have a register_cleanup hook like mod_perl1
# so wrap its serve_path method to call Talisman cleanup if it exists
{
   my $orig = *Plack::App::CGIBin::serve_path{CODE};
   no warnings 'redefine';
   *Plack::App::CGIBin::serve_path = sub {
      my @results = &$orig(@_);
      Tal::Common::sm_End() if defined &Tal::Common::sm_End; # Talisman cleanup
      return @results; # HTTP response header, content
   };
};
```

Plack_loader.psgi - app root and area name

```
my $rootdir = TalPath(); # e.g. /var/www/cgi-bin/tal_pe
my $area = $ENV{PLACK_TAL_AREA} # if set in systemd environment file use that
        || pop [split '/', $rootdir] # otherwise work out area name from rootdir e.g. tal_pe
        || die 'cannot locate area from rootdir '.$rootdir;
print STDERR "plack_loader.psgi running under rootdir [$rootdir] area [$area]\n";
```

Plack_loader.psgi - build the mount points

```
builder {
   # enable 'Plack::Middleware::MCCS', path => qr{/^icons/}, root =>
'/usr/share/httpd/icons/'; # alternative
   mount '/icons' => Plack::App::MCCS->new( # /usr/share/httpd/icons/apache_pb.gif
      root => '/usr/share/httpd/icons',
      defaults => { etag => 0, cache_control => ['no-cache'], compress => 0 },
   )->to_app;
...
```

Plack_loader.psgi - build the mount points

```
  # automatically load, compile and run perl scripts
  # e.g. http://localhost:5000/cgi-bin/tal_dev4/cgi/candidate/view.cgi/10234
  #      -> execute code in /var/www/cgi-bin/tal_dev4/cgi/candidate/view.cgi
  mount "/cgi-bin/$area/cgi" => builder {
     Plack::App::CGIBin->new(
        root => "/var/www/cgi-bin/$area/cgi",
        exec_cb => \&should_run_with_child_exec_perl,
     )->to_app;
  };
...
```

Plack_loader.psgi - build the mount points

```
   # how to mount a URL for a single perl script
   #mount '/cgi-bin/tal_dev4/cgi/advance/list.cgi' => Plack::App::WrapCGI->new(script =>
"advance/list.cgi")->to_app;
   # PATH_INFO '/tal_dev4/cgi/advance/list.cgi
   # REQUEST_URI /cgi-bin/tal_dev4/cgi/advance/list.cgi'
...
```

Plack_loader.psgi - end

```
  # default static serve everything else below root /
  mount '/' => Plack::App::MCCS->new(
    root => '/var/www/html',
    defaults => { etag => 0, cache_control => ['no-cache'], compress => 0 },
  )->to_app;
  # how to proxy static file serving to a shared central server
  #mount '/', Plack::App::Proxy->new(remote => 'http://localhost/')->to_app,
  #mount '/icons', Plack::App::Proxy->new(remote => 'http://localhost/icons')->to_app,
};
```

Normally apache2 does this in production,
via nginx and SSO proxy.
This will work for local dev.

Old headers in CGI scripts from perl 5.005

We're not going to rewrite 600 working scripts. Fix their headers instead.

```
#!/usr/bin/perl -w                #!/usr/bin/env perl
# bookingpay/amend.cgi            #vim:ts=3:shiftwidth=3:expandtab
# %I% %D%                         Antique SCCS field replaced when checking file out - delete!
use strict;
$^W = 1;                          becomes   use 'warnings';   in perl 5.008
package main;                     some legacy system modules expect to be in global namespace - keep
use lib ( "..", "../pmtal", "../pmsys", $ENV{PERLMOD_TAL}||"/usr/local/perlmod/tal",
$ENV{PERLMOD_SYS}||"/usr/local/perlmod/sys" );          fix
use vars qw($VERSION); $VERSION = do { my @r = (q$Revision: %I% $ =~ /\d+/g); sprintf
"%d."."%02d" x $#r, (@r?@r:0) }; # must be all one line, for MakeMaker     obsolete
```

After

We're not going to rewrite 600 working scripts. Fix their headers instead.

```
#!/usr/bin/env perl
# vim:ts=3:shiftwidth=3:expandtab
use strict;
use warnings;
package main; # some legacy system modules expect to be in global namespace
use Beacon::TalPath; # not FindBin::Real as it gets confused by Plack and CGI::Compile wrapper
use lib "$Beacon::TalPath::Bin/cgi"; # needed for Plack and VSCode perl debugger as it runs from
a different directory
use TalSyntax; # set strictures, turn on features, add lib paths, use standard Talisman modules
```

Common include header

This is something I wished I'd known about earlier, before editing 600+ .cgi scripts.

John Napiorkowski
https://dev.to/jjn1056/using-modern-perl-features-in-your-projects-4e7m

"I create a '::Syntax' module in the root of my project namespace and use that to setup all the features I want."

TalSyntax.pm - set perl library paths

```
package TalSyntax;
use strict;
use warnings;
use Import::Into;    # works from perl 5.006
use Module::Runtime; # works from perl 5.006

# FindBin gets confused by Plack and CGI::Compile wrapper so use Beacon::TalPath
use Beacon::TalPath; # needed to work across Plack + CGI::Compile, mod_perl, CGI, interactive
use lib "$Beacon::TalPath::Bin/cgi";
use lib "$Beacon::TalPath::Bin/cgi/pmtal";
use lib "$Beacon::TalPath::Bin/cgi/pmsys";
use lib "$Beacon::TalPath::Bin/cgi/am";
use lib "$Beacon::TalPath::Bin/cgi/document/Module";
```

TalSyntax.pm - turn on features

```
# standard use declarations to apply in header of all modules
sub importables {
  my ($class) = @_;
  return (
    # Features/pragmas
    'utf8',
    'strict',
    'warnings',
    ['feature', ':5.16'], # see https://perldoc.perl.org/feature, turns on the following features
      # bareword_filehandles current_sub evalbytes
      # fc indirect multidimensional say state
      # switch unicode_eval unicode_strings
    #['feature', 'say'], # if you want a specific feature
    #['experimental', 'signatures', 'postderef'], # needs perl 5.20.0 and later, our base standard was 5.16.3
```

TalSyntax.pm - use application modules

```
    # Modules. These were imported previously in a worse way by pmtal/Tal.pm into the 'main' namespace
    'Tal::GlobalVars',
    'Talsettings',
    'Tal',
    'Tal::Common',
    'Tal::Browser',
    'Tal::Date',
    'Tal::Display',
    'Tal::SQL',
    'Tal::Script',
    'Tal::Security',
    'Tal::User',
    'Tal::Utility',
  );
}
```

TalSyntax.pm - hook to do the import

```
sub import {
  my ($class, @args) = @_;
  my $caller = caller;
  foreach my $import_proto($class->importables) {
    my ($module, @args) = (ref($import_proto)||'') eq 'ARRAY' ?
      @$import_proto : ($import_proto, ());
    Module::Runtime::use_module($module)
      ->import::into($caller, @args)
  }
}
```

Badly behaved scripts need exec perl (fork and run a new perl each time to run those)

Some ancient code and frameworks are not cost-effective to fix as the products may be legacy and will go away in a few years.

Previously handled with apache2 proxy rules:

```
  <IfModule mod_rewrite.c>
  <VirtualHost *:80>
    Include /etc/httpd/conf.d/proxy_rules.inc
    ...
  # run tal_dev4 accounts framework under apache2 cgi-bin exec perl
  RewriteRule       ^/cgi-perl/tal_dev4/cgi/accounts/(.*$)
http://localhost:80/cgi-bin/tal_dev4/cgi/accounts/$1 [L,P]
  RewriteRule       ^/cgi-perl/tal_dev4/cgi/customer/(.*$)
http://localhost:80/cgi-bin/tal_dev4/cgi/customer/$1 [L,P]
```

Badly behaved scripts need exec perl (fork and run a new perl each time to run those)

Now handled by Plack in Plack::App::CGIBin exec_cb hook, that we saw earlier

```
# in plack_loader.psgi
   Plack::App::CGIBin->new(
      root => "/var/www/cgi-bin/$area/cgi",
      exec_cb => \&should_run_with_child_exec_perl,
   )->to_app;

sub should_run_with_child_exec_perl {
   my $file = shift // '';
   ...
```

Badly behaved scripts need exec perl (fork and run a new perl each time to run those)

```
# unsafe programs need child exec perl
if ( $file =~ m{
   (
      cgi/accounts/
   |  cgi/customer/
   ...
   )
   }x )
{
   $should_exec = 1;
}
# for other .pl and .cgi files, run persistently in this perl via a CGI wrapper
elsif ( $file =~m/(.cgi|.pl)$/ )
{
   $should_exec = 0;
}...
```

Finding application root

I mentioned we wrote a module Beacon::TalPath to work out application root.

You have a few options:

1. systemd environment to set PERL5LIB or application root directory APPDIR.
   Equivalent of apache config: PerlSetEnv MYLIB /srv/http/site/apps/thisone/lib
2. Relative to main app file: use File::Basename; use lib dirname(__FILE__);
3. For our legacy framework, it expects $ENV{SCRIPT_FILENAME} so we set that in plack_loader.psgi based off exec_cb callback (which is passed file path)
4. For CGI or mod_perl1 with Apache::PerlRun - FindBin::Real::Bin() - breaks under Plack and CGI::Compile as $0 is not what FindBin::Real expects

Other changes needed for CGI::Compile

Remove unneeded globals

Remember you have to reset them every run. They will persist the previous run's value leading to unpredictable failures if you are not careful. Why risk it?

```
my $booking_no;

sub run {
  $booking_no ||= CGI->new->multi_param('booking_no'); # oops, got last run value
```

Other changes needed for CGI::Compile

Replace 'my' lexicals with 'our' package globals.
https://metacpan.org/pod/CGI::Compile
“If your CGI script has a subroutine that references the lexical scope variable outside the subroutine, you'll see warnings such as:
Variable "$q" is not available at …
Variable "$counter" will not stay shared at …
This is due to the way this module compiles the whole script into a big sub. To solve this, you have to update your code to pass around the lexical variables, or replace my with our. See also http://perl.apache.org/docs/1.0/guide/porting.html#The_First_Mystery for more details.”

Example of tidying up an old script that is too much effort/risky to fully rewrite

```
#!/usr/bin/perl -w
# timesheets/attach.cgi
# %I% %D%
use strict;
$^W = 1;
package main;
use lib ( "..", "../pmtal", "../pmsys", $ENV{PERLMOD_TAL}||"/usr/local/perlmod/tal",
$ENV{PERLMOD_SYS}||"/usr/local/perlmod/sys" );
use vars qw($VERSION); $VERSION = do { my @r = (q$Revision: %I% $ =~ /\d+/g); sprintf "%d."."%02d" x
$#r, (@r?@r:0) }; # must be all one line, for MakeMaker
use Tal;
use Tal::List;
use timesheets::sm_timesheets;
use tsimage::ListTSImage;
```

The usual cruft

Example of tidying up an old script that is too much effort/risky to fully rewrite

```
use Document;
use View::Slcust;
require '../timesheets/display_common.pl';
require '../timesheets/display_timesheet.pl';
require '../timesheets/read_timesheet.pl';
sm_Common     ();
sm_PrintHead ("Attach Image To Timesheet");
sm_PrintBody ();
...
```

Written by a C programmer like #include “display_common.c”

Code in top scope only runs on initial parse, not on subsequent runs

After

```
#!/usr/bin/perl -w
# timesheets/attach.cgi
package main;
use Beacon::TalPath; # not FindBin as it gets confused by Plack and CGI::Compile wrapper
use lib "$Beacon::TalPath::Bin/cgi"; # needed for Plack and VSCode perl debugger as it runs from a different directory
use TalSyntax; # set strictures, turn on features and use standard Talisman modules
```

After

```
use Tal::List;
use timesheets::sm_timesheets;
use Document;
use View::Slcust;
# code moved into timesheets::Common
#require '../timesheets/display_common.pl';
#require '../timesheets/display_timesheet.pl';
#require '../timesheets/read_timesheet.pl';
use vars qw(%tc);
use timesheets::Common;
```

After

```
# globals the .pl files previously created, now declared as package globals
our ($timesheet_no, $surname, $serial_code, $serial_multi, $doctype, $button,
$multi_image,
    $image_cur_image, $lookfor, $timesheet_no_key, $timesheet_keys);

# code moved inside block to ensure it is run when CGI::Compile re-runs it
{      Must be in block or it won't run again next time
   sm_Common    ();
   sm_PrintHead ("Attach Image To Timesheet");
   sm_PrintBody ();
```

That’s it! Any questions?

Thanks.

https://www.talisman.tech/
https://linkedin.com/in/peteredwards/
http://dragonstaff.co.uk/perl/ - link to slides is here

PerlKohaCon, Helsinki Aug 15th 2023