Learning from Others

Solutions to problems inspired by Tech meets like this one

Problem One

- Need to monitor 12 environments
- Each environment runs on 7 machines
- Each reside in their own Solaris Zone (Virtual Machine)
- Each run in a dedicated Solaris Role
- So need to monitor processes on 84 different user/machine combinations

Solution Part One

- Set up an NFS partition mounted on all zones visible to all roles
- Write a script to checkout the status located on the NFS share and setup CRON or another scheduler to run the script repeatedly throughout the day - the script emails any issues.

Problem Two

- Management want an up-to-date status of all environments on request and regularly throughout the day

Solution Two

- Create a Memcached server
- Get the checking script to update memcached with the status of each machine
- Use an ajax enabled web page to display the contents of memcache

Setting up and using a Memcached Server

Get memcache from http://memcached.org and install it
On the server that you want to run memcached run
memcached -d
Install CPAN module Cache::Memcached

```
use Cache::Memcached;

my $memd = new Cache::Memcached {
     servers => [ ‘192.168.0.1:11211’], # replace with real IP address of memcached server
     debug => 0,
     };
$memd->set(“key”, “some value”);
my $value = $memd->get(“key”);
```

Notes:
“some value” can be anything - scalar, data structure, object e.t.c
In order to know what keys are available set up a entry with a know key value containing all the available keys - you can also maintain the last updated timestamp so you know when to update.
e.g.

```
my $keys = $mend->get(“allkeys”);
$keys->{“thiskey”} = time();
$mend->set(“allkeys”, $keys);
```

```
<html>
  <head>
    <script src="OpenThought.js"></script>
       <script type="text/javascript">
       var c=0; var t; var timer_is_on=0;
       function timedCount()
       {
              OpenThought.CallUrl( '/cgi-bin/summary_ajax.pl', 'summary');
              t=setTimeout("timedCount()",5000);  # run every 5000 miliseconds (i.e. 5 seconds)
       }
       function doTimer()
       {
              if (!timer_is_on)
             {
                     timer_is_on=1;
                     timedCount();
              }
       }
       function addLoadEvent(func) {
                     var oldonload = window.onload;
                     if (typeof window.onload != 'function') {
                            window.onload = func;
                     } else {
                            window.onload = function() {
                                   if (oldonload) {
                                          oldonload();
                                   }
                                   func();
                            }
                     }
              }
              addLoadEvent(doTimer);
       </script>
  </head>
  <body>
  <form name="my_form" onSubmit="return false">
         <p id="summary">Summary</p>
 </form>
 </body>
</html>
```

```
#!/usr/bin/perl
use strict;
use warnings;
use OpenThought();
use CGI;
my $q = new CGI;
my $params = $q->param(“summary”);
my $OT = OpenThought->new();
my $data;
$data->{summary} = get_summary();
$OT->param($data);
print $OT->response() or warn $!;

sub get_summary {
    return <Some HTML to display>;
}
```

Problem Three

- Need to find a way to start and stop environments without needing to log in to 7 boxes each time
- Also need to provide controlled access to 50+ developers without giving them full login access

Several Solutions

- Enable HTTPS on Apache and use the NTLM Module to authenticate users on Windows login and then provide selected access to cgi scripts which do the necessary work based on login
- Allow access to one server and from there give access to other boxes via Expect
- Use RPC::PlServer and RPC::PlClient to enable controlled access to certain actions

Access via Apache

- Lots of setup required e.g. Auth keys for every user
- Have to maintain who has access to what somewhere
- No feedback provided until script has finished so not appropriate for long running scripts

Using Expect

- Difficult to get right - has tendency to time-out waiting for a response
- Feedback can be confusing because every log-in displays a log-in banner so it is hard to tell if it worked or not
- Error handling is challenging

RPC::PIServer & RPC::PIClient

- Most flexible and powerful method
- Can provide up-to date feedback via Memcached
- Easy to tell is it worked or not
- Can be initiated from Perl on Windows

Setting up RPC::PlServer

```
#!/usr/bin/perl -T
package Whatever;
use RPC::PlServer;
use base “RPC:;PLServer”;

eval {
        my $server = My->new({
                'configfile' => "Whatever.conf",
                'methods'    => {
                  'Whatever' => {
                        NewHandle => 1,
                        CallMethod => 1,
                        AnotherMethod => 1
                             },
                      },
              }) or die "Can't create Server: $!\n";
        $server->Bind() or die "Can't Bind: $!\n";
};
if ($@) {
        warn("Can't create server: $@\n");
        exit(0);
}
~
```

Config File

```
{
        'pidfile'    => 'somepidfile.pid',
        'facility'   => 'daemon',
        'user'        => 'user',
        'group'       => 'user',
        'localport'  => 60009,
        'logfile'    => "STDERR",
        'mode'       => 'fork',   # Recommended for Unix
        'clients' => [
                        {
                                'mask' => '^10\.214\.11\.179$', # regexp for IP address(es)
                                'accept' => 1,
                                'users' => [ 'user1', 'user2', 'user3', 'user4' ]
                        },
                        {
                                'mask' => '.*', # anything else
                                'accept' => 0 # Don't allow access
                        },
                ]
}
```

An Example Client

```
#!/usr/bin/perl
use strict;
use RPC::PlClient;

my $client = RPC::PlClient->new(peeraddr        => <Host where Server is running>,
                                peerport        => 60009,
                                application     => 'Whatever',
                                version         => '1.0',
                                user            => 'user',
                               ) or die "Can't create Client: $!\n";
carp ("Here");
my $object = $client->ClientObject('Whatever', 'new', MXENV => $env) or die "Can't create Client Object $!\n";

#  Use object  handle as if local
```