ZABBIX 2013 Conference

Integrated Dashboard Design

integrating Zabbix data with other systems

Lukasz Lipski

IT Operations Specialist, Nordea Bank Polska SA

September 2013

NORDEA IT POLAND AND BALTIC COUNTRIES

- IT support for our Polish, Latvian, Lithuanian and Estonian branches
- core banking, e-banking, backoffice, and in-house development
- large, complex IT infrastructure under constant monitoring

ZABBIX AT NORDEA

- 300 hosts
- 40000 items
- 10000 triggers

MONITORING COMPLEXITY

ZABBIX

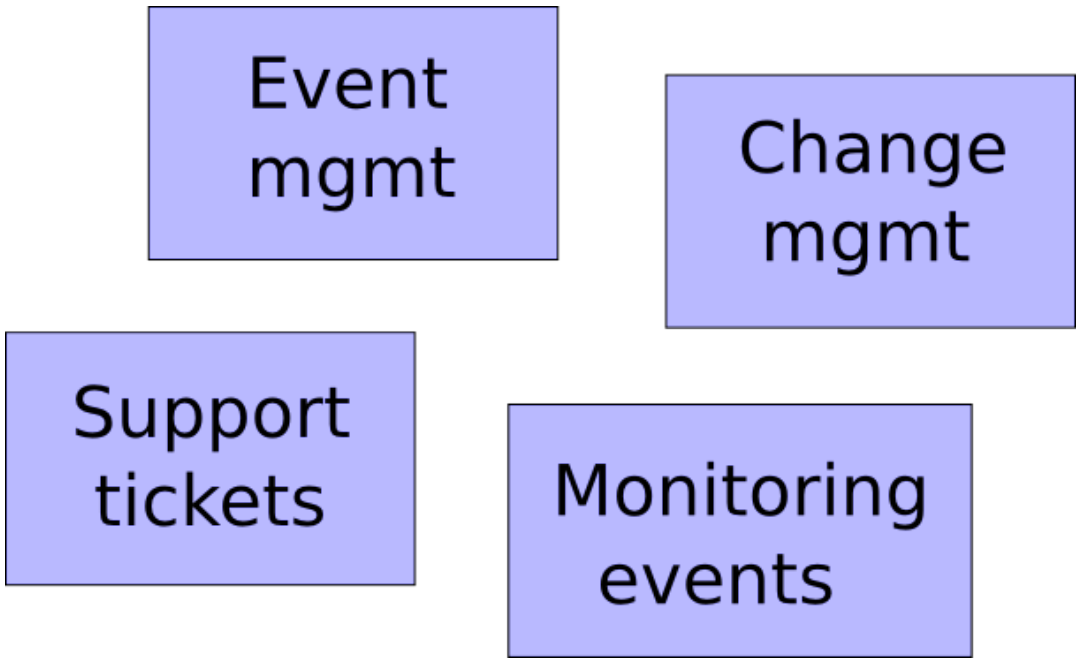
Microsoft® System Center Operations Manager

COMPUWARE APM
DYNATRACE | GOMEZ

bmcsoftware
Remedy

and more ...

SYSTEM COMPLEXITY



MAIN POINTS

- integrating data has its benefits
- those benefits are not difficult to obtain

INTEGRATING DATA

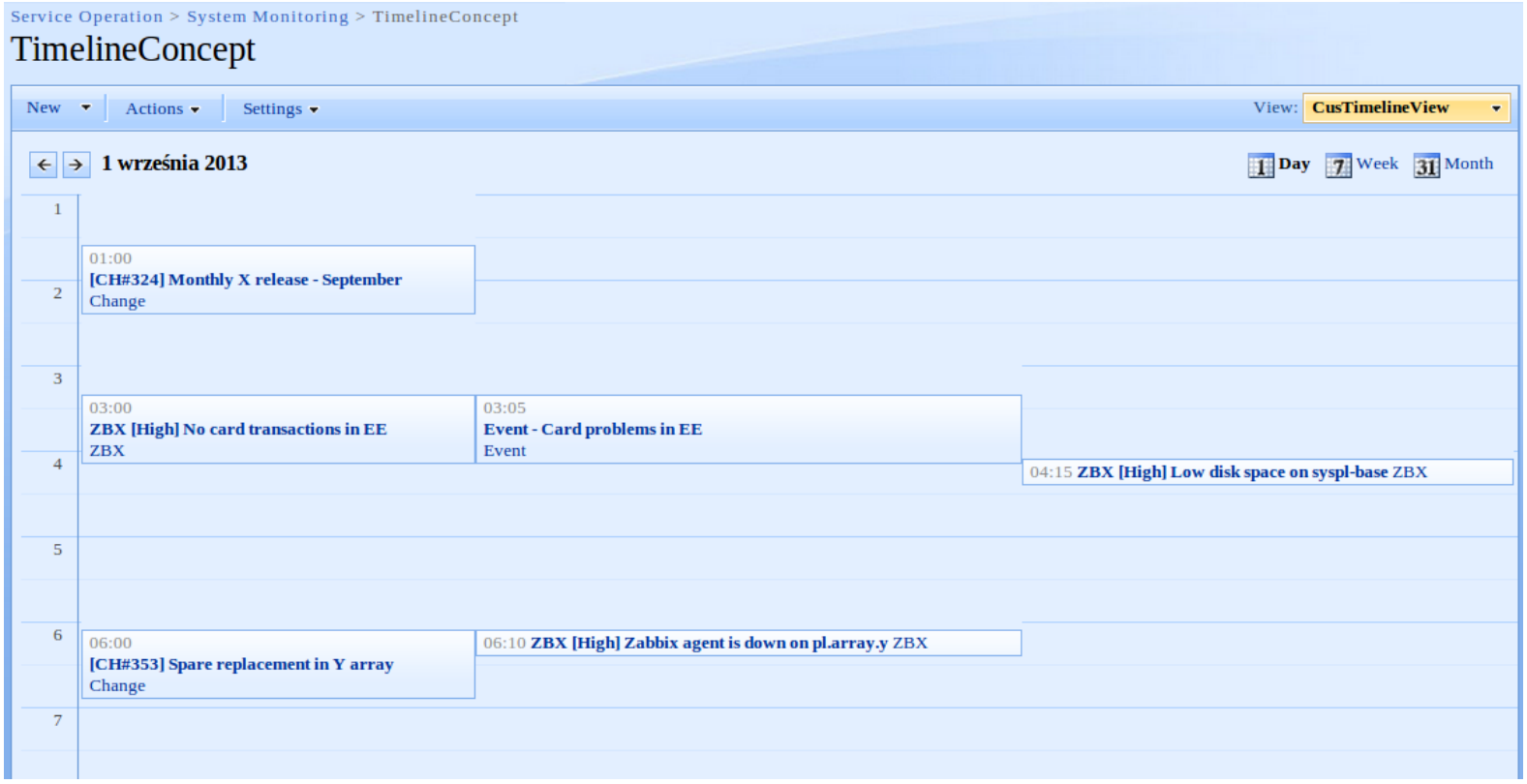
EXAMPLE 1: TIMELINE

Components:

- Change management data (BMC Remedy)
- Event management data (SharePoint)
- Monitoring data (Zabbix)

Allows to figure out some associations between incidents, registered events and recently introduced changes.

EXAMPLE 1: EXECUTION



EXAMPLE 2: DASHBOARD

Components:

- Registered changes – SharePoint
- System state – Zabbix

Allows to see immediately whether an ongoing change is affecting system state, and call the on-duty admin.

EXAMPLE 2: EXECUTION

System state and planned changes

Changes

#id	T start	T end	Change	Owner
453	2013.08.12 1:00	2013.08.12 3:00	AIX upgrade on DB Standby	John Kowalski
454	2013.08.13 1:00	2013.08.13 2:00	Live-Standby switch	John Kowalski
455	2013.08.14 1:00	2013.08.14 3:00	AIX upgrade on DB Live	John Kowalski
445	2013.08.14 8:00	2013.08.14 9:00	FW#4 firmware update, due to #567	Adam Smith

State

- DB LIVE
- DB STNDBY
- WSPHERE LIVE
- WSPHERE STNDBY
- MIDDLEWARE
- STORAGE
- REPO

EXAMPLE 3: TICKET SYSTEM

Components:

- JIRA
- Zabbix

Allows to process a ticket at the most appropriate time – for example, when there are no other heavy jobs running.

AN ONGOING THEME

- present relevant data together
- support decision making process

Implementation example

EXAMPLE SCENARIO: CREATING A DASHBOARD

- task list (SharePoint)
- central monitoring system (Zabbix)
- contact list (SharePoint)

GETTING THE DATA: ZABBIX

USAGE: get_zbx_item.pl "host:item.key"

```
#!/usr/bin/perl

use Modern::Perl;
use Zabbix::API;

my %cfg = (ip => '0.0.0.0', user => 'zbx_user', pass => 'zbx_pass');

my $optarg = shift or die "hostname:item.key must be an argument\n";
my ($hostname, $itemkey) = split /:/,$optarg;

my $zabbix = Zabbix::API->new(server=>'http://'.$cfg{ip}.'/zabbix/api_jsonrpc.php',
                              verbosity=>0 );

$zabbix->login(user => $cfg{user}, password=>$cfg{pass});

my $host = shift @{$zabbix->fetch('Host',
                                  params=>{filter=>{host=>$hostname}})};
my $item = shift @{$zabbix->fetch('Item',
                                  params=>{hostids=>$host->id(),
                                  search=>{key_=>$itemkey}})};
my $data = $item->data();
print $data->{'lastvalue'};
```

GETTING THE DATA: SHAREPOINT

For an in-depth explanation, see:

`http://lifo101.wordpress.com/`

Post name: `sharepoint-and-ntlmv2-with-soaplite`

The following proof of concept script will:

- get the first 10 lines from a SharePoint list
- print it in CSV

SCRIPT EXAMPLE

USAGE: get_sp_tasks.pl

```
#!/usr/bin/perl -w
# get_sp_list.pl

use Modern::Perl;
use Authen::NTLM;
use SOAP::Lite;

my %cfg = (
    host     => 'sp.hostname.net:80', # must include port
    endpoint => 'http://sp.hostname.net/tasks/_vti_bin/lists.asmx',
    user     => '\\user', # must have leading backslashes
    pass     => 'Pa$$w0rd',
);

# enable NTLMv2 in Authen::NTLM
ntlmv2(1);

my $soap = SOAP::Lite
    ->proxy($cfg{endpoint}, keep_alive => 1,
            credentials => [$cfg{host}, '', $cfg{user}, $cfg{pass}])
    ->default_ns('http://schemas.microsoft.com/sharepoint/soap/')
    ->on_action(sub { $_[0] . $_[1] }) # change default SOAPAction header
    ->readable(1);
```

SCRIPT EXAMPLE – CONTINUED

```
# fetch the list
my $list_name = '{B0101010-2323-3434-4545-565656565656}'; # Team tasks

my $som = $soap->GetListItems(
    name(listName => $list_name),
    name(query => \value(
        name(Query => \value(
            name(OrderBy => \value(
                name('FieldRef' => {Name => 'Created', Ascending => 'False'}),
            )),
        )),
    )),
    name(rowLimit => 10)
);
die $som->faultstring() if defined $som->fault();
```

SCRIPT EXAMPLE – PART 3

```
my @results = $som->dataof('//GetListItemsResult/listitems/data/row');

say "ID;Deadline;Owner;Task";
foreach my $data (@results) {
    my $item = $data->attr;
    say join(';', @$item{qw( ows_ID ows_Deadline ows_Owner ows_Task )});
}

# short-cut SOAP::Data methods
sub name {
    my ($name, $value) = @_;
    my $d = SOAP::Data->name($name);

    if (ref $value eq 'HASH') {
        return $d->attr($value);
    }
    elsif (defined $value) {
        return $d->value($value);
    }
    return $d;
}
sub value {
    SOAP::Data->value(@_)
}
```

STATIC WEB PAGE MOCKUP

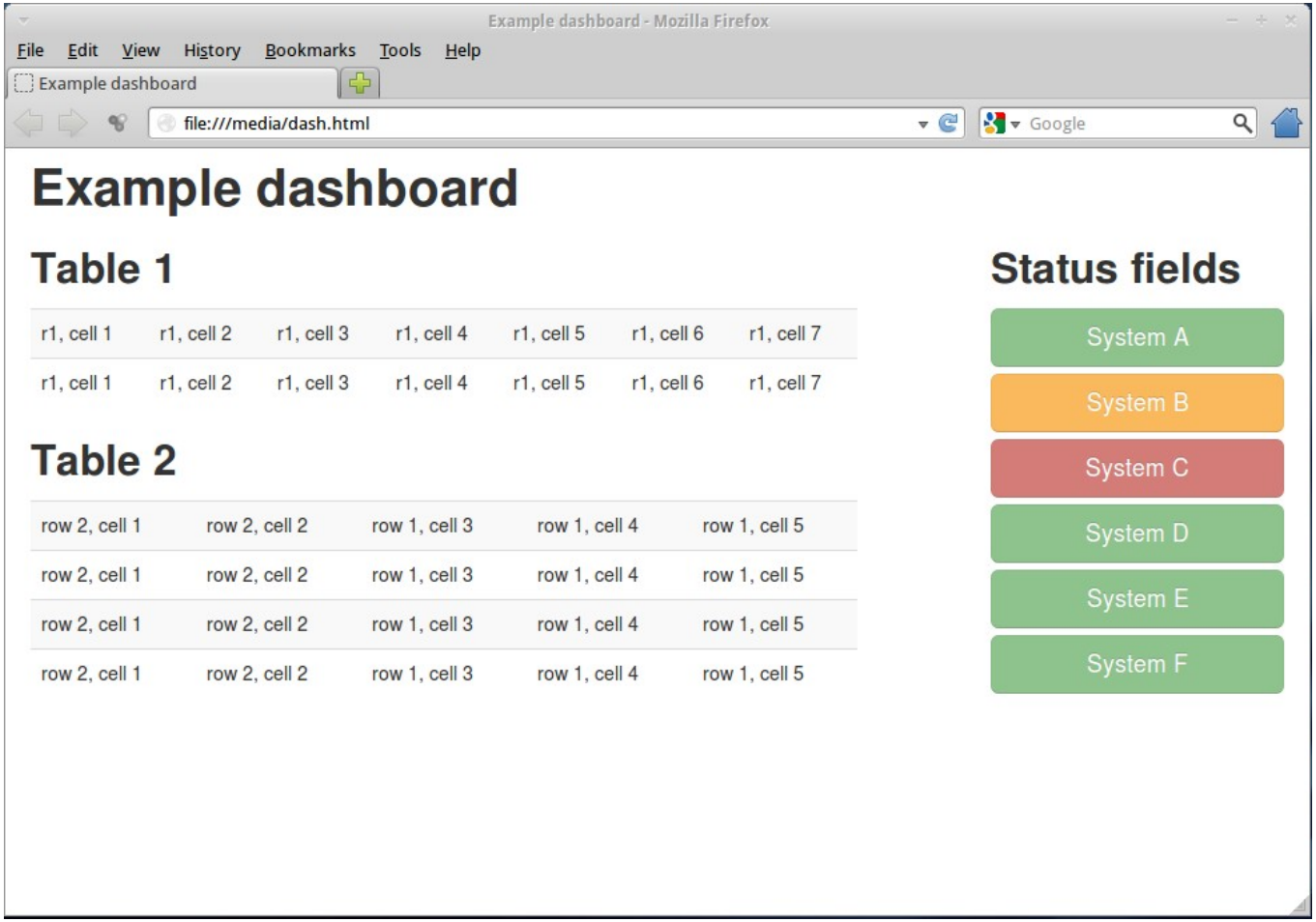
Basic building blocks:

- tables
- status fields

Considerably easier with Twitter Bootstrap:

`http://twitter.github.com/bootstrap/index.html`

EXAMPLE WEB PAGE



EXAMPLE WEB PAGE – CODE

```
<!DOCTYPE html>
<html><head>
    <title>Example dashboard</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
</head>
<body>
  <div class="container">
  <h1>Example dashboard</h1>
  <div class="row">
    <div class="span8">
        <h2>Table 1</h2>
        <table class="table table-striped">
            <tr> <td>row 1, cell 1</td> </tr>
        </table>
        <h2>Table 2</h2>
        <table class="table table-striped">
            <tr> <td>row 1, cell 1</td> </tr>
        </table>
    </div>
    <div class="span3 offset1">
        <h2>Status fields</h2>
        <button class="btn btn-large btn-block btn-success disabled" type="button">
            System A
        </button>
    </div>
  </div> </div> <!-- container -->
  <script src="http://code.jquery.com/jquery.js"></script>
  <script src="js/bootstrap.min.js"></script>
</body></html>
```

HTML TEMPLATE – NEEDED CHANGES

The HTML::Template module requires some minor changes:

Tables (tasks and admins)

```
<TMPL_LOOP NAME=tasks>
  <tr>
    <td><TMPL_VAR NAME=ID></td>
    <td><TMPL_VAR NAME=Deadline></td>
    <td><TMPL_VAR NAME=Owner></td>
    <td><TMPL_VAR NAME=Task></td>
  </tr>
</TMPL_LOOP>
```

Status fields

```
<TMPL_LOOP NAME=status>
  <button class="btn␣btn-large␣btn-block␣btn-<TMPL_VAR␣NAME=Value>␣disabled"
                type="button">
    <TMPL_VAR NAME=Item>
  </button>
</TMPL_LOOP>
```

PREPARING THE CONFIG

The whole config file:

```
[status]
system_A_status = tools/get_zbx_item.pl "hostA:system_A.status"
system_B_status = tools/get_zbx_item.pl "hostB:system_B.status"
system_X_status = tools/get_zbx_item.pl "hostX:system_X.status"
system_Y_status = tools/get_zbx_item.pl "hostY:system_Y.status"
backup_status   = tools/get_zbx_item.pl "hostZ:backup.status"

[tables]
admins = tools/get_sp_admins.pl
tasks  = tools/get_sp_tasks.pl
```

GENERATING THE DASHBOARD

Here is the final code:

`USAGE: create_dash.pl > output.html`

```
#!/usr/bin/perl
# create_dash.pl

use Modern::Perl;
use Config::IniFiles;
use HTML::Template;
use Text::CSV::Slurp;

my $cfg = Config::IniFiles->new( -file => 'dash.ini' )
    or die "Could not open config file.\n";

my $template = HTML::Template->new(filename => 'dash.tmpl')
    or die "Could not open template file.\n";

my ($cmd, $csvstring);

# admins
$cmd = $cfg->val('tables', 'admins');
$csvstring = `perl $cmd`;
my $tbl_admins = Text::CSV::Slurp->load(string=>$csvstring, sep_char=>';');
$template->param('admins' => $tbl_admins);
```

GENERATING THE DASHBOARD – CONTINUED

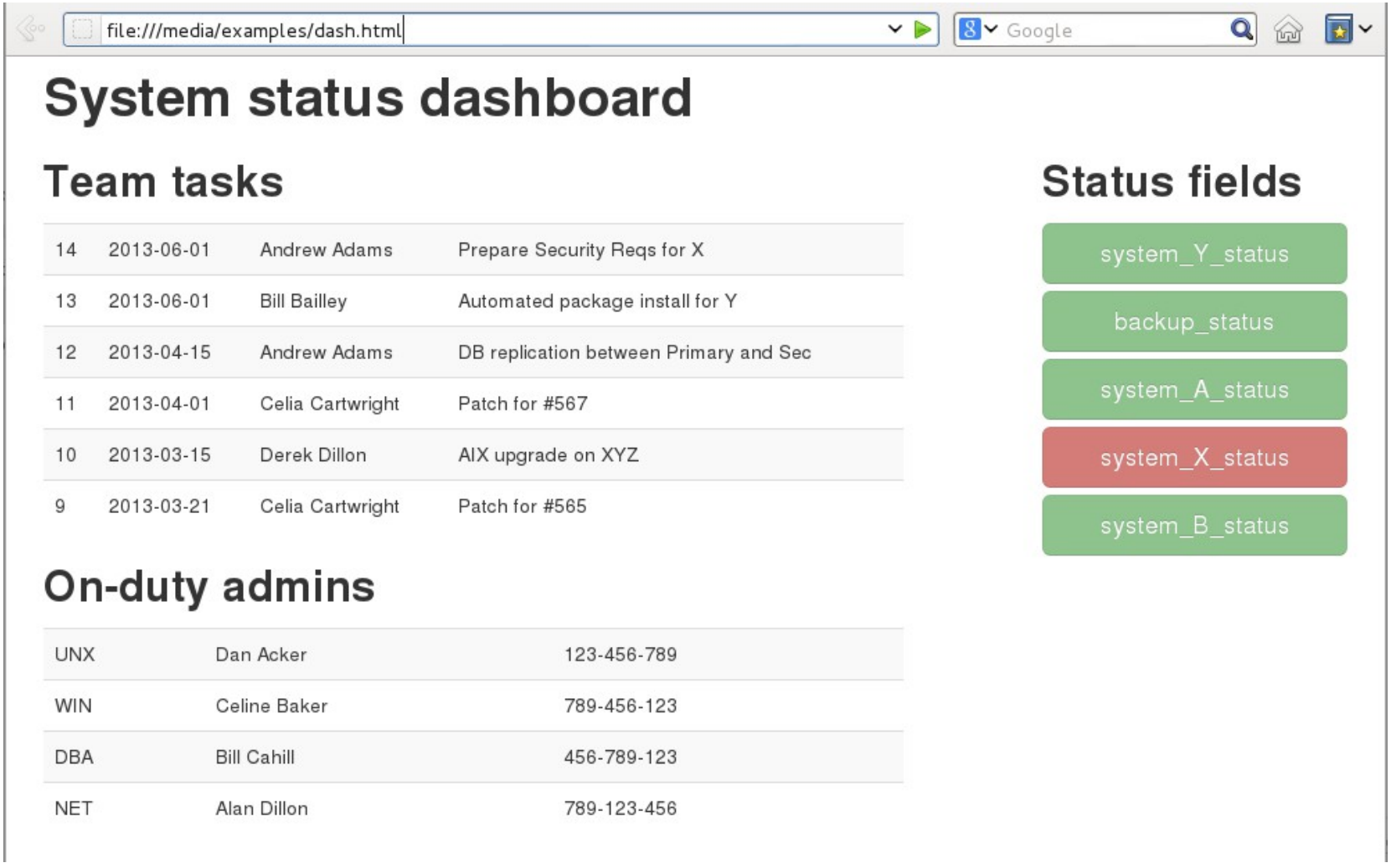
```
# tasks
$cmd = $cfg->val('tables', 'tasks');
$csvstring = `perl $cmd`;
my $tbl_tasks = Text::CSV::Slurp->load(string=>$csvstring, sep_char=>';');
$template->param('tasks' => $tbl_tasks);

# status fields
my @sts_fields;
for my $item ($cfg->Parameters('status')) {
    $cmd = $cfg->val('status',$item);
    my $value = `perl $cmd`;
    # danger and success correspond to bootstrap CSS class names
    $value = ( $value == 0 ) ?  'danger' : 'success';

    push @sts_fields, {'Item' => $item, 'Value' => $value};
}
$template->param('status' => \@sts_fields);

print $template->output();
```

FINAL RESULT



End remarks

OTHER API BINDINGS

Ruby:

`github.com/red-tux/zbxapi`

Python:

`github.com/gescheit/scripts/tree/master/zabbix`

Everything else:

`www.zabbix.org/wiki/Docs/api/libraries`

RUBY EXAMPLE

```
#!/usr/bin/ruby

require "rubygems"
require "zbxapi"
require "pp"

@server_url="http://zabbix.example.com/zabbix"
@login_user="admin"
@login_pass="zabbix"

@proxy_host="localhost"
@proxy_port=3128

zabbix = ZabbixAPI.new(@server)
zabbix.set_proxy(@proxy_host,@proxy_pass)
zabbix.login(@login_user,@login_pass)

hosts = zabbix.host.get

pp hosts
```

SOME USEFUL TOOLS

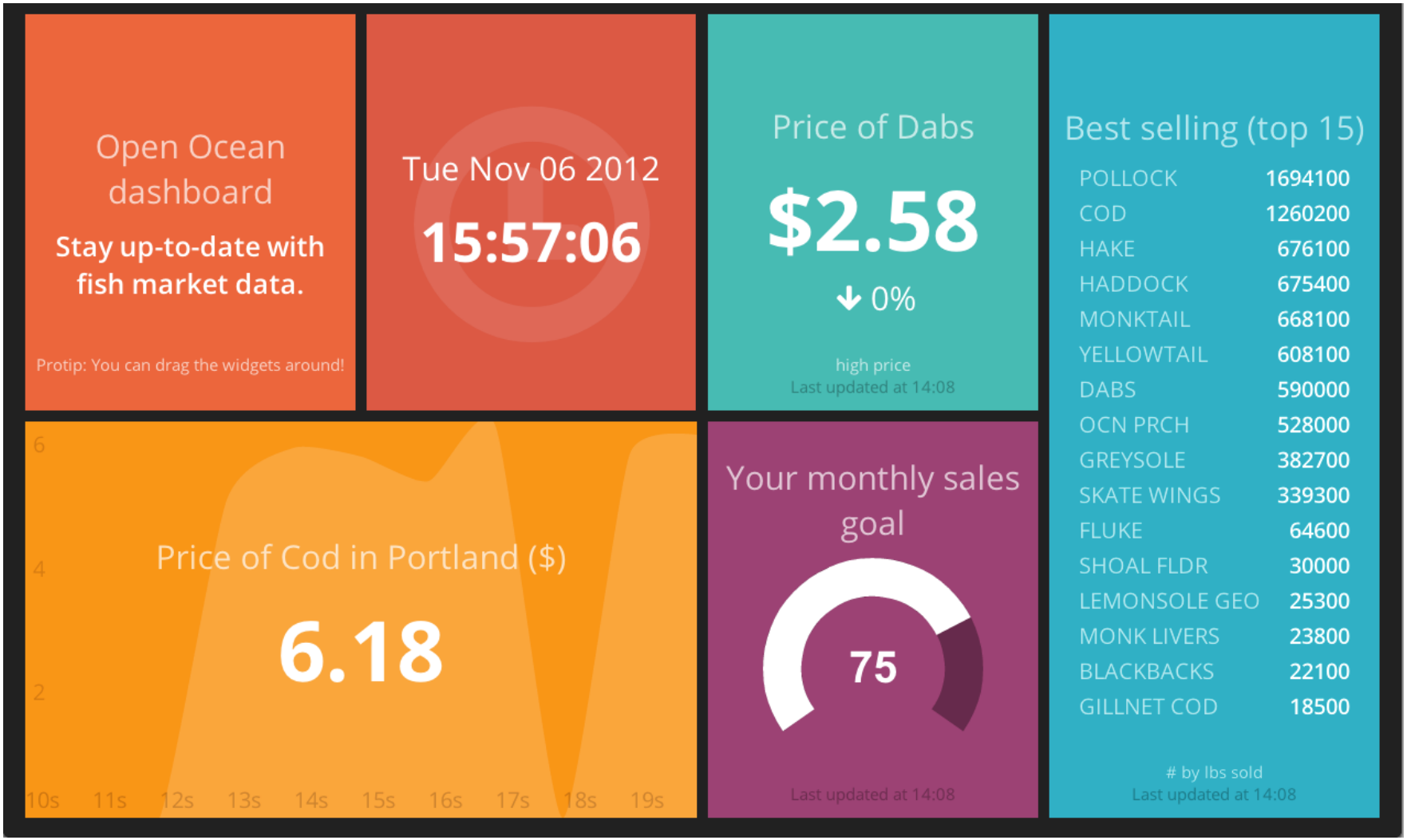
Twitter Bootstrap:

`http://getbootstrap.com/getting-started/`

Dashing:

`http://shopify.github.io/dashing/`

WHAT DASHING LOOKS LIKE



WHAT DASHING FEELS LIKE


```
curl -d '{ "auth_token": "YOUR_AUTH_TOKEN", \
  "current": 100 }' \
  http://localhost:3030/widgets/karma
```

Any questions?

Thank you!