# Smashing Dashboard

## Smashing, the spiritual successor to Dashing, is a Sinatra based framework that lets you build dashboards.

It’s Home Page and Wiki page are:

<https://smashing.github.io/>

<https://github.com/Smashing/smashing/wiki>

Smashing is released under a MIT license:

https://github.com/Smashing/smashing/blob/master/MIT-LICENSE

## Getting Started

Assuming an internet connected system running a Linux derivative (i.e. Ubuntu) with Ruby (>= 2.0) installed:

$ gem install smashing

$ smashing new dash01

$ cd dash01

$ bundle

$ smashing start

Point your browser at localhost:3030 to see the sample dashboard.

**Our Implementation**

### Setup

Create a suitable development environment and deployment environment.

If using an air-gapped development environment make sure you install the following packages:

build-essential

ruby ( Ver >= 2.0 )

ruby-dev

Once these are installed you will also need to move the following Gem packages across the air-gap and install them

bundler

smashing

To install Gems across an air gap, firstly, on an Internet connected machine:

$ gem install bundler smashing -i repo --no-rdoc –no-ri

$ cp -r repo/cache /path/to/USB\_drive/gems

Then, having moved the USB stick (or CD) across the air gap:

$ cd /path/to/USB\_drive/gems

$ gem install --force --local \*.gem

You can now create your initial Smashing installation and the templates for the jobs and widgets:

$ smashing new mis01

$ cd mis01

$ rm jobs/\*

$ rm dashboards/sample\*

$ smashing generate dashboard mis01

$ smashing generate widget misalert2

$ smashing generate widget mismeter

$ smashing generate widget mistext

$ smashing generate job fileservers

$ smashing generate job servers

$ smashing generate job terminals

$ smashing generate job users

Note we have deleted the sample jobs and dashboards that were generated by the ‘smashing new mis01’ command as we will not need them.

### The Grid

The Smashing dashboard is based on a grid. In the default dashboard the grid consists of blocks 300 pixels wide by 360 pixels high with a 5 pixel border around each block.

The main dashboard file ‘dashboard/mis01.erb’ places widgets in the grid using these two attributes:

data-row=”1”

data-col=”1”

An the following two attributes are used to define the width and height of the widget in grid units.

data-sizex=”2”

data-sizey=”2”

Our ‘mis01’ dashboard uses a grid of blocks 54 pixels wide by 54 pixels high with a 2 pixel border around each block. However, most of the widget we place on the dashboard will use 2x2 blocks or 4x4 blocks.

To set our grid sizes we need to modify the file ‘assets/javascripts/application.coffee’. Near the top of the file modify the these lines

Dashing.on 'ready', ->  
 Dashing.widget\_margins ||= [5, 5]  
 Dashing.widget\_base\_dimensions ||= [300, 360]  
 Dashing.numColumns ||= 4

to be

Dashing.on 'ready', ->  
 Dashing.widget\_margins ||= [2, 2]  
 Dashing.widget\_base\_dimensions ||= [54, 54]  
 Dashing.numColumns ||= 20

### The Widgets

We will make use of three different widgets in the dashboard; each is based on one of the standard widgets supplied with Smashing.

Each Widget consists of three files, for example the ‘misalert2’ widget is made of the the three files:

widgets/misalert2/misalert2.coffee

widgets/misalert2/misalert2.html

widgets/misalert2/misalert2.scss

The HTML file is used to define the basic look of the widget on the dashboard.

The SCSS file gives the widget its’ “look and feel” and may also contain extra classes to change the “look and feel” from its’ default.

The coffee file contains the code that defines the widgets behaviour.

### Misalert2

This widget is based on one developed during the Dashing Tutorial:

<https://github.com/Shopify/dashing/wiki/Dashing-Workshop>

It displays a numeric value with a heading. By default the background colour of the widget is green this can be changed by calling a function to test an assertion. If the result of the test is ‘true’ then an extra class is applied to this instance of the widget; this class can be used to change the background colour.

widgets/misalert2/misalert2.html

<h4 data-bind="title"></h4>

<div data-bind="value"></div>

<p class="updated-at" data-bind="updatedAtMessage"></p>

widgets/misalert2/misalert2.coffee

class Dashing.Misalert2 extends Dashing.Widget

ready: ->

# This is fired when the widget is done being rendered

onData: (data) ->

# Handle incoming data

# You can access the html node of this widget with `

# The following line will 'blink' the widget when new data is received

$(@node).fadeOut().fadeIn()

@accessor 'isTooHigh', ->

@get('value') > 300

@accessor 'isNotZero', ->

@get('value') > 0

widgets/misalert2/misalert2.scss

h4 {

color: #fff;

font-size: 18px;

font-weight: 600;

}

.widget-misalert2 {

background: #0f9;

font-size: 32px;

font-weight: bold;

.updated-at {

color: rgba(0, 0, 0, 0.3);

font-size: 8px;

}

}

.danger {

background: red;

}

.pending {

background: #f00;

}

.stopped {

background: #f00;

}

.loggedin {

background: #1aff1a;

}

.downloading {

background: orange;

}

.outofdate {

background: #c9f;

}

p.updated-at-small {

font-size: 8px;

}

When the ‘misalert2’ widget is used in the main dashboard file the following HTML is required:

<li data-row="3" data-col="3" data-sizex="2" data-sizey="2">

<div data-id="mis01\_users\_pending" data-view="Misalert2"

data-title="Pending" data-addclass-pending="isNotZero"></div>

</li>

In this example an ‘Misalert2’ widget is being used to display a widget with a title of “Pending”. The class “pending”, from ‘misalert2.scss’ will be added to the widget if the function (accessor in ‘misalert2.coffee’) ‘isNotZero()’ returns true. We have given the widget a unique ‘id’ of “ mis01\_users\_pending” and the jobs (see below) can use this ‘id’ to send data to this particular widget.

### Mismeter

This widget is based on the ‘meter’ widget supplied with Smashing.

It displays a numeric value with a gauge, filled to show how close the value is to a defined maximum. It also displays a title and additional text as ‘moreinfo’. By default the background colour of the widget is green. Two thresholds are defined when the widget is used ‘threshold1’ and ‘threshold2’. If the value is below ‘threshold1’ the default background id used, if the value is above ‘threshold1’ but below ‘threshold2’ the background is changed to orange and if the value is above ‘threshold2’ the background is changed to red.

widgets/mismeter/mismeter.html

<h4 class="title" data-bind="title"></h4>

<input class="meter" data-angleOffset=-125 data-angleArc=250

data-bind-data-height="height | default 200"

data-bind-data-width="width | default 200"

data-readOnly=true

data-bind-value="value | shortenedNumber | prepend prefix | append suffix"

data-bind-data-min="min"

data-bind-data-max="max">

<p class="more-info" data-bind="moreinfo"></p>

<p class="updated-at" data-bind="updatedAtMessage"></p>

widgets/mismeter/mismeter.scss

// ----------------------------------------------------------------------

// Sass declarations

//-----------------------------------------------------------------------

$background-color-default: #0f9; // green

$background-color-good: #0f9; // green

$background-color-middle: #ffa500; // orange

$background-color-bad: #f00; // red

$title-color: rgba(255, 255, 255, 1);

$moreinfo-color: rgba(255, 255, 255, 1);

$meter-background: rgba(0, 0, 0, .3);

//-----------------------------------------------------------------------

// Widget-meter2 styles

//-----------------------------------------------------------------------

.widget-mismeter {

background-color: $background-color-default;

input.meter {

background-color: $meter-background;

color: #fff;

font-size: 20px !important;

}

.title {

color: $title-color;

}

.more-info {

color: $moreinfo-color;

}

.updated-at {

color: rgba(0, 0, 0, .3);

font-size: 12px;

}

&.bad {

background-color: $background-color-bad;

}

&.middle {

background-color: $background-color-middle;

}

&.good {

background-color: $background-color-good;

}

}

widgets/mismeter/mismeter.scss

class Dashing.Mismeter extends Dashing.Widget

@accessor 'value', Dashing.AnimatedValue

constructor: ->

super

@observe 'value', (value) ->

$(@node).find(".meter").val(value).trigger('change')

ready: ->

meter = $(@node).find(".meter")

meter.attr("data-bgcolor", meter.css("background-color"))

meter.attr("data-fgcolor", meter.css("color"))

meter.knob()

@onData(this)

onData: (data) ->

# console.log data.value + ' ' + @get('threshold1') + ' ' + @get('threshold2')

if not data.value?

console.log "Remove all"

$(@node).removeClass('good')

$(@node).removeClass('bad')

else if data.value > @get('threshold2')

console.log "Add bad"

$(@node).addClass('bad')

$(@node).removeClass('good')

$(@node).removeClass('middle')

else if (data.value > @get('threshold1')) && (data.value < @get('threshold2'))

console.log "Add middle"

$(@node).addClass('middle')

$(@node).removeClass('bad')

$(@node).removeClass('good')

else

console.log "Add good"

$(@node).addClass('good')

$(@node).removeClass('bad')

$(@node).removeClass('middle')

When the ‘mismeter’ widget is used in the main dashboard file the following HTML is required:

<li data-row="6" data-col="7" data-sizex="4" data-sizey="4">

<div data-id="mis01\_dkrhost01\_loadavg\_1min"

data-view="Mismeter"

data-title="Load Average"

data-threshold1="0.4"

data-threshold2="0.7"

data-moreinfo="1 min"

data-height="180" data-width="162"

data-min="0" data-max="10" data-suffix="%"></div>

</li>

### Mistext

This widget is based on the ‘misalert2’ widget but does not display a value; it simply displays a value and is used to put Titles on the dashboard.

widgets/mistext/mistext.html

<h4 class="title" data-bind="title"></h4>

widgets/mistext/mistext.scss

.widget-mistext {

background: #3498db;

}

widgets/mistext/mistext.scss

class Dashing.Mistext extends Dashing.Widget

ready: ->

# This is fired when the widget is done being rendered

onData: (data) ->

# Handle incoming data

# You can access the html node of this widget with `@node`

# Example: $(@node).fadeOut().fadeIn() will make the node flash each time data comes in.

## Jobs

Jobs are used by Smashing to send data to widgets. A job can get its’ data from any where and send it to any number of widgets, they are scheduled to run at regular intervals. In our case the data will come from the Monitoring service via HTTPS requests.

### Fileservers Job

This job makes an HTTPS request to the following URL:

/monitor/mis/fileservers

It expects to get back JSON of the following form:

{ fileserver1: 0, fileserver2: 0}

# Use this as http template

# https://gist.github.com/charlesrg/f8246808b73f872d631e41336d1b05b7

# Could use this as https template

# https://gist.github.com/pszypowicz/bc07528ed7ae79bf49aa

require 'net/https'

require 'json'

cert = File.read("./certs/aspire.crt")

key = File.read("./certs/aspire-nopw.key")

host = 'aspire.local'

port = '3010'

http = Net::HTTP.new(host, port)

http.use\_ssl = true

http.cert = OpenSSL::X509::Certificate.new(cert)

http.key = OpenSSL::PKey::RSA.new(key)

http.ca\_file = './certs/CA1.crt'

http.verify\_mode = OpenSSL::SSL::VERIFY\_NONE

terminals\_uri = '/monitor/mis/fileservers'

data = { fileserver1: 0, fileserver2: 0}

SCHEDULER.every '5s', :first\_in => 0 do |job|

dataprev = data

resp = http.get(terminals\_uri)

next unless '200'.eql? resp.code

data = JSON.parse(resp.body)

if data["fileserver1"] != dataprev["fileserver1"]

send\_event('mis01\_fileserver\_1', { value: data["fileserver1"]} )

end

if data["fileserver2"] != dataprev["fileserver2"]

send\_event('mis01\_fileserver\_2', { value: data["fileserver2"]} )

end

end

### Servers Job

This job makes an HTTPS request to the following URL:

/monitor/mis/server

It expects to get back JSON of the following form :

[ {"name":"dkrhost01",

"uptime":1,

"cpuUsage":0.28,

"containers": {"running":17,"stopped":1},

"loadAverages":{"1min":0.38,"5min":0.08,"15min":0.57} }  
]

Where the returned array contains an entry per server.

require 'net/https'

require 'json'

cert = File.read("./certs/aspire.crt")

key = File.read("./certs/aspire-nopw.key")

host = 'aspire.local'

port = '3010'

http = Net::HTTP.new(host, port)

http.use\_ssl = true

http.cert = OpenSSL::X509::Certificate.new(cert)

http.key = OpenSSL::PKey::RSA.new(key)

http.ca\_file = './certs/CA1.crt'

http.verify\_mode = OpenSSL::SSL::VERIFY\_NONE

server\_uri = '/monitor/mis/server'

SCHEDULER.every '5s', :first\_in => 0 do |job|

resp = http.get(server\_uri)

next unless '200'.eql? resp.code

server\_data = JSON.parse(resp.body)

# puts server\_data.inspect

server\_data.each { |server|

id\_base = "mis01\_" + server["name"]

send\_event(id\_base + '\_running', { value: server["containers"]["running"]} )

send\_event(id\_base + '\_stopped', { value: server["containers"]["stopped"]} )

send\_event(id\_base + '\_uptime', { value: server["uptime"]} )

send\_event(id\_base + '\_cpuusage', { value: server["cpuUsage"]} )

send\_event(id\_base + '\_loadavg\_1min', { value: server["loadAverages"]["1min"]} )

send\_event(id\_base + '\_loadavg\_5min', { value: server["loadAverages"]["5min"]} )

send\_event(id\_base + '\_loadavg\_15min', { value: server["loadAverages"]["15min"]} )

}

end

### Terminals Job

This job makes an HTTPS request to the following URL:

/monitor/mis/terminals

It expects to get back JSON of the following form :

{ provisioned: 0, active: 0, loggedIn: 0, outofdate: 0}

require 'net/https'

require 'json'

cert = File.read("./certs/aspire.crt")

key = File.read("./certs/aspire-nopw.key")

host = 'aspire.local'

port = '3010'

http = Net::HTTP.new(host, port)

http.use\_ssl = true

http.cert = OpenSSL::X509::Certificate.new(cert)

http.key = OpenSSL::PKey::RSA.new(key)

http.ca\_file = './certs/CA1.crt'

#Not secure is to leave it NONE, VERIFY\_PEER is preffered.

http.verify\_mode = OpenSSL::SSL::VERIFY\_NONE

terminals\_uri = '/monitor/mis/terminals'

data = { provisioned: 0, active: 0, loggedIn: 0, outofdate: 0}

SCHEDULER.every '5s', :first\_in => 0 do |job|

# http = Net::HTTP.new('192.168.0.240', '3000')

# resp = http.request(Net::HTTP::Get.new("/monitor/mis/terminals"))

# next unless '200'.eql? resp.code

dataprev = data

resp = http.get(terminals\_uri)

next unless '200'.eql? resp.code

data = JSON.parse(resp.body)

if data["provisioned"] != dataprev["provisioned"]

send\_event('mis01\_terminals\_deployed', { value: data["provisioned"]} )

end

if data["active"] != dataprev["active"]

send\_event('mis01\_terminals\_active', { value: data["active"]} )

end

if data["loggedIn"] != dataprev["loggedIn"]

send\_event('mis01\_terminals\_loggedIn', { value: data["loggedIn"]} )

end

if data["outOfDate"] != dataprev["outOfDate"]

send\_event('mis01\_terminals\_outofdate', { value: data["outOfDate"]} )

end

end

### Users Job

This job makes an HTTPS request to the following URL:

/monitor/mis/users

It expects to get back JSON of the following form :

{ registered: 0, pending: 0}

require 'net/https'

require 'json'

cert = File.read("./certs/aspire.crt")

key = File.read("./certs/aspire-nopw.key")

host = 'aspire.local'

port = '3010'

http = Net::HTTP.new(host, port)

http.use\_ssl = true

http.cert = OpenSSL::X509::Certificate.new(cert)

http.key = OpenSSL::PKey::RSA.new(key)

http.ca\_file = './certs/CA1.crt'

#Not secure is to leave it NONE, VERIFY\_PEER is preffered.

http.verify\_mode = OpenSSL::SSL::VERIFY\_NONE

terminals\_uri = '/monitor/mis/users'

data = { registered: 0, pending: 0}

SCHEDULER.every '5s', :first\_in => 0 do |job|

# http = Net::HTTP.new('192.168.0.240', '3000')

# resp = http.request(Net::HTTP::Get.new("/monitor/mis/users"))

# next unless '200'.eql? resp.code

dataprev = data

resp = http.get(terminals\_uri)

next unless '200'.eql? resp.code

data = JSON.parse(resp.body)

if data["registered"] != dataprev["registered"]

send\_event('mis01\_users\_registered', { value: data["registered"]} )

end

if data["pending"] != dataprev["pending"]

send\_event('mis01\_users\_pending', { value: data["pending"]} )

end

end

## The Layout

Finally Smashing pulls all the components together in a dashboard file:

mis01/dashboards/mis01.erb

This file adds widgets to the grid and uses a series of data binds to provide the information needed by the widgets and jobs. The file is too large to reproduce in full here.

The top part of the dashboard

<div class="gridster">

<ul>

<!-- Dashboard Title -->

<li data-row="1" data-col="1" data-sizex="18" data-sizey="1">

<div data-id="text\_dashboard\_title" data-view="Mistext" data-title="System Dashboard"></div>

</li>

<!-- Users -->

<li data-row="2" data-col="1" data-sizex="4" data-sizey="1">

<div data-id="text\_users" data-view="Mistext" data-title="Users"></div>

</li>

<li data-row="3" data-col="1" data-sizex="2" data-sizey="2">

<div data-id="mis01\_users\_registered" data-view="Misalert2" data-title="Registered"></div>

</li>

<li data-row="3" data-col="3" data-sizex="2" data-sizey="2">

<div data-id="mis01\_users\_pending" data-view="Misalert2" data-title="Pending" data-addclass-pending="isNotZero"></div>

</li>

<!-- Spacer -->

<li data-row="2" data-col="5" data-sizex="1" data-sizey="3">

<div data-id="spacer1" data-view="Mistext"></div>

</li>

<!-- Terminals -->

<li data-row="2" data-col="6" data-sizex="8" data-sizey="1">

<div data-id="text\_terminals" data-view="Mistext" data-title="Terminals"></div>

</li>

<li data-row="3" data-col="6" data-sizex="2" data-sizey="2">

<div data-id="mis01\_terminals\_deployed" data-view="Misalert2" data-title="Deployed"></div>

</li>

<li data-row="3" data-col="8" data-sizex="2" data-sizey="2">

<div data-id="mis01\_terminals\_active" data-view="Misalert2" data-title="Active"></div>

</li>

<li data-row="3" data-col="10" data-sizex="2" data-sizey="2">

<div data-id="mis01\_terminals\_loggedIn" data-view="Misalert2" data-title="Logged In" data-addclass-loggedin="isNotZero"></div>

</li>

<li data-row="3" data-col="12" data-sizex="2" data-sizey="2">

<div data-id="mis01\_terminals\_outofdate" data-view="Misalert2" data-title="Out of Date" data-addclass-outofdate="isNotZero"></div>

</li>

<!-- Spacer -->

<li data-row="2" data-col="14" data-sizex="1" data-sizey="3">

<div data-id="spacer2" data-view="Mistext"></div>

</li>

And the first Docker Host

<!-- Hosts Banner -->

<li data-row="5" data-col="1" data-sizex="18" data-sizey="1">

<div data-id="text\_hosts\_title" data-view="Mistext" data-title="Docker Hosts"></div>

</li>

<!-- DKRHOST01 -->

<li data-row="6" data-col="1" data-sizex="2" data-sizey="4">

<div data-id="text\_dkrhost01" data-view="Mistext" data-title="DKRHOST01"></div>

</li>

<li data-row="6" data-col="3" data-sizex="2" data-sizey="2">

<div id="alertSmaller" data-id="mis01\_dkrhost01\_running" data-view="Misalert2" data-title="Running Containers"></div>

</li>

<li data-row="8" data-col="3" data-sizex="2" data-sizey="2">

<div id="alertSmaller" data-id="mis01\_dkrhost01\_stopped" data-view="Misalert2" data-title="Stopped Containers" data-addclass-stopped="isNotZero"></div>

</li>

<li data-row="6" data-col="5" data-sizex="2" data-sizey="2">

<div data-id="mis01\_dkrhost01\_uptime" data-view="Misalert2" data-title="Uptime" ></div>

</li>

<li data-row="8" data-col="5" data-sizex="2" data-sizey="2">

<div data-id="mis01\_dkrhost01\_cpuusage" data-view="Misalert2" data-title="CPU Usage" ></div>

</li>

<li data-row="6" data-col="7" data-sizex="4" data-sizey="4">

<div data-id="mis01\_dkrhost01\_loadavg\_1min" data-view="Mismeter" data-title="Load Average" data-threshold1="0.4" data-threshold2="0.7" data-moreinfo="1 min" data-height="180" data-width="162" data-min="0" data-max="10" data-suffix="%"></div>

</li>

<li data-row="6" data-col="11" data-sizex="4" data-sizey="4">

<div data-id="mis01\_dkrhost01\_loadavg\_5min" data-view="Mismeter" data-title="Load Average" data-threshold1="0.4" data-threshold2="0.7" data-moreinfo="5 min" data-height="180" data-width="162" data-min="0" data-max="100" data-suffix="%"></div>

</li>

<li data-row="6" data-col="15" data-sizex="4" data-sizey="4">

<div data-id="mis01\_dkrhost01\_loadavg\_15min" data-view="Mismeter" data-title="Load Average" data-threshold1="0.4" data-threshold2="0.7" data-moreinfo="15 min" data-height="180" data-width="162" data-min="0" data-max="100" data-suffix="%"></div>

</li>

## Deployment Via Docker

The Smashing dashboard can be run from a Docker container. Our approach is to create a base Docker Smashing image on an Internet connected machine and transfer this image across the air gap.

Next the base image is customised for the intended environment.

Finally a container is created and run.

### Part 1 – Creating a Base Smashing Docker Image

On an Internet connected machine use the following ‘Dockerfile’ to create a base Smashing image from Ubuntu base image. When the image is complete save it and transfer it to the final environment.

### Part 2 – Customising the Base Smashing Docker Image

Now customize the image and add the Smashing source files using this ‘Dockerfile’.

### Part 3 – Deploying a Containers

Finally create a container and run it using this shell script.

## Security

This implementation of Smashing uses HTTPS to connect to the Monitoring service.

However, all connections to the Smashing dashboard are HTTP and are un-authenticated.

Smashing does not currently support HTTPS.

Smashing does support some simple plain text authentication methods and has a simple plug-in environment into which you can add your own authentication.

Researching Smashing on the Internet the most common suggest for over-coming these limitations is to place your Smashing implementation (container) behind an authenticating reverse proxy that supports HTTPS, such as Nginx.