Exploiting VCS for fun and profit

Anders Nordby <anders@fupp.net>
2016-11-03

https://github.com/anordby/

Anders Nordby?

- Was a sysadmin for Aftenposten/Schibsted IT. Handled Varnish caching for aftenposten.no, bt.no, finn.no etc. Did paywall setup, mobile device handling using Device Atlas, WURFL and homegrown regexps.
- Was involved in discussions leading to Varnish before it was started. Initial idea was to bribe someone to fix Squid ...
- Sent a lot of bug reports when Varnish was new trying to use it with large datasets for finn.no, which frequently made it crash.
- Is a sysadmin/webops guy for Møller Group.
- Recovering Perl addict, now a Rubyist.

MøllerGruppen

- One of North Europes leading car groups with 4000 employees
- Norways biggest car importer
- 67 car dealers. Also a number of connected, independent dealers cooperating with and using IT services from Møller
- Core operations are import, sales, finance and service of Volkswagen, Audi & Skoda cars
- Operates in Norway, Sweden, Lithuania, Estonia, Latvia. Expanding in Sweden
- Headquarters and IT centralized at Frysja in Oslo, Norway

VCS. The unsung hero of the Varnish ecosystem?

- Varnish Custom Statistics is a near realtime statistics system for Varnish giving buckets of your favourite Varnish data aggregated in JSON format. Ready to use from your favourite programming language. Familiar to everyone?
- Not free software. Requires a license from Varnish Software.
- Sharing of info and 3rd party software building on VCS is lacking or not too common. There is no community for VCS? Can we improve and increase its popularity? Only VCS is currently a good reason to become a Varnish Plus customer!

Things I did with VCS before

- Munin graphs showing data per website: bytes transferred, cache hit ratio, number of requests, status codes, time to firste byte.
- Ugly HTML based "dashboard" showing most popular URLs, uncached URLs, URLs with 4xx or 5xx responses etc.
- Mail based alerts to developers when number of requests per (base) URL reached a certain threshold. That way I did not have to pester them manually each time someone introduced a cache-wise broken URL (popular URL with bad/no caching), they could fix it themselves from the report.

Varnish at Møller

- Not a big news site, which Varnish was initially developed for. We sell and give service for cars.
- Varnish is used to handle access to, statistics for and load balancing for a growing list of APIs.
- Focus for us is stability, load balancing, good monitoring, continous deployment, automation.
- VCLs are generated by Ruby ERB templates in Puppet.
- Production & staging environment has 214 backends, test & development has 194.

VCS at Møller

- I was a new employee setting up a lot of new systems unknown to other people.
- The need to show status clearly on dashboards screens, internal web pages was clear to me.
- I had already gotten started with a Ruby based dashboard tool Dashing.io, which makes it easy to build nice widget based dashboards and run them on your own servers and big screen TVs. It was time to get data from VCS there.

Getting most eager/bullyish consumer IPs on the dashboard

- Varnish gets traffic from a F5 hardware load balancer that sets X-Forwarded-For with the original client IP. So lets use it in VCL (in vcl_recv):

```
if (req.http.X-Forwarded-For) {
  # We trust it
  set req.http.X-Client-IP = regsub(req.http.X-Forwarded-For, "(,|:).*", "");
} else {
  set req.http.X-Client-IP = client.ip;
}
```

- And log it (in vcl_deliver):

```
std.log("vcs-key: hostip:" + req.http.host + " " + req.http.X-Client-IP);
```

Preparing table widget for Dashing

```
#! /usr/bin/ruby
require 'httparty'
require 'json'
require 'ipaddress'
require 'resolv'

title = "Top API consumers"
url = "http://no000010sapip0.moller.local:6555/match/%5Ehostip:/top"

headers = [{
        "style" => "font-size: x-small",
        "cols" => [
                {"value"=>"IP"},
                {"value"=>"Req/min"},
        ]
}]
rows = []
```

More preparations

```
def resolvip (iptxt)
  if IPAddress.valid?(iptxt)
    hname = Resolv.getname(iptxt)
  else
    hname = iptxt
  end
  return hname
end

response = HTTParty.get(url)
json = JSON.parse(response.body)
```

Preparing the rows

```
json.each_pair do |key, val|
  key = key.gsub(/^hostip:api.moller.local /, "")
  key = resolvip(key)
  style = "font-size: x-small"
  # More than 1000 requests? Color me yellow!
  if val >= 1000
    style += "; background-color:#ffff00"
  end
  rows.push({
    "style" => style,
    "cols"  => [
      {"value" => key},
      {"value" => val},
    ]})
end
```

Posting the data to Dashing

```
response =
HTTParty.post("http://dashing.moller.foo/widgets/v
cs_top_api_consumers",
  :body => {
    auth_token: "XXXX",
    hrows: headers,
    rows: rows,
    title: title
  }.to_json)
```

VCS data in Dashing.io

Top API consumers

IP	Req/min
was6	3085
mbrp1	82
mbrp0	76
mbap0	23
was7	11
was8	9
mbap1	1

Last updated at 22:02

Top API URLs

URL	Req/min
/AuthorizationEndpoint/applications/KISS/latest-authorization-change	3078
/ControlDataEndpoint/countries/NOR/dealers/importerCode/A/importerDealerNumber/001/productionType/3?includeTestDealers=true	30
/ControlDataEndpoint/countries/NOR/dealers/importerCode/A/importerDealerNumber/001/productionType/2?includeTestDealers=true	30
/ControlDataEndpoint/countries/NOR/dealers/importerCode/A/importerDealerNumber/001/productionType/1?includeTestDealers=true	30
/ControlDataEndpoint/countries/NOR/dealers/importerCode/C/importerDealerNumber/951/productionType/1?includeTestDealers=true	30
/ReportEndpoint/invoice/country/N/dealer/185/year/2014/invoiceNo/6821716	9
/ControlDataEndpoint/countries/NOR/dealerContactInfo/108/all/	5
/WorkshopEndpoint/workshopSession/countries/NOR/dealerNumber/365/year/2016/workshopSessionNumber/519227	2
/OnlineBookingEndpoint/onlineBooking/country/NOR/dealerNumber/137/workshopSessionNumber/578877/year/2016	2
/CarSystemEndpoint/mnetVehicle/countryCode/NOR/vin/TMBLG9NE5G0053577	2

Last updated at 22:02

Top API 5xx response URLs

URL	Req/min
No problems found.	

Last updated at 22:05

Top API modules

Module	Req/min
AuthorizationEndpoint	3008
ControlDataEndpoint	33
CarSystemEndpoint	16
WorkshopPlanningEndpoint	11
WorkshopEndpoint	8
ReportEndpoint	7
consolidatedcustomermaster	7
PackageMasterService	4
OnlineBookingIntegrationEndpoint	3
CarModelEndpoint	3

Last updated at 22:06

More VCS data in Dashing.io

Slowest response per API

URL	Seconds
/PackageMasterService/packageinformation/countries/NOR/dealer/001/vin/WVWZZZ1KZAW570770/nextservice	0.789
/WorkshopEndpoint/workshopHistory/countries/NOR/carHistory/vin/WVWZZZAUZFW903795/partyId/154290?getP	0.652
/PostMasterEndpoint/api/smssurvey	0.445
/consolidatedcustomermaster/Party/154290?depth=3&carRelationFilter=Active&allCarRelations=true&prefe	0.328
/EA189Endpoint/api/ea189Info/countries/NOR/vin/TMBLD75L4B6021317	0.157
/OnlineBookingIntegrationEndpoint/api/bookings/countries/NOR/vin/WVWZZZ1KZAW570770	0.102
/CustomerPortal/portal/countries/NOR/dealers/820	0.078
/ControlDataEndpoint/countries/NOR/dealerContactInfo/820/all/	0.068
/WorkshopPlanningEndpoint/countries/NOR/appointment/regnumber/EL57330/fromDate/2016-11-02	0.047
/OnlineBookingEndpoint/onlineBooking/country/NOR/dealerNumber/210/status/1	0.046
/VehicleControlEndpoint/countries/NOR/nextVehicleControl/registrationNumber/PX52720	0.016
/CarSystemEndpoint/mnetVehicle/countryCode/NOR/licenseNumber/XA18165	0.013

Last updated at 22:08

Slowest average response per API

Module	Requests	Seconds
CarInformationEndpoint	6	0.82
BackboneIntegrationEndpoint	6	0.646
PostMasterEndpoint	1	0.445
AzureFileHandlerEndpoint	2	0.354
WorkshopEndpoint	20	0.199
EA189Endpoint	1	0.157
consolidatedcustomermaster	7	0.144
OnlineBookingIntegrationEndpoint	3	0.098
PackageMasterService	19	0.084
CustomerPortal	55	0.043
WorkshopPlanningEndpoint	3	0.033
ControlDataEndpoint	56	0.032

Last updated at 22:08

VCS and RRDTool

- In order to make more trend data for each of our APIs available I wrote Ruby scripts to make RRDTool graphs and combined it with a simple Rails app to browse/view
- The part using data from VCS is easy. I've put the key JSON data for each API module in a hash. To graph the cache hit ratio:

```
if $data["apis"][api]["cached"] == true
  n_req = $data["apis"][api]["json"]["key"]["n_req"].to_f
  n_miss = $data["apis"][api]["json"]["key"]["n_miss"].to_f
  cachehitratio = (n_req-n_miss)/n_req*100
  graph_gauge("api_#{api}_cachehitratio", api, cachehitratio.round(2),
"cachehitratio", "%")
end
```

RRDTool?

- But defining a graph is cumbersome, needs a lot of details and is inflexible when it comes to changes:

```
def graph_gauge (graph, api, number, title, type)
  rrd = "#{$rrdsubdir}/#{graph}.rrd"
  step=60
  dsname="requests"
  heartbeat=300
  puts "Updating graph #{graph} with value #{number}"

  if not File.exist?(rrd)
    RRD.create(
            rrd,
            "--start", "N",
            "--step", "#{step}",
            "DS:#{dsname}:GAUGE:#{heartbeat}:0:U",
```

RRDTool updates?

```
            "RRA:AVERAGE:0.5:1:129600", # 1 minute 90 days
            "RRA:AVERAGE:0.5:60:12960", # 1 hour 18 months
            "RRA:AVERAGE:0.5:60:12960" # 1 day 10 years
    )
  end
  RRD.update(rrd, "N:#{number}")
  graph_gauge_period(graph, api, rrd, dsname, title, "daily", "end-24h", type)
  graph_gauge_period(graph, api, rrd, dsname, title, "weekly", "end-7d", type)
  graph_gauge_period(graph, api, rrd, dsname, title, "monthly", "end-1M",
type)
  graph_gauge_period(graph, api, rrd, dsname, title, "yearly", "end-1y", type)
end
```

RRDTool Graphing

```
def graph_gauge_period (graph, api, rrd, dsname, title, period, start, type)
        png = $graphdir + "/#{graph}-#{period}.png"
        RRD.graph(png,
                "--pango-markup",
                "--title", "#{title} #{period}",
                "--vertical-label", "#{api}",
                "--end", "now",
                "--start", "#{start}",
                "--interlace",
                "--base=1000",
                "--imgformat", "PNG",
                "--width=#{$width}",
                "--height=#{$height}",
                "--full-size-mode",
                "--watermark", "#{$updated}",
                "--lower-limit", "0",
                "DEF:#{dsname}=#{rrd}:#{dsname}:AVERAGE",
                "AREA:#{dsname}#7648ec:#{type}:STACK",
                "LINE:#{dsname}#4d18e4",
                "GPRINT:#{dsname}:LAST:Cur\\:%8.2lf %s",
                "GPRINT:#{dsname}:AVERAGE:Avg\\:%8.2lf %s"
        )
end
```

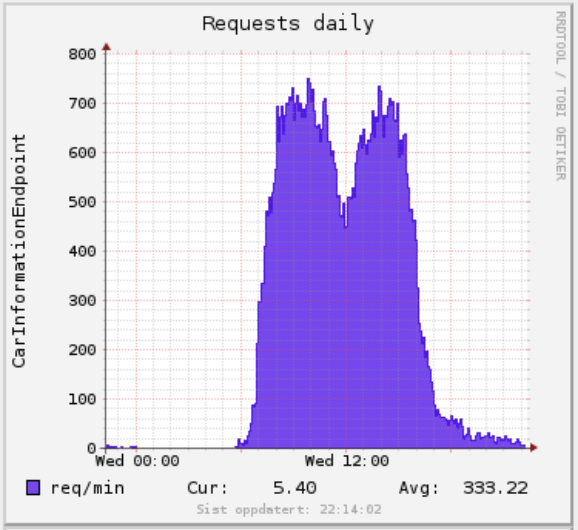
- AppAdminEndpoint
- AuditLogEndpoint
- AuthorizationEndpoint
- AutosysIntegrationEndpoint
- AzureFileHandlerEndpoint
- BackboneIntegrationEndpoint
- CarInformationEndpoint
- CarModelEndpoint
- CarSystemEndpoint
- consolidatedcustomermaster
- ControlDataEndpoint
- CustomerEndpoint
- CustomerPortal
- DealerLocationEndpoint
- EA189Endpoint
- mnet_request_transformer
- OnlineBookingEndpoint
- OnlineBookingIntegrationEndpoint
- PackageMasterService
- PostMasterEndpoint
- ReportEndpoint
- SMSRest
- SurveyEndpoint
- sva
- sva_mnet_communication
- TelematicsService
- TimeRegistrationEndpoint
- Translation
- TranslationEndpoint
- VehicleControlEndpoint
- VehicleMileageHistory
- WarehouseOrderEndpoint
- WorkshopEndpoint
- WorkshopPlanningEndpoint
- Total

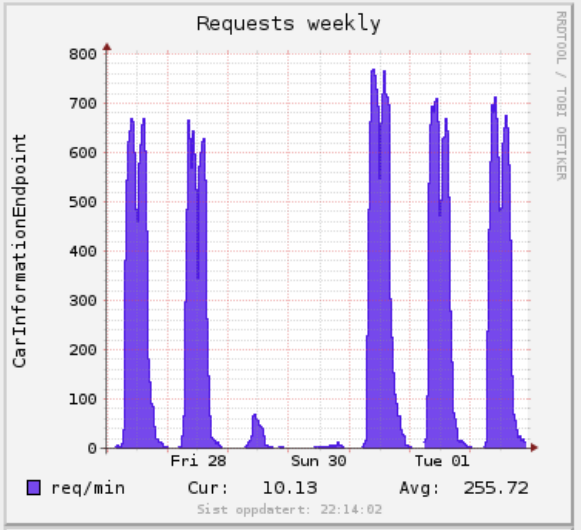
Møller IT API

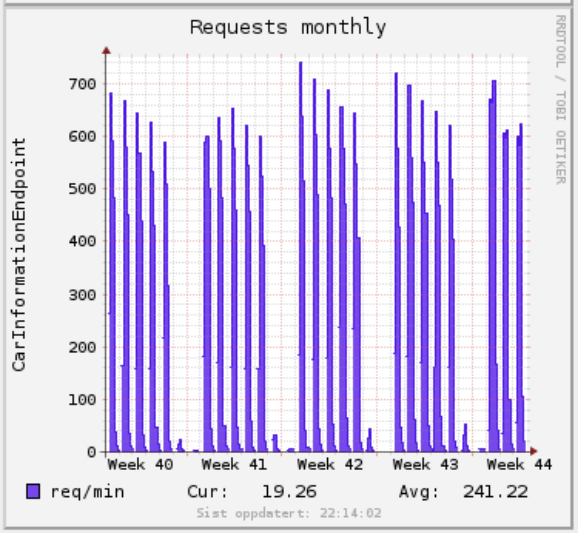
Production statistics/trends.

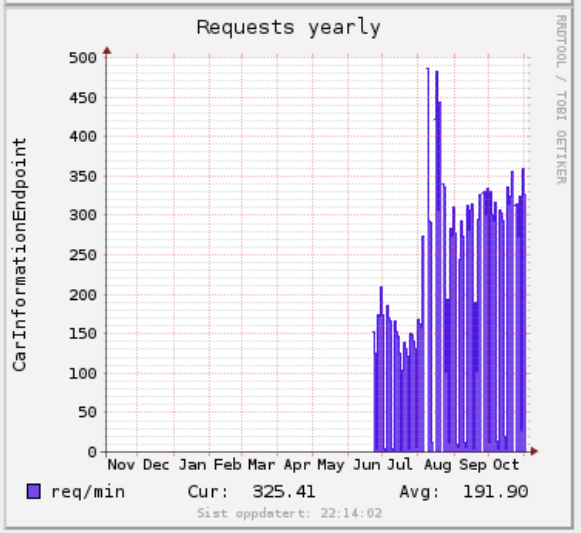
Requests | Response time | Response size | Slow time | Cache hit ratio

CarInformationEndpoint









anders.nordby@moller.no

Database load issues

- For some days in august database load was high on the agenda for everyone in IT, we were getting too many requests in DB2 pointing to the same API servers and we did not have any good overview of the source or nature of the requests.
- I felt ashamed and unhelpful, not really having anything useful to help correlating HTTP traffic with the load.

Something more needed for the graphs

Considering my previous experience with VCS and graphs in Munin / RRDTool, I knew wanted to do this different. I made a list of what I wanted:

- A data oriented cloud based graphing system that I could just readily use and not have to fiddle with
- Which did not require me to preconfigure the datasets and metrics for each graph in detail
- That could correlate a lot of constantly changing metrics on the same graph, to spot interesting peaks and traffic patterns.

Enter DataDog

- A wellknown commercial SaaS-based monitoring and analytics platform
- Has a well documented API to submit custom data, set up graphs and timeboards and a lot more
- Annoyingly forces you to install an agent even when you don’t want one and just want to use custom metrics

Pulling data from VCS with a reusable Ruby helper function

```
def jvcs (url)
        baseurl = "http://no000010sapix0.moller.foo:6555"
        vcsurl = "#{baseurl}#{url}"
        begin
                response = HTTParty.get(vcsurl)
        rescue Exception => e
                puts "Error fetching data from VCS URL #{vcsurl}: #{e.to_s}"
        end
        if response.code != 200
                puts "Unexpected response #{response.code.to_s} fetching data from VCS URL #{vcsurl}."
                return {}
        end
        begin
                json = JSON.parse(response.body)
        rescue Exception => e
                puts "Error parsing JSON data from VCS URL #{vcsurl}: #{e.to_s}"
        end
        if json.nil?
                return {}
        else
                return json
        end
end
```

Collecting and submitting to DataDog

- For information about submitting custom metrics, see http://docs.datadoghq.com/api/#metrics
- To see multiple metrics on one graph, send data with the same data point but with different tag value for what is different. More info about tags: https://help.datadoghq.com/hc/en-us/articles/204312749-Getting-started-with-tags
- I want to collect info for each API, so I log this in VCL:

```
std.log("vcs-key: apimodule-prod:" + req.http.X-Moller-api-module);
```

- Question for the audience: why do I bother with threads?

Collect & submit to DataDog

```
require 'dogapi'
environments[env]["apiconfig"]["moller::api"]["services"].each_pair do |api,mdata|
  thr = Thread.new {
  # For each API get VCS data, send to DataDog..
  apidata = jvcs("/key/apimodule-#{env}:#{api}", env)[0]
  if apidata["n_req"] > 0
    dog.emit_point("api.#{env}.mod.#{api}.resptime", apidata["ttfb_miss"], :host => hostname)
    dog.emit_point("api.#{env}.resptime_perapi", apidata["ttfb_miss"], :host => hostname, :tags => ["api:#{api}"])
    dog.emit_point("api.#{env}.mod.#{api}.resp_error", apidata["resp_5xx"], :host => hostname, :tags => ["code:5xx"])
    dog.emit_point("api.#{env}.resp_error_perapi", apidata["resp_5xx"], :host => hostname, :tags => ["api:#{api}"])
  end
  } # thread end
  threadlist.push(thr)
end
```

Collect & submit to DataDog

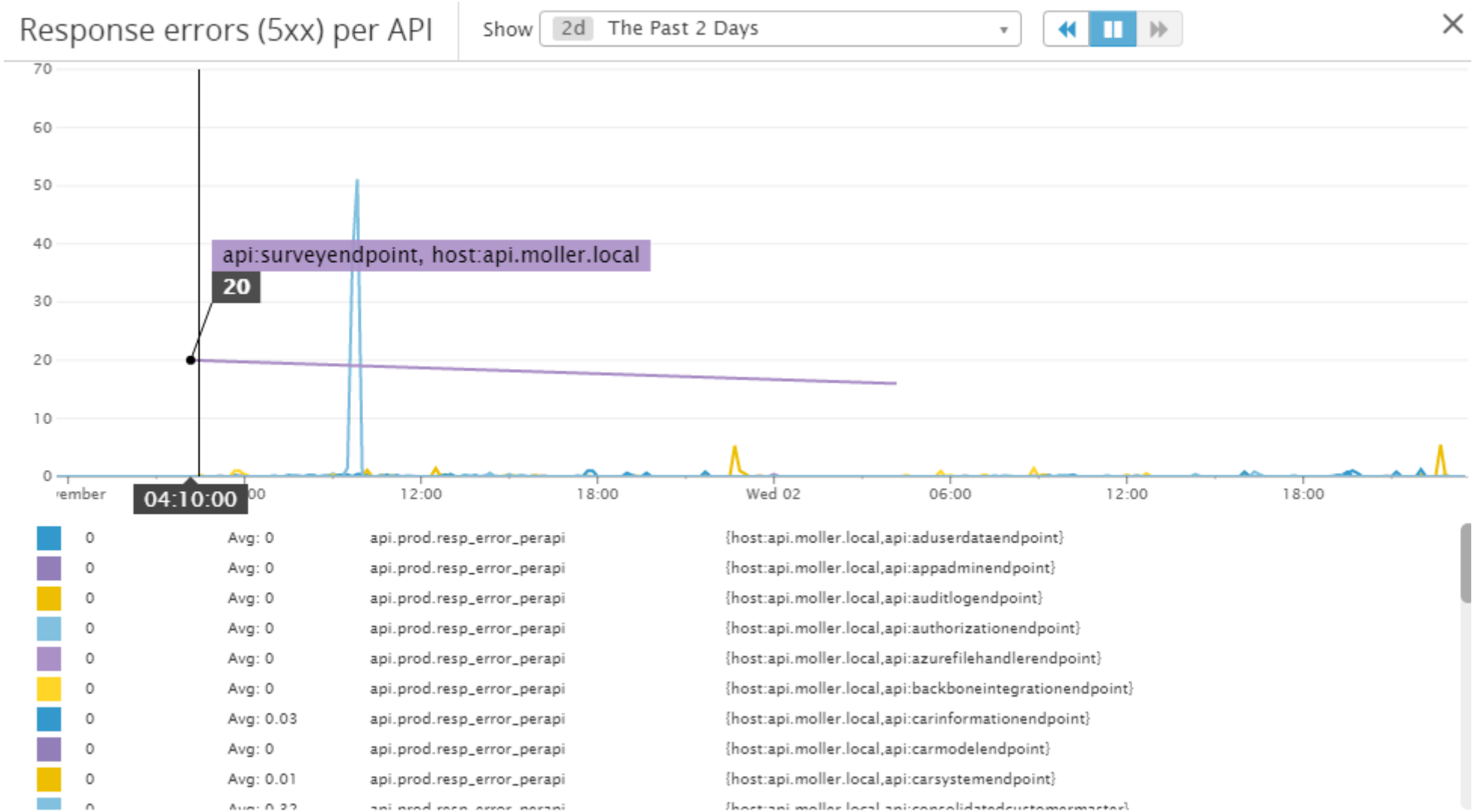
- The reason I use threads is because I submit a lot of data points each minute and DataDogs API takes a little time to receive them. We have to complete submitting everything every minute. If not data sent would be sent at the incorrect time, and load could be constantly growing.
- Currently at 177 threads. Will grow more as new APIs gets added. I need to separate pulling data from VCS and submitting them, and divide into less threads. And also see if I can have less data points submitted (I have some duplication).

Getting the results

- Either manually create timeboards, or with the API
- JSON data for the 5xx errors per API graph:

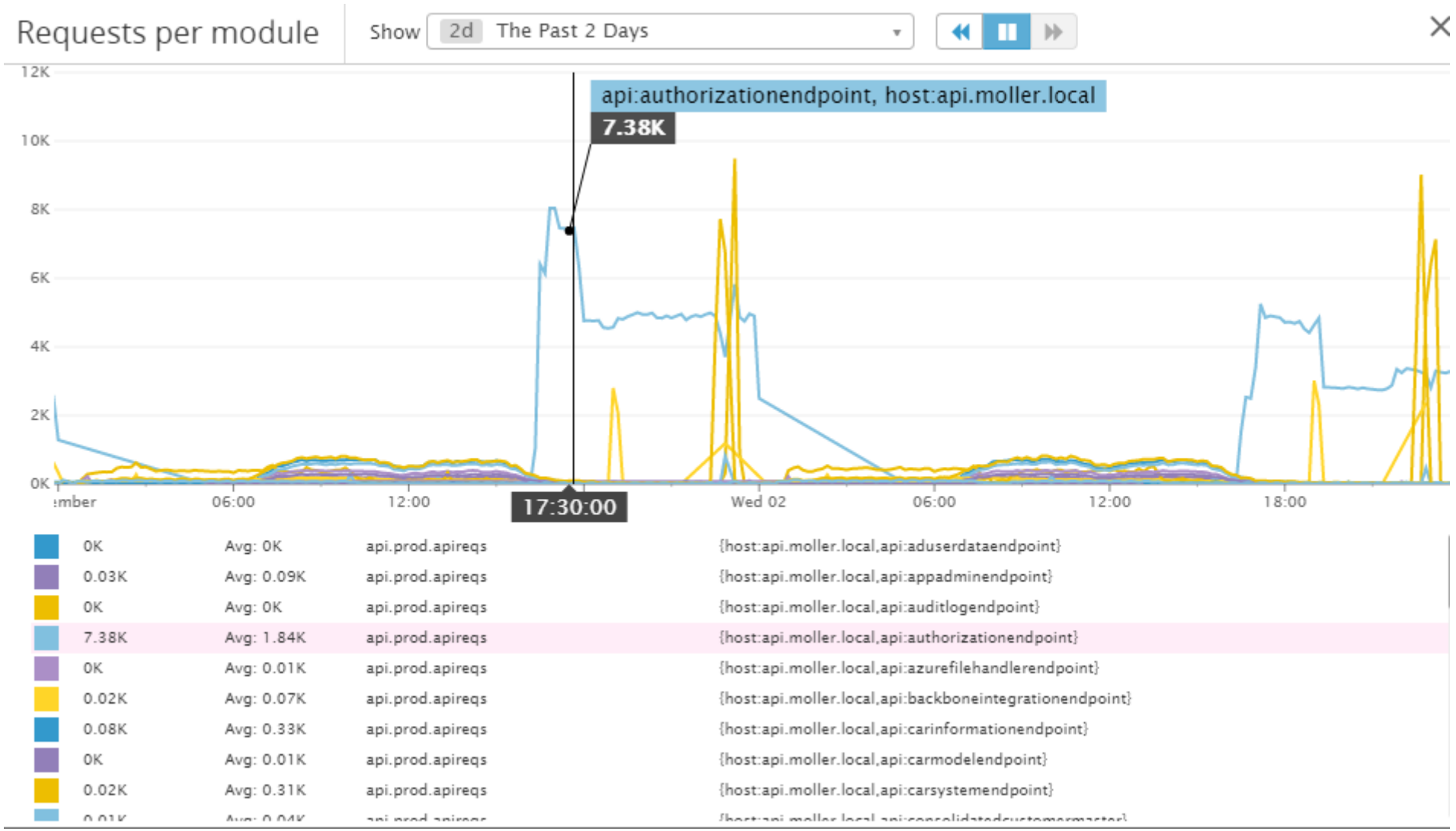

```
{
  "viz": "timeseries",
  "requests": [
    {
      "q": "avg:api.prod.resp_error_perapi{host:api.moller.local} by {api}",
      "conditional_formats": [],
      "type": "line",
      "aggregator": "avg"
    }
  ]
}
```

Level of 5xx response errors per minute per API



Average response time per API

Number of requests per API module

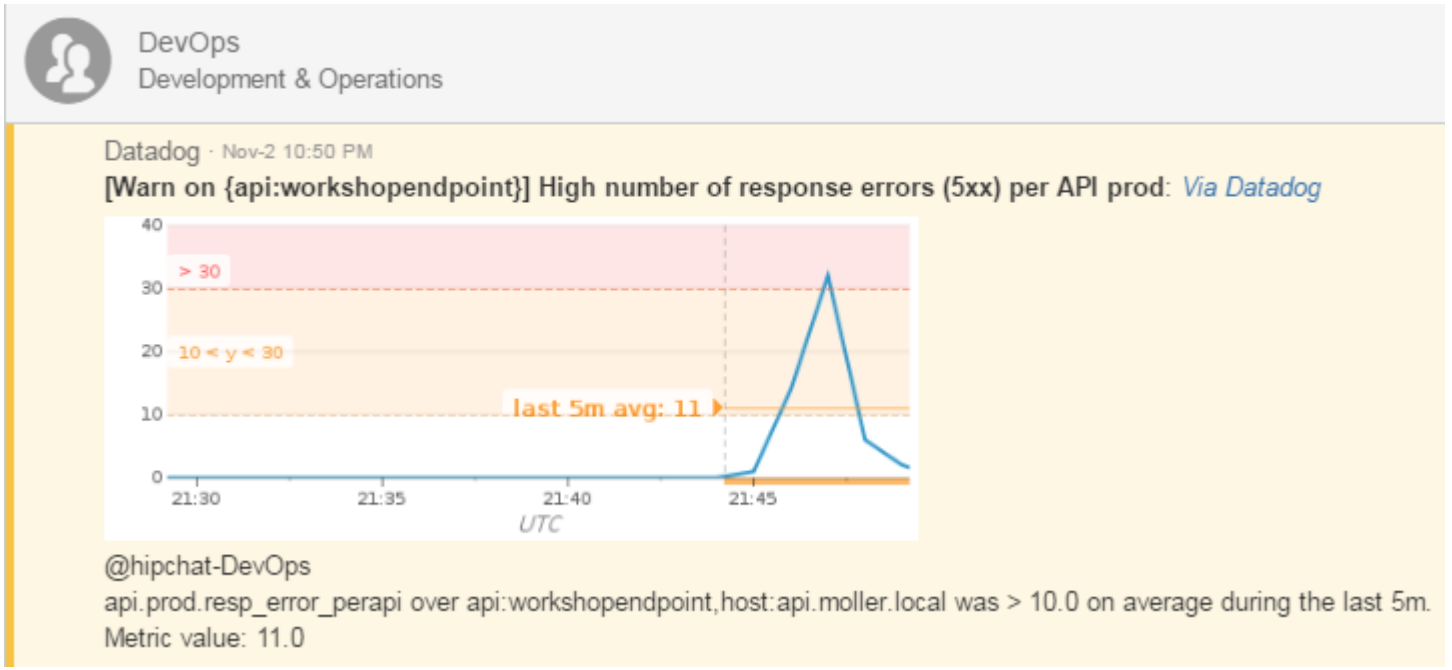


Things I like about DataDogs graphs

- Easy to zoom and scroll in time, to focus on a particular period
- Easy to see exactly when a deviance in traffic patterns starts and ends
- But what more can you do? DataDog has monitors and integrations, and you can automatically create new timeboards (dashboards with time based data) from the API. One example of an integration is HipChat, which monitors can report to.

A DataDog monitor at work

- On our HipChat, we see an alert about an important API endpoint getting more than 10 errors per minute. What is going on?



Did VCS+DataDog solve any DB issues for us?

- No. But it helps find peaks, errors, slowness that may be related.
- In the future we may want to submit DB2 client info usage to DataDog. And require DB clients to identify what application it is.
- Some ongoing work with access keys to identify real consumers of API services. This can also easily be graphed.

Questions? When in need of a car there’s always..

MG MøllerGruppen