FREE eBook

LEARNING Bosun

Free unaffiliated eBook created from
Stack Overflow contributors.

#bosun

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: bosun

It is an unofficial and free Bosun ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Bosun.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with Bosun

Remarks

Bosun is an open-source, MIT licensed, monitoring and alerting system created by Stack Overflow. It has an expressive domain specific language for evaluating alerts and creating detailed notifications. It also lets you test your alerts against historical data for a faster development experience. More details at http://bosun.org/.

Bosun uses a config file to store all the system settings, macros, lookups, notifications, templates, and alert definitions. You specify the config file to use when starting the server, for example `/opt/bosun/bosun -c /opt/bosun/config/prod.conf`. Changes to the file will not be activated until bosun is restarted, and it is highly recommended that you store the file in version control.

Versions

Version	Release Date
0.3.0	2015-06-13
0.4.0	2015-09-18
0.5.0	2016-03-15

Examples

Sample Alert

Bosun alerts are defined in the config file using a custom DSL. They use functions to evaluate time series data and will generate alerts when the warn or crit expressions are non-zero. Alerts use templates to include additional information in the notifications, which are usually an email message and/or HTTP POST request.

```
template sample.alert {
    body = `<p>Alert: {{.Alert.Name}} triggered on {{.Group.host}}
    <hr>
    <p><strong>Computation</strong>
    <table>
        {{range .Computations}}
            <tr><td><a href="{{$.Expr .Text}}">{{.Text}}</a></td><td>{{.Value}}</td></tr>
        {{end}}
    </table>
    <hr>
    {{ .Graph .Alert.Vars.metric }}`

    subject = {{.Last.Status}}: {{.Alert.Name}} cpu idle at {{.Alert.Vars.q | .E}}% on
{{.Group.host}}
```

```
}

notification sample.notification {
    email = alerts@example.com
}

alert sample.alert {
    template = sample.template
    $q = avg(q("sum:rate:linux.cpu{host=*,type=idle}", "1m"))
    crit = $q < 40
    notification = sample.notification
}
```

The alert would send an email with the subject `Critical: sample.alert cpu idle at 25% on hostname` for any host who's Idle CPU usage has averaged less than 40% over the last 1 minute. This example is a "host scoped" alert, but Bosun also supports cluster, datacenter, or globally scoped alerts (see the fundamentals video series for more details).

Sample Configuration File

Here is an example of a Bosun config file used in a development environment:

```
tsdbHost = localhost:4242
httpListen = :8070
smtpHost = localhost:25
emailFrom = bosun@example.org
timeAndDate = 202,75,179,136
ledisDir = ../ledis_data
checkFrequency = 5m

notification example.notification {
        email = alerts@example.org
        print = true
}
```

In this case the config file indicates Bosun should connect to a local OpenTSDB instance on port 4242, listen for requests on port 8070 (on all IP addresses bound to the host), use the localhost SMTP system for email, display additional time zones, use built in Ledis instead of external Redis for system state, and default alerts to a 5 minute interval.

The config also defines an example.notification that can be assigned to alerts, which would usually be included at the end of the config file (see sample alert example).

Docker Quick Start

The quick start guide includes information about using Docker to stand up a Bosun instance.

```
$ docker run -d -p 4242:4242 -p 80:8070 stackexchange/bosun
```

This will create a new instance of Bosun which you can access by opening a browser to http://docker-server-ip. The docker image includes HBase/OpenTSDB for storing time series data, the Bosun server, and Scollector for gathering metrics from inside the bosun container. You can

then point additional scollector instances at the Bosun server and use Grafana to create dashboards of OpenTSDB or Bosun metrics.

The Stackexchange/Bosun image is designed only for testing. There are no alerts defined in the config file and the data will be deleted when the docker image is removed, but it is very helpful for getting a feel for how bosun works. For details on creating a production instance of Bosun see http://bosun.org/resources

Read Getting started with Bosun online: https://riptutorial.com/bosun/topic/565/getting-started-with-bosun

Chapter 2: Alerts: Advanced Scoping

Examples

Understanding the transpose function: t()

Overview

The transpose function is one of Bosun's more powerful functions, but it also takes effort to understand. It is powerful because it lets us alert at different levels than the tag structure of the underlying data.

Transpose changes the *scope* of your alert. This lets you *scope* things into larger collections. So for example if you have queries that return a scope of `host,cluster` and want to alert based on cluster health and not individual hosts, transpose can be used to do this.

What is *scope*?
Scope is the list of tag keys that make up your final result. For example:

- If the scope is `host`, you get per host results in your alerts.
- If your scope is empty (no tag keys) then you could only possibly get one alert.
- If your scope is `host,iface` you could get alerts for every interface on every host in the result.

So the alerts we get are tied to the tags for the data. The transpose function allows us to alert at different scopes other than the metric tag structure. So we can query things that result in `host,cluster` but alert at a `cluster` scope.

Breaking down the function

The signature of the transpose function is:

```
t(numberSet, group string) seriesSet
```

So it takes a *numberSet*, a *scope* a.k.a. group for the result, and returns a *seriesSet*

What are those things?

Set, numberSets, and seriesSets

The results of many functions in bosun are sets, usually a *numberSet* or *seriesSet*. The entire set in the result shares the same tag *keys*. And each item in the set is unique to the value of each corresponding key. If the value of each item in the set is a series (timestamp:value,timestamp:value) then we have a *seriesSet*. If the value of each item is just a

number, then we have a *numberSet*.

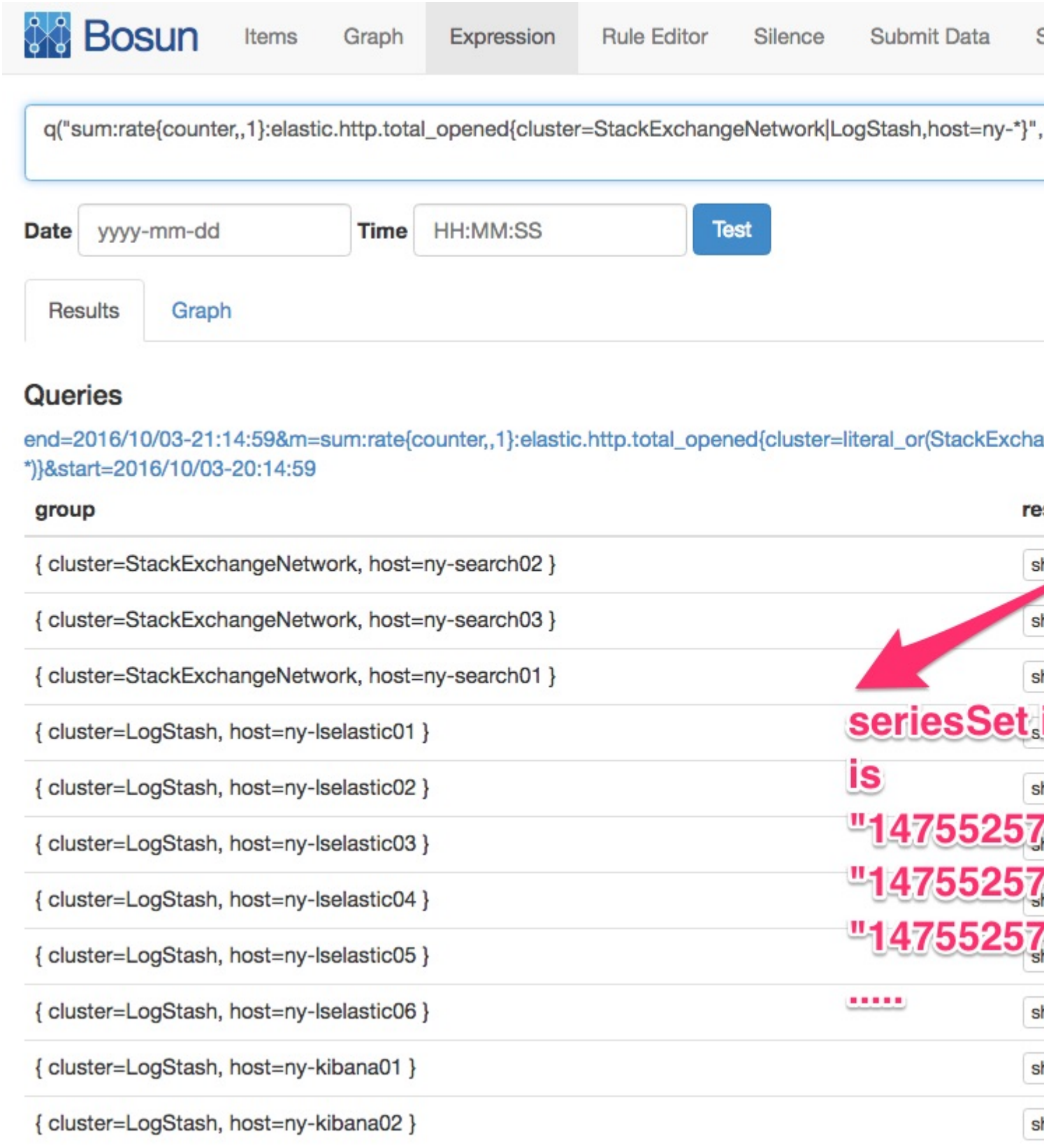
The meat of it

Transpose takes a *numberSet* and returns a *seriesSet* with a larger scope (less tag keys). The resulting *seriesSet* is a bit strange because the index is not time as is the usual case of a *seriesSet*, so timevalue is no longer time and is just an index number. It should therefore be ignored.

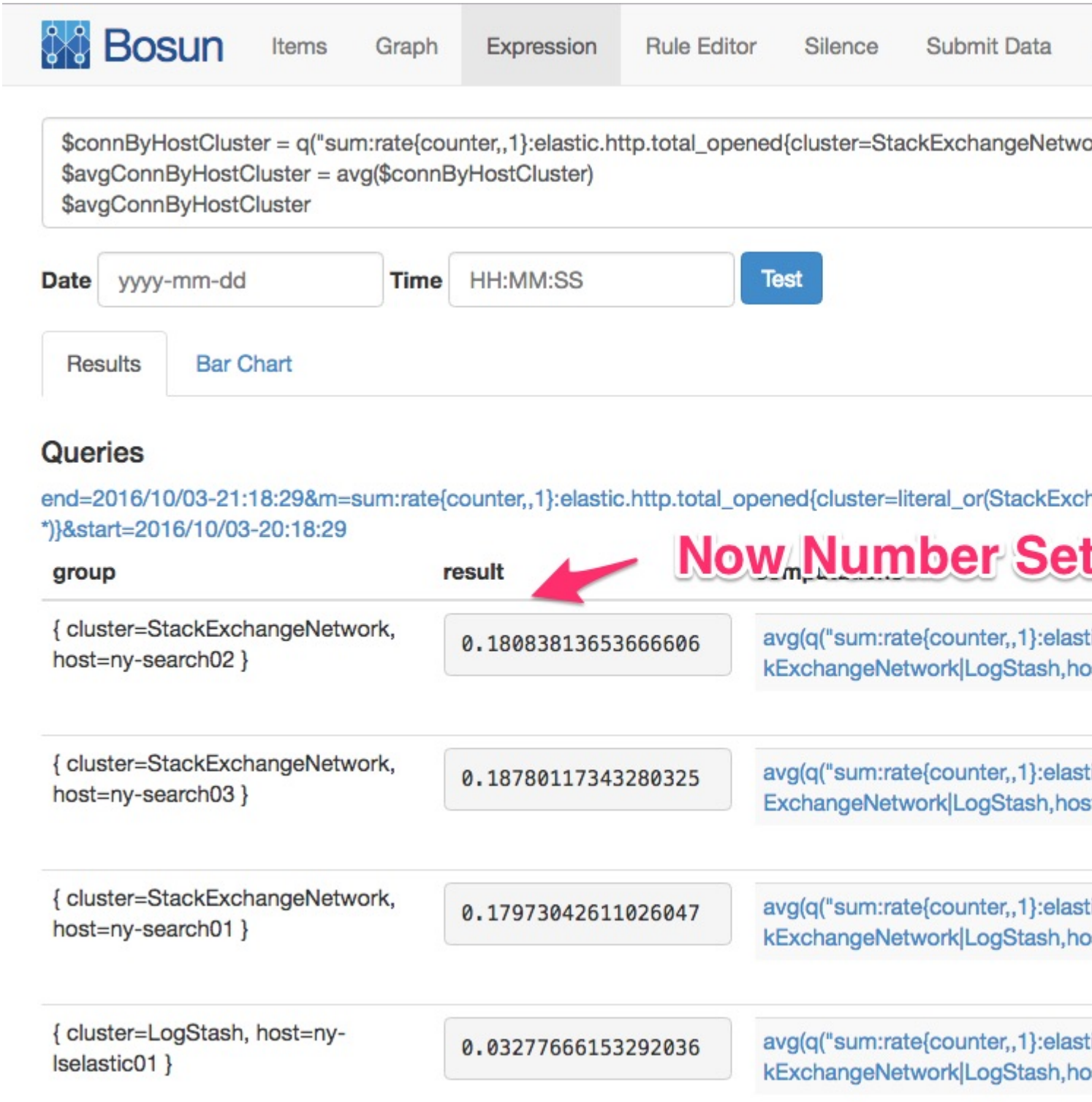
So we end up *transposing* set items into values of the resulting set, where the resulting set type (a seriesSet) can hold multiple values:

Lets step through an example:

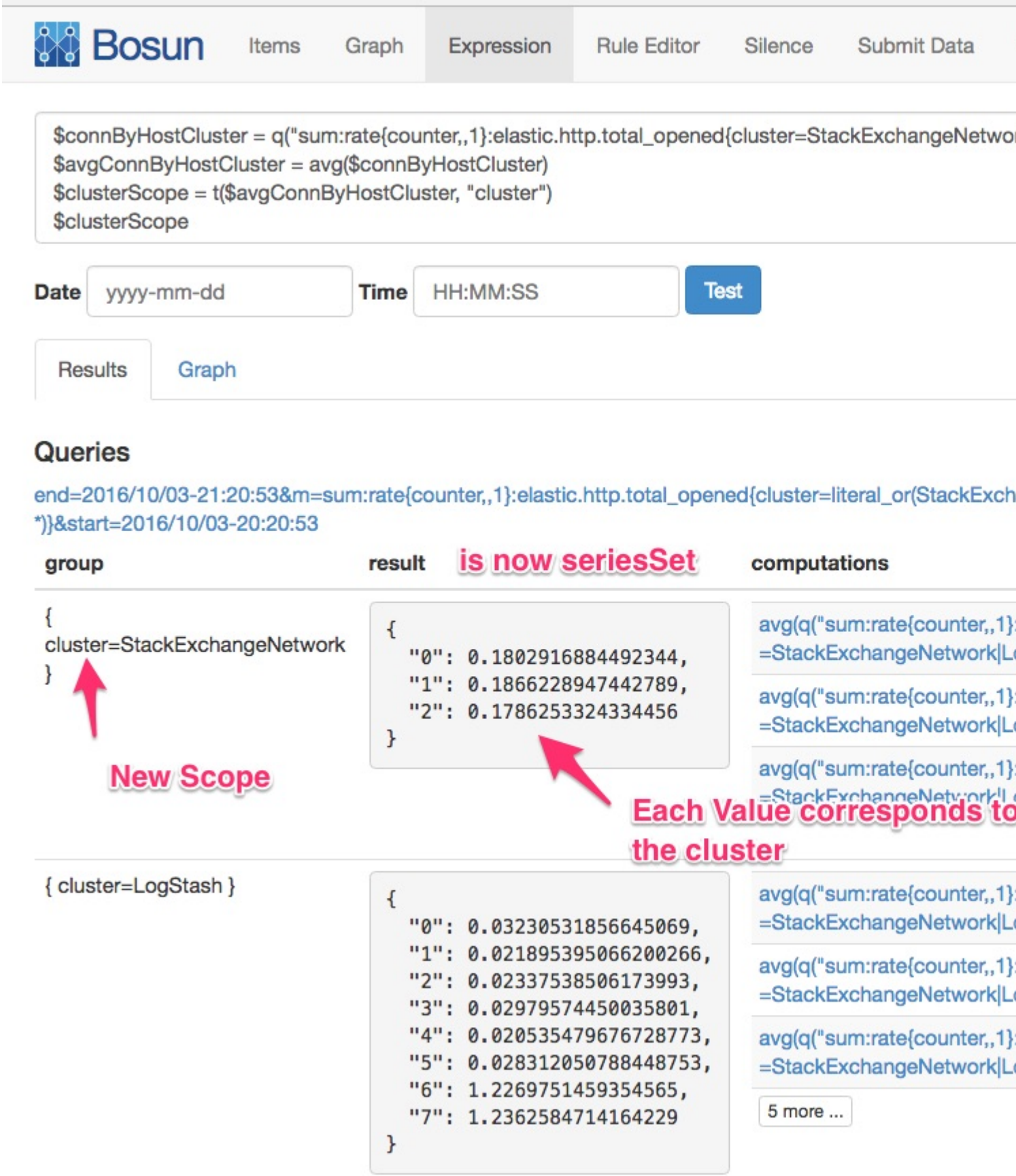
```
# This returns a seriesSet of a scope of host,cluster
$connByHostCluster =
q("sum:rate{counter,,1}:elastic.http.total_opened{cluster=StackExchangeNetwork|LogStash,host=ny-
*}", "1h", "")
```




```
# Turn each item in the set into a numberSet by reducing it via average
$avgConnByHostCluster = avg($connByHostCluster)
```




```
# Transpose to new scope
$clusterScope = t($avgConnByHostCluster, "cluster")
```



You can now do neat things with each item that represents the cluster. For example you could do `sum($clusterScope > 5)` (Note that `$clusterScope` is a *seriesSet*) to get the count of items in the cluster where each item has a rate above five. You could then alert if the count is greater than a certain value. For example, you could also use `len($clusterScope)` to get the number of hosts in

each cluster, and alert on the count of hosts above the threshold relative to the number of hosts in the cluster.

Read Alerts: Advanced Scoping online: https://riptutorial.com/bosun/topic/7213/alerts--advanced-scoping

Chapter 3: Complete Examples

Examples

SSL Certs Expiring

This data is collected by the http_unit and scollector. It warns when an alert is going to expire within a certain amount of days, and then goes critical if the cert has passed the expiration date. This follows the recommended default of warn and crit usage in Bosun (warn: something is going to fail, crit: something has failed).

Template Def

```
template ssl.cert.expiring {
    subject = {{.Last.Status}}: SSL Cert Expiring in {{.Eval .Alert.Vars.daysLeft | printf
"%.2f"}} Days for {{.Group.url_host}}
    body = `
    {{ template "header" . }}
    <table>
       <tr>
            <td>Url</td>
            <td>{{.Group.url_host}}</td>
       </tr>
       <tr>
            <td>IP Address Used for Test</td>
            <td>{{.Group.ip}}</td>
       </tr>
       <tr>
            <td>Days Remaining</td>
            <td>{{.Eval .Alert.Vars.daysLeft | printf "%.2f"}}</td>
       </tr>
       <tr>
            <td>Expiration Date</td>
            <td>{{.Last.Time.Add (parseDuration (.Eval .Alert.Vars.hoursLeft | printf "%vh"))
}}</td>
       </tr>
    </table>
    `
}
```

Alert Definition

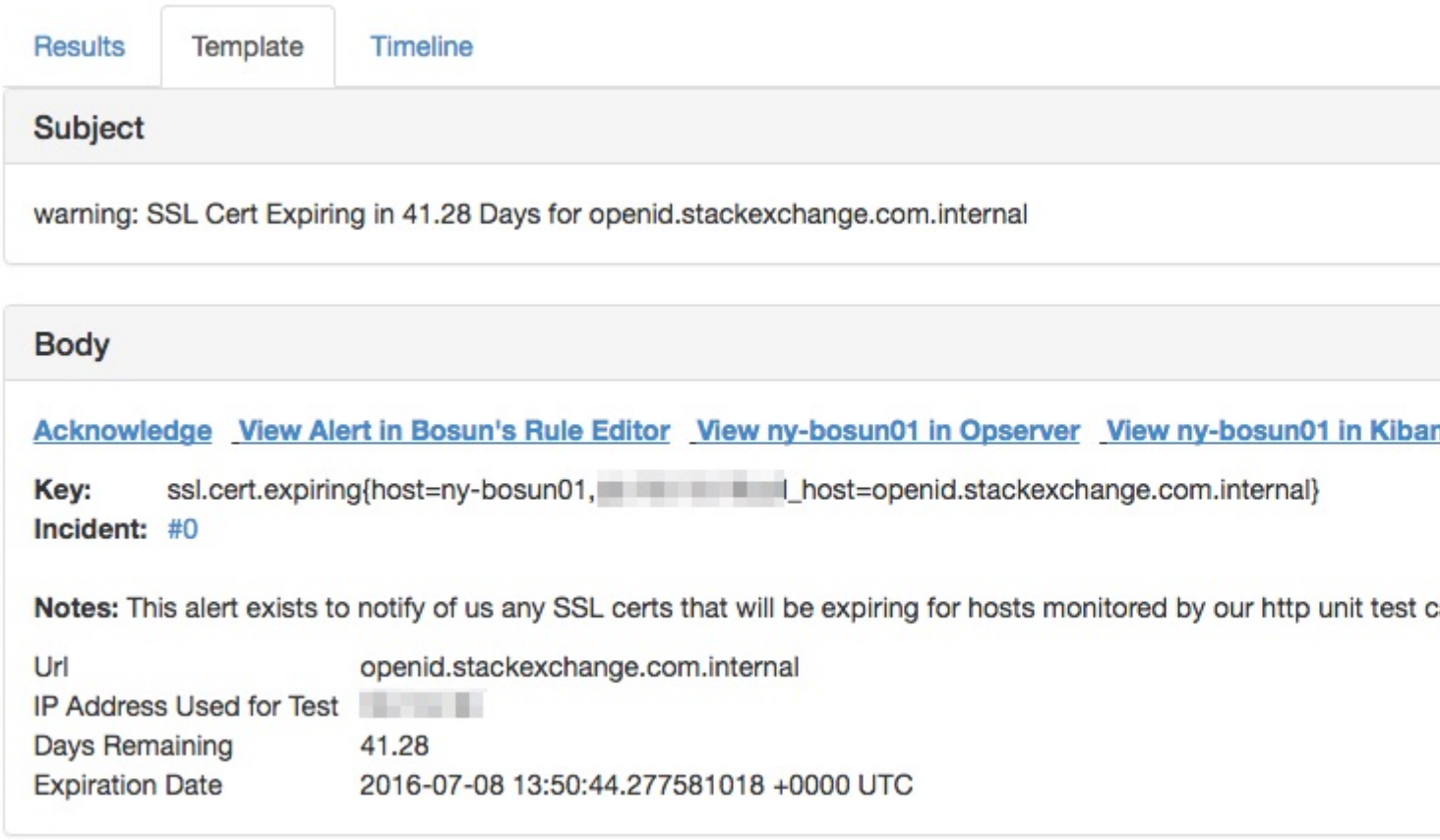
```
alert ssl.cert.expiring {
    template = ssl.cert.expiring
    ignoreUnknown = true
    $notes = This alert exists to notify of us any SSL certs that will be expiring for hosts
monitored by our http unit test cases defined in the scollector configuration file.
    $expireEpoch = last(q("min:hu.cert.expires{host=ny-bosun01,url_host=*,ip=*}", "1h", ""))
    $hoursLeft = ($expireEpoch - epoch()) / d("1h")
```

```
    $daysLeft = $hoursLeft / 24
    warn = $daysLeft <= 50
    crit = $daysLeft <= 0
    warnNotification = default
    critNotification = default
}
```

Alert Explanation

- `q(..)` (func doc) querties OpenTSDB, one of Bosun's supported backends. In returns a type called a seriesSet (which is set of time series, each identified by tag).
- `last()` (func doc) takes the last value of each series in the seriesSet and returns a numberSet.
- The metric, `hu.cert.expires`. is returning the Unix time stamp of when the cert will expire
- `epoch()` (func doc) returns the current unix timestamp. So subtracting current unix timestamp from the expiration epoch gives is the remaining time.
- `d()` (func doc) returns the number of seconds represented by the duration string, the duration string uses the same units as OpenTSDB.

Notification Preview



Example Section of `scollector.toml` referencing the

config for httpunit test cases:

```
[[HTTPUnit]]
  TOML = "/opt/httpunit/data/httpunit.toml"
```

Header Template

In Bosun templates can reference other templates. For emails notifications, you might have a header template to show things you want in all alerts.

Header Template

```
template header {
    body = `
    <style>
    td, th {
        padding-right: 10px;
    }
    </style>
    <p style="font-weight: bold; text-decoration: underline;">
        <a style="padding-right: 10px;" href="{{.Ack}}">Acknowledge</a>
        <a style="padding-right: 10px;" href="{{.Rule}}">View Alert in Bosun's Rule Editor</a>
        {{if .Group.host}}
            <a style="padding-right: 10px;"
href="https://status.stackexchange.com/dashboard/node?node={{.Group.host}}">View
{{.Group.host}} in Opserver</a>
            <a
href="http://kibana.ds.stackexchange.com/app/kibana?#/discover?_g=(refreshInterval:(display:Off,pause:
15m,mode:quick,to:now))&_a=(columns:!(_source),index:%5Blogstash-
%5DYYYY.MM.DD,interval:auto,query:(query_string:(analyze_wildcard:!t,query:'logsource:{{.Group.host}}'
{{.Group.host}} in Kibana</a>
        {{end}}
    </p>
    <table>
        <tr>
            <td><strong>Key: </strong></td>
            <td>{{printf "%s%s" .Alert.Name  .Group }}</td>
        </tr>
        <tr>
            <td><strong>Incident: </strong></td>
            <td><a href="{{.Incident}}">#{{.Last.IncidentId}}</a></td>
        </tr>
    </table>
    <br/>
    {{if .Alert.Vars.notes}}
        <p><strong>Notes:</strong> {{html .Alert.Vars.notes}}</p>
    {{end}}
    {{if .Alert.Vars.additionalNotes}}
        <p>
        {{if not .Alert.Vars.notes}}
            <strong>Notes:</strong>
        {{end}}
        {{ html .Alert.Vars.additionalNotes }}</p>
    {{end}}
```

```
    `
}
```

Explanations:

- `<style>...`: Although style blocks are not supported in email, bosun processes style blocks and then inlines them into the html. So this is shared css for any templates that include this template.
- The `.Ack` link takes you to a Bosun view where you can acknowledge the alert. The `.Rule` link takes you to Bosun's rule editor setting the template, rule, and time of the alert so you can modify the alert, or run it at different times.
- `{{if .Group.host}}...`: `.Group` is the tagset of the alert. So when the warn or crit expression has tags like host=*, we know the alert is in reference to a specific host in our environment. So we then show some links to host specific things.
- The Alert name and key are included to ensure that at least the most basic information is in any alert
- `.Alert.Vars.notes` this is included so if in any alert someone defines the `$notes` variables it will be show in the alert. The encourages people to write notes explaining the purpose of the alert and how to interpret it.
- `.Alert.Vars.additionalNotes` is there in case we want to define a macro with notes, and then have instances of that macro with more notes added to the macro notes.

Linux Bonding Health

Template Definition

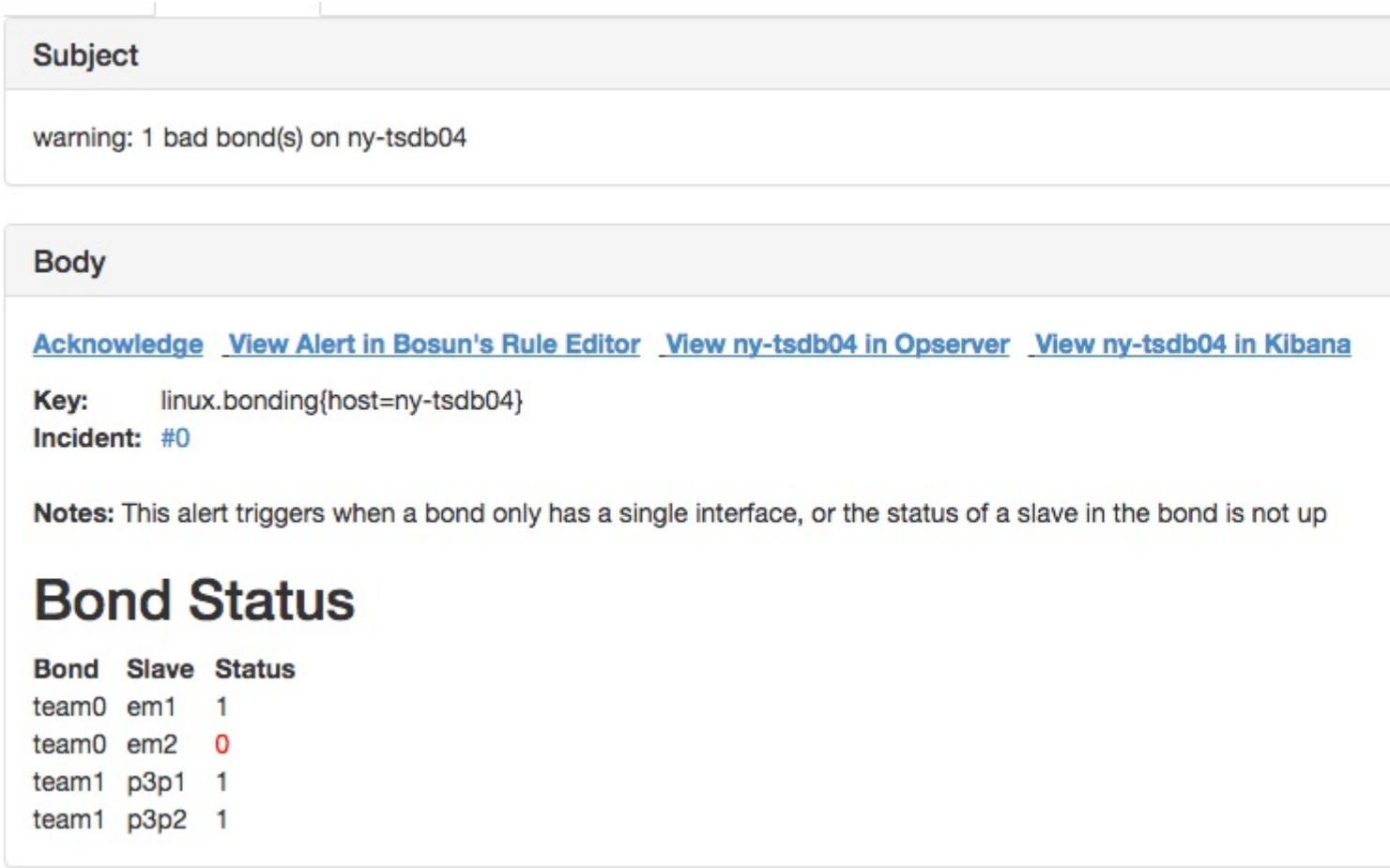
```
template linux.bonding {
    subject = {{.Last.Status}}: {{.Eval .Alert.Vars.by_host}} bad bond(s) on {{.Group.host}}
    body = `{{template "header" .}}
    <h2>Bond Status</h2>
    <table>
    <tr><th>Bond</th><th>Slave</th><th>Status</th></tr>
    {{range $r := .EvalAll .Alert.Vars.slave_status}}
        {{if eq $.Group.host .Group.host}}
            <tr>
                <td>{{$r.Group.bond}}</td>
                <td>{{$r.Group.slave}}</td>
                <td {{if lt $r.Value 1.0}} style="color: red;" {{end}}>{{$r.Value}}</td>
            </tr>
        {{end}}
    {{end}}
    </table>
    `
}
```

Alert Definition

```
alert linux.bonding {
```

```
    template = linux.bonding
    macro = host_based
    $notes = This alert triggers when a bond only has a single interface, or the status of a
slave in the bond is not up
    $slave_status = max(q("sum:linux.net.bond.slave.is_up{bond=*,host=*,slave=*}", "5m", ""))
    $slave_status_by_bond = sum(t($slave_status, "host,bond"))
    $slave_count = max(q("sum:linux.net.bond.slave.count{bond=*,host=*}", "5m", ""))
    $no_good = $slave_status_by_bond < $slave_count || $slave_count < 2
    $by_host = max(t($no_good, "host"))
    warn = $by_host
}
```

Notification Priview



Read Complete Examples online: https://riptutorial.com/bosun/topic/714/complete-examples

Chapter 4: Expression Tips and Tricks

Examples

Avoiding Divide by Zero with NumberSet Operations

In order to avoid a divide by zero with a numberSet (what you get after a reduction like avg()) you can short-circuit the logic:

```
$five = min(q("sum:rate{counter,,1}:haproxy.frontend.hrsp{}{status_code=5xx}", "1h", ""))
$two = avg(q("sum:rate{counter,,1}:haproxy.frontend.hrsp{}{status_code=2xx}", "1h", ""))

$five && $two / $five
```

If the above were just `$two / $five` then when $five is zero, the result will be `+Inf` which will cause an error when used as warn or crit value in an alert expression.

Avoiding Divide by Zero in SeriesSet Operations

With series operations, things are dropped from the left side if there is no corresponding timestamp/datapoint in the right side. You can mix this with the dropbool function to avoid divide by zero:

```
$five = q("sum:rate{counter,,1}:haproxy.frontend.hrsp{}{status_code=5xx}", "1h", "")
$two = q("sum:rate{counter,,1}:haproxy.frontend.hrsp{}{status_code=2xx}", "1h", "")

$two / dropbool($five, ($five > 0))
```

It is possible after dropbool there will be an empty set which would also error. So series operations are recommended for visualization and for alerting it is recommended to use reduction functions earlier in the expression. Alternatively you could wrap the operation in the nv func after reduction:
`nv(avg($two / dropbool($five, ($five > 0))), 0)`

Read Expression Tips and Tricks online: https://riptutorial.com/bosun/topic/5487/expression-tips-and-tricks

Chapter 5: lscount

Parameters

Parameter	Details
indexRoot	The root name of the index to hit, the format is expected to be `fmt.Sprintf("%s-%s", index_root, d.Format("2006.01.02"))`
keyString	Creates groups (like tagsets) and can also filter those groups. It is the format of `"field:regex,field:regex..."`. The `:regex` can be ommited.
filterString	An Elastic regexp query that can be applied to any field. It is in the same format as the keystring argument.
bucketDuration	The same format is an opentsdb duration, and is the size of buckets returned (i.e. counts for every 10 minutes)
startDuration	set the time window from now - see the OpenTSDB `q()` function for more details.
endDuration	set the time window from now - see the OpenTSDB `q()` function for more details.

Remarks

Deprecation

The LogStash query functions are deprecated, and only for use with v1.x of ElasticSearch. If you are running v2 or above of ElasticSearch, then you should refer to the Elastic Query functions.

Caveats

- There is currently no escaping in the keystring, so if you regex needs to have a comma or double quote you are out of luck.
- The regexs in keystring are applied twice. First as a regexp filter to elastic, and then as a go regexp to the keys of the result. This is because the value could be an array and you will get groups that should be filtered. This means regex language is the intersection of the golang regex spec and the elastic regex spec. Elastic uses lucene style regex. This means regexes are always anchored (see the documentation).
- If the type of the field value in Elastic (aka the mapping) is a number then the regexes won't act as a regex. The only thing you can do is an exact match on the number, ie "eventlogid:1234". It is recommended that anything that is a identifier should be stored as a

string since they are not numbers even if they are made up entirely of numerals.
- Alerts using this information likely want to set ignoreUnknown, since only “groups” that appear in the time frame are in the results

Examples

Counting total number of documents in last 5 minutes

`lscount` returns a time bucketed count of matching documents in the LogStash index, according to the specified filter.

A trivial use of this would be to check how many documents in total have been received in the 5 minutes, and alert if it is below a certain threshold.

A Bosun alert for this might look like:

```
alert logstash.docs {
    $notes = This alerts if there hasn't been any logstash documents in the past 5 minutes
    template = logstash.docs
    $count_by_minute = lscount("logstash", "", "", "5m", "5m", "")
    $count_graph = lscount("logstash", "", "", "1m", "60m", "")
    $q = avg($count_by_minute)
    crit = $q < 1
    critNotification = default
}

template logstash.docs {
    body = `{{template "header" .}}
    {{.Graph .Alert.Vars.count_graph }}
    {{template "def" .}}
    {{template "computation" .}}`
    subject = {{.Last.Status}}: Logstash docs per second: {{.Eval .Alert.Vars.q | printf
"%.2f"}} in the past 5 minutes
}
```

This has two instances of lscount:

- $count_by_minute = lscount("logstash", "", "", "5m", "5m", "")
    - This counts the number of documents from the last 5 minutes, in a single 5 minute bucket. You will get one data point in the returned seriesSet with the total number of documents from the last 5 minutes, in the latest `logstash` index
- $count_graph = lscount("logstash", "", "", "1m", "60m", "")
    - This counts the number of documents from the last hour, in 1 minute buckets. There will be a total of 60 data points in the seriesSet returned, which in this instance is used in a graph.

Read lscount online: https://riptutorial.com/bosun/topic/568/lscount

Chapter 6: lsstat

Parameters

Parameter	Details
indexRoot	The root name of the index to hit, the format is expected to be `fmt.Sprintf("%s-%s", index_root, d.Format("2006.01.02"))`
keyString	Creates groups (like tagsets) and can also filter those groups. It is the format of `"field:regex,field:regex..."`. The `:regex` can be ommited.
filterString	An Elastic regexp query that can be applied to any field. It is in the same format as the keystring argument.
field	The field in ElasticSearch to perform the operation on. Must be a numeric field.
rStat	Can be one of `avg`, `min`, `max`, `sum`, `sum_of_squares`, `variance`, `std_deviation`
bucketDuration	The same format is an opentsdb duration, and is the size of buckets returned (i.e. counts for every 10 minutes)
startDuration	set the time window from now - see the OpenTSDB `q()` function for more details.
endDuration	set the time window from now - see the OpenTSDB `q()` function for more details.

Remarks

Deprecation

The LogStash query functions are deprecated, and only for use with v1.x of ElasticSearch. If you are running v2 or above of ElasticSearch, then you should refer to the Elastic Query functions.

Caveats

- There is currently no escaping in the keystring, so if you regex needs to have a comma or double quote you are out of luck.
- The regexs in keystring are applied twice. First as a regexp filter to elastic, and then as a go regexp to the keys of the result. This is because the value could be an array and you will get groups that should be filtered. This means regex language is the intersection of the golang

regex spec and the elastic regex spec. Elastic uses lucene style regex. This means regexes are always anchored (see the documentation).
- If the type of the field value in Elastic (aka the mapping) is a number then the regexes won't act as a regex. The only thing you can do is an exact match on the number, ie "eventlogid:1234". It is recommended that anything that is a identifier should be stored as a string since they are not numbers even if they are made up entirely of numerals.
- Alerts using this information likely want to set ignoreUnknown, since only "groups" that appear in the time frame are in the results

Examples

The average value of a field over time

`lsstat` returns various summary stats per bucket for the specified field. The field *must* be **numeric** in elastic.

`rStat` can be one of `avg`, `min`, `max`, `sum`, `sum_of_squares`, `variance`, `std_deviation`.

The rest of the fields behave the same as `lscount`, except that there is no division based on bucketDuration (since these are summary stats)

```
$max_querytime_by_minute = lsstat("logstash", "", "env:prod", "querytime", "max", "1m", "1h",
"")
```

The `lsstat` in this queries the `logstash` indexes, filters on a field `env` with the value `prod`, and gives the `max` value of `querytime` for the last hour, in one minute buckets.

Read lsstat online: https://riptutorial.com/bosun/topic/569/lsstat

Chapter 7: Notifications: Chat Systems

Remarks

In Bosun notifications are used for both new alert incidents and when an alert is acked/closed/etc. If you don't want the other events to trigger a notification add runOnActions = false to the notification definition.

See Notification Overview for more examples.

Examples

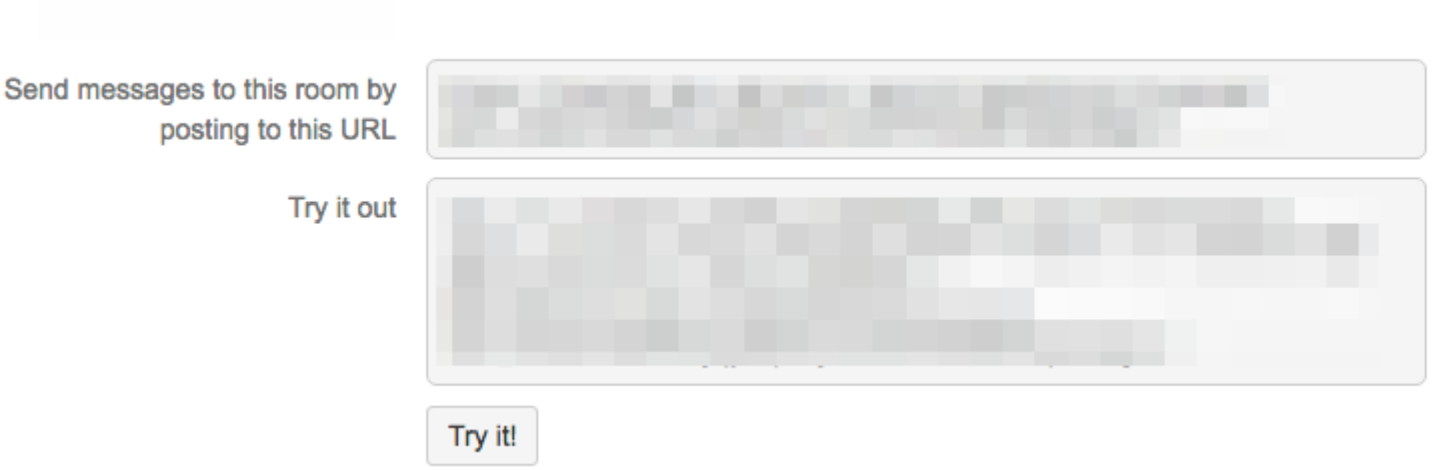
Slack Notifications

```
#Post to a slack.com chatroom via their Incoming Webhooks integration
notification slack{
    post = https://hooks.slack.com/services/abcdefg/abcdefg/abcdefghijklmnopqrstuvwxyz
    body = {"text": {{.|json}}}
}
#To customize the icon and user use:
#   body = {"text": {{.|json}}, "icon_emoji": ":hammer_and_wrench:", "username": "Bosun"}
```

HipChat

Bosun notifications are assigned to alert definitions using warnNotification and critNotification and indicate where to send the rendered alert template when a new incident occur. The ${env.VARIABLENAME} syntax can be used to load values from an Environmental Variable.

In order to post alerts to HipChat, start by creating an Integration named "Bosun". The Integration will provide the URL necessary to post messages (including the token) as seen here:



All that's left is to setup the template and notification:

```
#Example template
```

```
template hipchat.bandwidth {
    subject = `{"color":{{if lt (.Eval .Alert.Vars.dlspeed) (.Eval .Alert.Vars.dlcritval)
}}"red"{{else}} {{if lt (.Eval .Alert.Vars.dlspeed) (.Eval .Alert.Vars.dlwarnval)
}}"yellow"{{else}}"green"{{end}}{{end}},"message":"Server: {{.Group.host}}<br/>Metric:
{{.Alert.Name}}<br/><br/>DL speed: {{.Eval .Alert.Vars.dlspeed | printf "%.2f" }}<br/>DL
Warning threshold: {{.Alert.Vars.dlwarnval}}<br/>DL Critical threshold:
{{.Alert.Vars.dlcritval}}<br/><br/>Notes: {{.Alert.Vars.notes}}<br/><br/>RunBook: <a
href={{.Alert.Vars.runbook}} >wiki article</a>","notify":false,"message_format":"html"}`
}

#Example notification
notification hipchat {
    #Create an Integration in HipChat to generate the POST URL
    #Example URL:  https://<YOURHIPCHATSERVER_FQDN>/v2/room/<ROOM_NUMBER>/<TOKEN>
    post = ${env.HIPCHAT_ROOM_ABC}
    body = {{.}}
    contentType = application/json
}
```

Read Notifications: Chat Systems online: https://riptutorial.com/bosun/topic/7153/notifications---chat-systems

Chapter 8: Notifications: Overview

Syntax

- notification *name* {
  - email = dev-alerts@example.com, prod-alerts@example.com, ...
  - post = http://example.com
  - get = http://example.com
  - next = another-notification-definition
  - timeout = 30m
  - runOnActions = false
  - body = {"text": {{.|json}}}
  - contentType = application/json
  - print = true
- }

Remarks

In Bosun notifications are used for both new alert incidents and when an alert is acked/closed/etc. If you don't want the other events to trigger a notification add `runOnActions = false` to the notification definition.

See also:

- Notifications: Chat Systems

Examples

SMS Notifications with plivo

There are two ids you will need from your plivo account. Replace `authid` and `authtoken` in this snippet with those values. The `src` value should also be a valid number assigned to your account. `dst` can be any number you want, or multiple seperated by `<`.

```
notification sms {
  post = https://authid:authtoken@api.plivo.com/v1/Account/authid/Message/
  body = {"text": {{.|json}}, "dst":"15551234567","src":"15559876543"}
  contentType = application/json
  runOnActions = false
}
```

This will text the alert subject to all numbers in `dst`.

Email Notifications

To send email notifications you need to add the following settings to your config file:

```
#Using a company SMTP server (note only one can be define)
smtpHost = mail.example.com:25
emailFrom = bosun@example.com

#Using Gmail with TLS and username/password
smtpHost = smtp.gmail.com:587
emailFrom = myemail@gmail.com
smtpUsername = myemail@gmail.com
smtpPassword = ${env.EMAILPASSWORD}

#Chained notifications will escalate if an incident is not acked before the timeout
notification it {
    email = it-alerts@example.com
    next = oncall
    timeout = 30m
}

#Could set additional escalations here using any notification type (email/get/post)
#or set next = oncall to send another email after the timeout if alert is still not acked
notification oncall {
    email = escalated-alerts@example.com
}

#Multiple email addresses can be specified in one notification definition
#Use runOnActions = false to exclude updates on actions (ack/closed/forget)
notification engineering {
    email = core-alerts@example.com,qa-alerts@example.com,prod-alerts@example.com
    runOnActions = false
}
```

Overview

Bosun notifications are assigned to alert definitions using warnNotification and critNotification and indicate where to send the rendered alert template when a new incident occur. Notifications can be sent via email or use HTTP GET/POST requests. There also is a Print notification that just adds information to the Bosun log file.

If you want to hide a URL, Password, or API Key from being in plain text you can use `${env.VARIABLENAME}` to load the value from an Environmental Variable (usually exported from the Bosun init script). Please note that there are no protections on who can access the variables (they can easily be displayed in a template) but it does prevent them from being displayed directly on the Rule Editor page or in the .conf file.

```
notification logfile {
    print = true
}

#print can be added to any notification type to help with debugging
notification email {
    email = sysadmins@example.com
    print = true
}
```

HTTP GET/POST Notifications

Alert incidents can be sent to other system using HTTP GET or HTTP POST requests. You can either send the rendered alert directly (using markdown in the template perhaps) or use `body = ... {{.|json}} ...` and contentType to send the alert data over as part of a JSON object. Another approach is to only send the basic incident information and then have the receiving system pull additional details from the bosun API.

```
notification postjson {
    post = ${env.POSTURL}
    body = {"text": {{.|json}}, apiKey=${env.APIKEY}}
    contentType = application/json
}
```

The contentType for HTTP GET/POST requests is **application/x-www-form-urlencoded** by default.

SMS Notifications with Twilio

Swap out `AccountSid`, `AuthToken`, `ToPhoneNumber` and `FromPhoneNumber` for your credentials/intended recipients. You need to ensure that if the `ToPhoneNumber` and `FromPhoneNumber` have + in them, they are urlencoded (ie: as %2B)

```
notification sms {
    post = https://{AccountSid}:{Authtoken}@api.twilio.com/2010-04-
01/Accounts/{AccountSid}/Messages.json
    body = Body={{.}}&To={ToPhoneNumber}&From={FromPhoneNumber}
}
```

From gist: https://gist.github.com/aodj/58535c4c152b6073eaf5

PagerDuty Notifications

```
#Post to pagerduty.com
notification pagerduty {
    post = https://events.pagerduty.com/generic/2010-04-15/create_event.json
    contentType = application/json
    runOnActions = false
    body = `{
      "service_key": "myservicekey",
      "incident_key": {{.|json}},
      "event_type": "trigger",
      "description": {{.|json}},
      "client": "Bosun",
      "client_url": "http://bosun.example.com/"
    }`
}
```

Changing Notification Using Lookup

In some cases you may want to change which notification you use based on a tag in the Alert keys. You can do this using the Lookup feature. Note: Lookup only works if you are using OpenTSDB and sending data to the Bosun to be indexed. For other backends or non-indexed data

you have to use lookupSeries instead.

```
notification default {
    email = team@example.com
}

notification JSmith{
    email = JSmith@example.com
}

#This will use the JSmith lookup for any alerts where the host tag starts with ny-jsmith
lookup host_base_contact {
    entry host=ny-jsmith* {
        main_contact = JSmith
    }
    entry host=* {
        main_contact = default
    }
}

alert blah {
    ...
    warn = q(...)
    warnNotification = lookup("host_base_contact", "main_contact")
    critNotification = lookup("host_base_contact", "main_contact")
}
```

This can also be applied to multiple alerts using Macros:

```
macro host.based.contacts {
    warnNotification = lookup("host_base_contact", "main_contact")
    critNotification = lookup("host_base_contact", "main_contact")
}
```

Read Notifications: Overview online: https://riptutorial.com/bosun/topic/612/notifications--overview

Chapter 9: Packages and Initialization Scripts

Remarks

There currently aren't any installation packages provided for Bosun or Scollector, only binaries on the Bosun release page. It is up to the end user to find the best way to deploy the files and run them as a service.

Examples

Scollector init.d script

Example init script for scollector:

```
#!/bin/bash
#
# scollector        Startup script for scollector.
#
# chkconfig: 2345 90 60
# description: scollector is a replacement for OpenTSDB's TCollector \
# and can be used to send metrics to a Bosun server

# Source function library.
. /etc/init.d/functions

RETVAL=0
PIDFILE=/var/run/scollector.pid

prog=scollector
exec=/opt/scollector/scollector-linux-amd64
scollector_conf=/opt/scollector/scollector.toml
scollector_logs=/var/log/scollector
scollector_opts="-conf $scollector_conf -log_dir=$scollector_logs"

lockfile=/var/lock/subsys/$prog

# Source config
if [ -f /etc/sysconfig/$prog ] ; then
    . /etc/sysconfig/$prog
fi

start() {
        [ -x $exec ] || exit 5
        umask 077
        echo -n $"Starting scollector: "
        daemon --check=$exec --pidfile="$PIDFILE" "{ $exec $scollector_opts & } ; echo \$! >|
$PIDFILE"
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && touch $lockfile
        return $RETVAL
}
stop() {
        echo -n $"Shutting down scollector: "
```

```
        killproc -p "$PIDFILE" $exec
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && rm -f $lockfile
        return $RETVAL
}
rhstatus() {
        status -p "$PIDFILE" -l $prog $exec
}
restart() {
        stop
        start
}

case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart)
        restart
        ;;
  status)
        rhstatus
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|status}"
        exit 3
esac

exit $?
```

Bosun init.d script

Here is an init.d script for Bosun that includes setting Environmental Variables that can be used to hide secrets from the raw config. It uses http://software.clapper.org/daemonize/ to run the program as a daemon.

```
#!/bin/sh
#
# /etc/rc.d/init.d/bosun
# bosun
#
# chkconfig: - 98 02
# description: bosun

### BEGIN INIT INFO
# Provides:          bosun
# Required-Start:    networking
# Required-Stop:     networking
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Runs teh bosun
# Description:       bosun

### END INIT INFO
```

```
# Source function library.
. /etc/rc.d/init.d/functions

base_dir="/opt/bosun"
exec="/opt/bosun/bosun"
prog="bosun"
config="${base_dir}/config/prod.conf"

[ -e /etc/sysconfig/$prog ] && . /etc/sysconfig/$prog

lockfile=/var/lock/subsys/$prog
pidfile=/var/run/bosun.pid
logfile=/var/log/$prog.log

#These "secrets" can be used in the prod.conf using syntax like ${env.CHAT} or ${env.API_KEY}
export CHAT=https://chat.company.com/rooms/123?key=12345678901234567890123456789O
export API_KEY=12345678901234567890123456789O

check() {
    $exec -t -c $config
    if [ $? -ne 0 ]; then
       echo "Errors found in configuration file, check it with '$exec -t'."
       exit 1
    fi
}

start() {
    [ -x $exec ] || exit 5
    [ -f $config ] || exit 6
    check
    echo -n $"Starting $prog: "
    # if not running, start it up here, usually something like "daemon $exec"
    ulimit -n 65536
    daemon daemonize -a -c $base_dir -e $logfile -o $logfile  -p $pidfile -l $lockfile $exec -
c $config $OPTS
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
    return $retval
}

stop() {
    echo -n $"Stopping $prog: "
    # stop it here, often "killproc $prog"
    killproc -p $pidfile -d 5m
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
    return $retval
}

restart() {
    check
    stop
    start
}

reload() {
    restart
}
```

```
force_reload() {
    restart
}

rh_status() {
    # run checks to determine if the service is running or use generic status
    status $prog
}

rh_status_q() {
    rh_status >/dev/null 2>&1
}


case "$1" in
    start)
        rh_status_q && exit 0
        $1
        ;;
    stop)
        rh_status_q || exit 0
        $1
        ;;
    restart)
        $1
        ;;
    reload)
        rh_status_q || exit 7
        $1
        ;;
    force-reload)
        force_reload
        ;;
    status)
        rh_status
        ;;
    condrestart|try-restart)
        rh_status_q || exit 0
        restart
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|condrestart|try-restart|reload|force-
reload}"
        exit 2
esac
```

Bosun systemd unit file

```
#Create Bosun unit file at /etc/systemd/system/bosun.service
[Unit]
Description=Bosun Service
After=network.target
After=rsyslog.service

[Service]
Type=simple
User=root
ExecStart=/opt/bosun/bosun -c /opt/bosun/config/prod.conf
Restart=on-abort
```

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

#enable and start service
#systemctl enable bosun
#systemctl start bosun
#If you edit this file, be sure to run `systemctl daemon reload` so Systemd recognizes the
changes made
```

Scollector systemd unit file

```
#Create Scollector unit file at /etc/systemd/system/scollector.service
[Unit]
Description=Scollector Service
After=network.target

[Service]
Type=simple
User=root
ExecStart=/opt/scollector/scollector -h mybosunserver.example.com
Restart=on-abort

[Install]
WantedBy=multi-user.target

#enable and start service
#systemctl enable scollector
#systemctl start scollector
#If you edit this file, be sure to run `systemctl daemon reload` so Systemd recognizes the
changes made
```

TSDBRelay systemd unit file

TSDBRelay can be used to forward metrics to an OpenTSDB instance, send to Bosun for indexing, and relay to another opentsdb compatible instance for backup/DR/HA. It also has options to denormalize metrics with high tag cardinality or create redis/ledis backed external counters.

```
#Create tsdbrelay unit file at /etc/systemd/system/tsdbrelay.service
[Unit]
Description=tsdbrelay Service
After=network.target

[Service]
Type=simple
User=root
ExecStart=/opt/tsdbrelay/tsdbrelay -b localhost:8070 -t localhost:4242 -l 0.0.0.0:5252  -r
localhost:4243 #Local tsdb/bosun and influxdb opentsdb endpoint at 4243
#For external counters add: -redis redishostname:6379 -db 0
#For denormalized metrics: -
denormalize=os.cpu__host,os.mem.used__host,os.net.bytes__host,os.net.bond.bytes__host,os.net.other.byte

Restart=on-abort

[Install]
```

```
WantedBy=multi-user.target
```

Scollector and Bosun Packages for Chef/Puppet/Vagrant/Ansible

Chef Scollector Cookbook: https://github.com/alexmbird/chef-scollector

Chef Bosun Cookbook: https://github.com/ptqa/chef-bosun

Puppet scollector module: https://github.com/axibase/axibase-puppet-modules

Bosun Ansible/Vagrant example: https://github.com/gnosek/bosun-deploy

Install scollector on CentOS 7

As a privileged user (root or sudo):

Create scollector directory:

```
mkdir /opt/scollector
```

In the /opt/scollector directory, download the latest binary build from the bosun/scollector site, [http://bosun.org/scollector/][1]

```
wget https://github.com/bosun-monitor/bosun/releases/download/"version"/scollector-"OS"-"arch"
```

ex:
```
wget https://github.com/bosun-monitor/bosun/releases/download/0.5.0/scollector-linux-amd64
```

Create a symbolic link in /usr/local/bin:

```
ln -s /opt/scollector/scollector-linux-amd64 /usr/local/bin/scollector
```

Create the configuration directory;

```
mkdir /etc/scollector
```

Using this guide create your scollector configuration file, scollector.toml

The path for the configuration file is then /etc/scollector/scollector.conf

ex:

```
 Host = "http://xxx.xxx.xxx.xxx:8070" #replace xxx with the IP of your Bosun server
    Hostname = "DevOps-Bosun-Prod"
    [[ICMP]]
      Host = "some.hostname.here"
    [[ICMP]]
      Host = "some.other.hostname.here"
    [tags]
      hostgroup = "system"
    #[[GoogleAnalytics]]
```

```
    #  ClientID = ""
    #  Secret = ""
    #  Token = ""
```

Create the Service file, /etc/systemd/system/scollector.service

ex:

```
[Unit]
Description=Scollector Service
After=network.target

[Service]
Type=simple
User=root
ExecStart=/usr/local/bin/scollector -conf=/etc/scollector/scollector.toml
Restart=on-abort


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

Tell Systemd that you have created a new service:

```
systemctl enable scollector.service
```

Start scollector:

```
systemctl start scollector
```

You can see if scollector has started by running:

```
systemctl status scollector
```

Alternatively, you can view the system message log, you're looking for something like:

```
Jul 29 23:19:27 bosun-prod systemd: Started Scollector Service.
Jul 29 23:19:27 bosun-prod systemd: Starting Scollector Service...
Jul 29 23:19:27 bosun-prod scollector[4363]: info: main.go:213: OpenTSDB host:
http://127.0.0.1:8070
```

Read Packages and Initialization Scripts online: https://riptutorial.com/bosun/topic/775/packages-and-initialization-scripts

Chapter 10: Scollector: External Collectors

Remarks

Scollector supports tcollector style external collectors that can be used to send metrics to Bosun via custom scripts or executables. External collectors are a great way to get started collecting data, but when possible it is recommended for applications to send data directly to Bosun or to update scollector so that it natively supports additional systems.

The ColDir configuration key specifies the external collector directory, which is usually set to something like **/opt/scollector/collectors/** in Linux or **C:\Program Files\scollector\collectors** in Windows. It should contain numbered directories just like the ones used in OpenTSDB tcollector. Each directory represents how often scollector will try to invoke the collectors in that folder (example: 60 = every 60 seconds). Use a directory named 0 for any executables or scripts that will run continuously and create output on their own schedule. Any non-numeric named directories will be ignored, and a lib and etc directory are often used for library and config data shared by all collectors.

External collectors can use either the simple data output format from tcollector or they can send JSON data if they want to include metadata.

Examples

Sample collector written in PowerShell

```
#Example of a PowerShell external collector. See http://bosun.org/scollector/external-
collectors for details
#This file should be saved in C:\Program Files\scollector\collectors\0\mymetrics.ps1 since it
is a continuous output script
#scollector.toml should have ColDir = 'C:\Program Files\scollector\collectors'

#Setup format strings and other variables
$epoch = New-Object DateTime (1970,1,1)
$MetricMetadata='{{"metric":"{0}","name":"{1}","value":"{2}"}}'
$MetricData='{{"metric":"{0}","timestamp":{1:F0},"value":{2:G}{3}}}'
$MetricTags=',"tags":{{{0}}}'
$Base="mymetric"

#Send metadata for each metric once on startup (Scollector will resend to Bosun periodically)
Write-Output ($MetricMetadata -f "$Base.test","rate","gauge") #See
https://godoc.org/bosun.org/metadata#RateType
Write-Output ($MetricMetadata -f "$Base.test","unit","item")  #See
https://godoc.org/bosun.org/metadata#Unit
Write-Output ($MetricMetadata -f "$Base.test","desc","A test metric")

#Create tags and send metrics
$tags=$MetricTags -f '"mykey":"myvalue"' #generate static tags here. Can append additional
tags in the loop if needed.
#Use $tags="" to exclude all tags but those added by Scollector.
Write-Output ($MetricData -f
```

```
"$Base.test",[datetime]::UtcNow.Subtract($epoch).TotalSeconds,42.123,$tags)
do {
    $delay = Get-Random -Minimum 5 -Maximum 25
    sleep -Seconds $delay
    Write-Output ($MetricData -f
"$Base.test",[datetime]::UtcNow.Subtract($epoch).TotalSeconds,$delay,$tags)
} while ($true)

#If a continuous output script ever exits scollector will restart it. If you just want
periodic data every 60 seconds you
#can use a /60/ folder instead of /0/ and allow the script to exit when finished sending a
batch of metrics.
```

Twitter Collector written in Go

The following can be saved as main.go. After you update the EDITME settings and build the executable it can be used as a continuous external collector.

```
package main

import (
    "fmt"
    "log"
    "net/url"
    "strconv"
    "time"

    "github.com/ChimeraCoder/anaconda"
)

func main() {
    anaconda.SetConsumerKey("EDITME")
    anaconda.SetConsumerSecret("EDITME")
    api := anaconda.NewTwitterApi("EDITME", "EDITME")
    v := url.Values{}
    sr, err := api.GetSearch("stackoverflow", nil)
    if err != nil {
        log.Println(err)
    }
    var since_id int64 = 0
    for _, tweet := range sr {
        if tweet.Id > since_id {
            since_id = tweet.Id
        }
    }
    count := 0
    for {
        now := time.Now().Unix()
        v.Set("result_type", "recent")
        v.Set("since_id", strconv.FormatInt(since_id, 10))
        sr, err := api.GetSearch("stackoverflow", nil)
        if err != nil {
            log.Println(err)
        }
        for _, tweet := range sr {
            if tweet.Id > since_id {
                count += 1
                since_id = tweet.Id
            }
```

```
        }
        fmt.Println("twitter.tweet_count", now, count, "query=stackoverflow")
        time.Sleep(time.Second * 30)
    }
}
```

Hadoop HDFS disk usage written in Bash

This is a continuous collector that uses the `hadoop fs -du -s /hbase/*` command to get details about the HDFS disk usage. This metric is very useful for tracking space in an OpenTSDB system.

```
#!/bin/bash
while true; do
    while read -r bytes raw_bytes path; do
        echo "hdfs.du $(date +"%s") $bytes path=$path"
        #https://community.cloudera.com/t5/Storage-Random-Access-HDFS/hdfs-du-format-
change/td-p/27192 KMB 2015-08-24T12:01:20Z
        echo "hdfs.du.raw $(date +"%s") $raw_bytes path=$path"
    done < <(hadoop fs -du -s  /hbase/*)
    sleep 30
done
```

StackExchange.Exceptional collector written in Go with Metadata

The following Go file can be compiled into a continuous external collector that will query a MSSQL server database that uses the StackExchange.Exceptional schema. It will query multiple servers/databases for all exceptions since UTC 00:00 to convert the raw entries into a counter. It also uses the bosun.org/metadata package to include metadata for the **exceptional.exceptions.count** metric.

```
/*
Exceptional is an scollector external collector for StackExchange.Exceptional.
*/
package main

import (
    "database/sql"
    "encoding/json"
    "fmt"
    "log"
    "strings"
    "time"

    "bosun.org/metadata"
    "bosun.org/opentsdb"

    _ "github.com/denisenkom/go-mssqldb"
)

func mssqlConnect(server, database, user, pass, port string) (*sql.DB, error) {
    dsn := fmt.Sprintf("server=%s;port=%s;database=%s;user id=%s;password=%s", server, port,
database, user, pass)
    return sql.Open("mssql", dsn)
}
```

```
type Exceptions struct {
    GUID            string
    ApplicationName string
    MachineName     string
    CreationDate    time.Time
    Type            string
    IsProtected     int
    Host            string
    Url             string
    HTTPMethod      string
    IPAddress       string
    Source          string
    Message         string
    Detail          string
    StatusCode      int
    SQL             string
    DeletionDate    time.Time
    FullJson        string
    ErrorHash       int
    DuplicateCount  int
}

type ExceptionsCount struct {
    ApplicationName string
    MachineName     string
    Count           int64
    Source          string
}

type ExceptionsDB struct {
    Server     string
    DBName     string
    DBPassword string
    DBPort     string
    Source     string
}

const (
    defaultPassword = "EnterPasswordHere"
    defaultPort     = "1433"

    metric     = "exceptional.exceptions.count"
    descMetric = "The number of exceptions thrown per second by applications and machines.
Data is queried from multiple sources. See status instances for details on exceptions."
)

func main() {
    mds := []metadata.Metasend{
        {
            Metric: metric,
            Name:   "rate",
            Value:  "counter",
        },
        {
            Metric: metric,
            Name:   "unit",
            Value:  metadata.Error,
        },
        {
            Metric: metric,
            Name:   "desc",
```

```
            Value:  descMetric,
        },
    }
    for _, m := range mds {
        b, err := json.Marshal(m)
        if err != nil {
            log.Fatal(err)
        }
        fmt.Println(string(b))
    }
    instances := [...]ExceptionsDB{
        {"NY_AG", "NY.Exceptions", defaultPassword, defaultPort, "NY_Status"},
        {"CO-SQL", "CO.Exceptions", defaultPassword, defaultPort, "CO_Status"},
        {"NY-INTSQL", "Int.Exceptions", defaultPassword, defaultPort, "INT_Status"},
    }
    for _, exdb := range instances {
        go run(exdb)
    }
    select {}
}

func run(exdb ExceptionsDB) {
    const interval = time.Second * 30

    query := func() {
        // Database name is the same as the username
        db, err := mssqlConnect(exdb.Server, exdb.DBName, exdb.DBName, exdb.DBPassword,
exdb.DBPort)
        if err != nil {
            log.Println(err)
        }
        defer db.Close()
        var results []ExceptionsCount
        sqlQuery := `
        SELECT ApplicationName, MachineName, MAX(Count) as Count FROM
        (
            --New since UTC rollover
            SELECT ApplicationName, MachineName, Sum(DuplicateCount) as Count from Exceptions
            WHERE CreationDate > CONVERT (date, GETUTCDATE())
            GROUP BY MachineName, ApplicationName
            UNION --Zero out any app/machine combos that had exceptions in last 24 hours
            SELECT DISTINCT ex.ApplicationName, ex.MachineName, 0 as Count from Exceptions ex
WHERE ex.CreationDate Between Convert(Date, GETUTCDATE()-1) And Convert(Date, GETUTCDATE())
        ) as T
        GROUP By T.MachineName, T.ApplicationName`
        rows, err := db.Query(sqlQuery)
        if err != nil {
            log.Println(err)
            return
        }
        defer rows.Close()
        for rows.Next() {
            var r ExceptionsCount
            if err := rows.Scan(&r.ApplicationName, &r.MachineName, &r.Count); err != nil {
                log.Println(err)
                continue
            }
            r.Source = exdb.Source
            results = append(results, r)
        }
        if err := rows.Err(); err != nil {
```

```
            log.Println(err)
        }
        if len(results) > 0 {
            now := time.Now().Unix()
            for _, r := range results {
                application, err := opentsdb.Clean(r.ApplicationName)
                if err != nil {
                    log.Println(err)
                    continue
                }
                db := opentsdb.DataPoint{
                    Metric:    metric,
                    Timestamp: now,
                    Value:     r.Count,
                    Tags: opentsdb.TagSet{
                        "application": application,
                        "machine":     strings.ToLower(r.MachineName),
                        "source":      r.Source,
                    },
                }
                b, err := db.MarshalJSON()
                if err != nil {
                    log.Println(err)
                    continue
                }
                fmt.Println(string(b))
            }
        }
    }

    for {
        wait := time.After(interval)
        query()
        <-wait
    }
}
```

Powershell external collector script function

```
<#
    .DESCRIPTION
        Writes the metric out in bosun external collector format which is compatible with
scollector external scripts
    .PARAMETER metric
        Name of the metric (eg : my.metric)
    .PARAMETER type
        Type of metric (counter, gauge, etc)
    .PARAMETER unit
        Type of unit (connections, operations, etc)
    .PARAMETER desc
        Description of the metric
    .PARAMETER value
        The current value for the metric
#>
function Write-Metric
{
param(
    [string]$metric,
    [string]$type,
```

```
    [string]$unit,
    [string]$desc,
    $value
)

$epoch = New-Object DateTime (1970,1,1)


$obj = @{
    metric = $metric
    name = "rate"
    value = $type
}

Write-Host (ConvertTo-Json $obj -Compress)

$obj.name="unit"
$obj.value=$unit

Write-Host (ConvertTo-Json $obj -Compress)

$obj.name="desc"
$obj.value=$desc

Write-Host (ConvertTo-Json $obj -Compress)

$output = @{
    metric = $metric
    timestamp= [int]([datetime]::UtcNow.Subtract($epoch).TotalSeconds)
    value=$value
    tags= @{
        host=$env:computername.ToLower()
    }
}

Write-Host (ConvertTo-Json $output -Compress)

}
```

Read Scollector: External Collectors online: https://riptutorial.com/bosun/topic/720/scollector--external-collectors

Chapter 11: Scollector: Overview

Remarks

Scollector is a monitoring agent that can be used to send metrics to Bosun or any system that accepts OpenTSDB style metrics. It is modelled after OpenTSDB's tcollector data collection framework but is written in Go and compiled into a single binary. One of the design goals is to auto-detect services so that metrics will be sent with minimal or no configuration needed. You also can create external collectors that generate metrics using a script or executable and use Scollector to queue and send the metrics to the server.

You are NOT required to use Scollector when using Bosun, as you can also send metrics directly to the /api/put route, use another monitoring agent, or use a different backend like Graphite, InfluxDB, or ElasticSearch.

Examples

Setup with sample scollector.toml file

Scollector binaries for Windows, Mac, and Linux are available from the Bosun release page and can be saved to **/opt/scollector/** or **C:\Program Files\scollector**. The Scollector configuration file uses TOML v0.2.0 to specify various settings and defaults to being named scollector.toml in the same folder as the binary. The configuration file is optional and only required if you need to override a default value or include settings to activate a specific collector.

```
#Where to send metrics. If omitted the default is bosun:80.
#Config file setting can also be overridden using -h bosunhostname on command line
Host = "mybosunserver.example.com:8080"

#Optional folder where to find external collector scripts/binaries
ColDir  = 'C:\Program Files\scollector\collectors'

#Number of data points to include in each batch. Default is 500, should be set higher if you
are sending a lot of metrics.
BatchSize = 5000
```

You can then either install Scollector as a service or just run it manually via:

```
#Override default configuration file location
scollector -conf /path/to/myconfig.toml

#List all built-in collectors
scollector -l

#-p will print metrics to the screen instead of sending to Bosun.
#-f "..." will only run specific collectors. Add DisableSelf = true to toml file to exclude
scollector.* self metrics
scollector -p -f "c_cpu_windows,c_network_"
```

Running Scollector as a service

On Windows you can install Scollector as a service using the -winsvc="install" flag. On Mac and Linux you must manually create a service or init script. For example here is a basic systemd unit file:

```
#Scollector unit file saved to /etc/systemd/system/scollector.service
[Unit]
Description=Scollector Service
After=network.target

[Service]
Type=simple
User=root
ExecStart=/opt/scollector/scollector -h mybosunserver.example.com
Restart=on-abort

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

Read Scollector: Overview online: https://riptutorial.com/bosun/topic/719/scollector--overview

Chapter 12: Scollector: Process and Service Monitoring

Remarks

Scollector can be used to monitor processes and services in Windows and Linux. Some processes like IIS application pools are monitored automatically, but usually you need to specify which processes and services you want to monitor.

Examples

Linux process and systemd service monitoring

Scollector will monitor Linux processes specified in the configuration file.

```
[[Process]]
  Command = "/opt/bosun/bosun"
  Name = "bosun"

[[Process]]
  Command = "ruby"
  Name = "puppet-agent"
  Args = "puppet"

[[Process]]
  Command = "/haproxy$"
  Name = "haproxy-t1"
  Args = "/etc/haproxy-t1/haproxy-t1.cfg"

[[Process]]
  Command = '/usr/bin/redis-server \*:16389'
  Name = "redis-bosun-dev"
  IncludeCount = true
```

Scollector can also use the D-Bus API to determine the state of services managed by systemd and specified in the configuration file.

```
[[SystemdService]]
  Name = "^(puppet|redis-.*|keepalived|haproxy-t.*)$"
  WatchProc = false

[[SystemdService]]
  Name = "^(scollector|memcached)$"
  WatchProc = true
```

Windows proccess and service monitoring

Scollector will monitor any Windows processes or services specified in the configuration file.

```
[[Process]]
  Name = "^scollector"

[[Process]]
  Name = "^chrome"

[[Process]]
  Name = "^(MSSQLSERVER|SQLSERVERAGENT)$"
```

Windows .NET process monitoring

Scollector can also monitor any Windows processes using the .NET framework. If no ProcessDotNet settings are specified it will default to just monitoring the w3wp worker processes for IIS. You can specify which applications to monitor in the configuration file.

```
[[ProcessDotNet]]
  Name = "^w3wp"

[[ProcessDotNet]]
  Name = "LINQPad"
```

Matching process will be monitored under the `dotnet.*` metrics, and if there is more than one matching process they will be assigned incrementing id tag values starting at 1. Where possible the w3wp names will be changed to match the iis_pool-names used for process monitoring.

Monitoring Docker Containers

Scollector has built in support for using cAdvisor to generate **container.*** metrics in Bosun for each Docker container on a host. To get started you will need to start a new container on each docker host:

```
docker run --name cadvisor --restart=always -d -p 8080:8080 google/cadvisor
```

And then from an external source poll for metrics using scollector with the Cadvisor configuration option. If you are using Kubernetes to manage containers you may also want to use the TagOverride option to override the `docker_id` tags (shorten to 12 chars), add a `container_name` and `pod_name` tag, and remove the `docker_name` and `name` tag:

```
[[Cadvisor]]
  URL = "http://mydockerhost01:8080"

[[Cadvisor]]
  URL = "http://mydockerhost02:8080"

#Override tags for Kubernetes containers
[[TagOverride]]
  CollectorExpr = "cadvisor"
  [TagOverride.MatchedTags]
    docker_name = 'k8s_(?P<container_name>[^\.]+)\.[0-9a-z]+_(?P<pod_name>[^-]+)'
    docker_id = '^(?P<docker_id>.{12})'
  [TagOverride.Tags]
    docker_name = ''
```

```
    name = ''
```

You may also want to send the metrics to a test instance of Bosun (maybe using the Bosun Docker Container) to verify the metrics look correct before sending them to a production Bosun instance (hard to clean up data after it is sent).

Read Scollector: Process and Service Monitoring online: https://riptutorial.com/bosun/topic/721/scollector--process-and-service-monitoring

Chapter 13: Silencing and Squelching Alerts

Examples

Squelching a host

If one does not want to receive any alert for a specific host or service - at least momentarily - one can squelch it.

```
alert thisis.down {
  macro = host.mymacro
  template = mytemplate
  $notes = This alert will...
  $metric = "avg:os.service.running{host=*,name=...
  warn = min( a($metric, ...

  squelch = host=sqldev01,flavor=amq
  squelch = host=test01
}
```

This alert won't appear in the dashboard for service *amq* on host *sqldev01*, and won't appear at all for any service running on host *test01*.

Read Silencing and Squelching Alerts online: https://riptutorial.com/bosun/topic/6791/silencing-and-squelching-alerts

Chapter 14: Templates: Graph and GraphAll

Remarks

Bosun Templates can include graphs to provide more information when sending a notification. The graphs can use variables from the alert and filter base on the tagset for the alert instance or use the GraphAll function to graph all series. When viewed on the Dashboard or in an email you can click on the graph to load it in the Expression page.

You can also create a Generic Template with optional Graphs that can be shared across multiple alerts.

Examples

Graph using Alert Variable

Using .Graph will filter the results to only include those that match the tagset for the alert. For instance an alert for os.low.memory{host=ny-web01} would only include series with the host=ny-web01 tags. If multiple series match then only the first matching result will be used.

```
template graph.template {
    subject = ...

    body = `{{template "header" .}}

    <strong>Graph</strong>
    <div>{{.Graph .Alert.Vars.graph}}</div>

    <strong>Graph With Y Axis Label Literal</strong>
    <div>{{.Graph .Alert.Vars.graph "Free Memory in GB"}}</div>

    <strong>Graph With Y Axis Label From Variable</strong>
    <div>{{.Graph .Alert.Vars.graph .Alert.Vars.graph_unit}}</div>

    `
}

alert os.low.memory {
    template = graph.template
    ...
    $graph = q("avg:300s-avg:os.mem.percent_free{host=$host}", "1d", "")
    $graph_unit = Percent Free Memory (Including Buffers and Cache)
    ...
}
```

GraphAll using Alert Variable

Using .GraphAll will include all the results in the graph.

```
template graph.template {
```

```
    subject = ...

    body = `{{template "header" .}}

    <strong>GraphAll</strong>
    <div>{{.GraphAll .Alert.Vars.graph}}</div>

    <strong>GraphAll With Y Axis Label Literal</strong>
    <div>{{.GraphAll .Alert.Vars.graph "All Systems Free Memory in GB"}}</div>

    <strong>GraphAll With Y Axis Label From Variable</strong>
    <div>{{.GraphAll .Alert.Vars.graph .Alert.Vars.graph_unit}}</div>

    `
}

alert os.low.memory {
    template = graph.template
    ...
    $graph = q("avg:300s-avg:os.mem.percent_free{host=$host}", "1d", "")
    $graph_unit = All Systems Percent Free Memory (Including Buffers and Cache)
    ...
}
```

Graph or GraphAll using inline or dynamic query

Graph queries can be defined inline if you don't want to use an Alert variable.

```
template graph.template {
    subject = ...

    body = `{{template "header" .}}

    <strong>Graph With Inline Query</strong>
    <div>{{.Graph "q(\"avg:300s-avg:os.mem.percent_free{host=specifichost}\", \"1d\", \"\")"
"Free Memory in GB"}}</div>

    <strong>GraphAll with Inline Query</strong>
    <div>{{.GraphAll "q(\"avg:300s-avg:os.mem.percent_free{host=host1|host2|host3}\", \"1d\",
\"\")" "All Systems Free Memory in GB"}}</div>

    `
}
```

Sometimes you may want to create the query for a graph dynamically in the template itself by combining one or more variables. For instance a host down alert might want to include the Bosun known hosts ping metric using the dst_host tag.

```
template host.down {
    subject = ...

    body = `{{template "header" .}}

    <strong>Graph from one variable</strong>
    <div>{{printf "q(\"sum:bosun.ping.timeout{dst_host=%s}\", \"8h\", \"\")" (.Group.host) |
.Graph}}</div>
```

```
    <strong>Graph from multiple variables</strong>
    <div>{{printf "q(\"sum:%s{host=%s,anothertag=%s}\", \"8h\", \"\")" "some.metric.name"
.Group.host "anothervalue" | .Graph}}</div>
    `
}
```

The printf statement will generate `q("sum:bosun.ping.timeout{dst_host=alerthostname}", "8h", "")` when that host triggers an alert and then use that to create the graph in the notification.

Filter, Sort, Limit and Graph

When using GraphAll you may still want to filter the results, in which case you can use an Alert variable with the Filter, Sort, and Limit functions.

```
template graph.template {
    subject = ...

    body = `{{template "header" .}}

    <strong>Graph Filtered Variable</strong>
    <div>{{.Graph .Alert.Vars.graph_below_5 .Alert.Vars.graph_unit}}</div>

    <strong>Graph Filter+Sort+Limit Variable (Maximum of 10 series)</strong>
    <div>{{.Graph .Alert.Vars.graph_lowest_10 .Alert.Vars.graph_unit2}}</div>

    `
}

alert os.low.memory {
    template = graph.template
    ...
    $graph_all = q("avg:300s-avg:os.mem.percent_free{host=ny-*}", "1d", "")
    $graph_unit = All Systems with Less than 5 Percent Free Memory
    $graph_below_5 = filter($graph_all, min($graph_all) < 5)

    $graph_unit2 = Ten Systems with lowest Percent Free Memory
    $graph_lowest_10 = filter($graph_all, limit(sort(min($graph_min_5),"asc"),10))
    ...
}
```

Using Merge to Combine Series

If you want to graph two series on one graph, you can use the Merge function. This can also be combined with the Series function to manipulate the Y axis (like forcing it to start at zero).

```
template graph.template {
    subject = ...

    body = `{{template "header" .}}

    <strong>Graph With Merge+Series so Y Axis Starts At Zero</strong>
    <div>{{.Graph .Alert.Vars.graph_merged .Alert.Vars.graph_unit}}</div>
    `
}
```

```
alert os.low.memory {
    template = graph.template
    ...
    $graph_time = "1d"
    $graph_host = q("avg:300s-avg:os.mem.percent_free{host=myhost}", $graph_time, "")
    $graph_unit = Notice the Y axis always starts at zero now
    $graph_series = series("value=zero", epoch()-d($graph_time), 0, epoch(),0)
    $graph_merged = merge($graph_host,$graph_series)
    ...
}
```

Read Templates: Graph and GraphAll online: https://riptutorial.com/bosun/topic/716/templates--graph-and-graphall

Chapter 15: Templates: HTTPGet and HTTPGetJSON

Examples

HTTPGetJSON

HTTPGetJSON performs an HTTP request to the specified URL and returns a jsonq.JsonQuery object for use in the alert template. Example:

```
template example {
    {{ $ip := 8.8.8.8 }}
    {{ $whoisURL := printf "http://whois.arin.net/rest/ip/%s" $ip }}
    {{ $whoisJQ := $.HTTPGetJSON $whoisURL }}
    IP {{$ip}} owner from ARIN is {{ $whoisJQ.String "net" "orgRef" "@name" }}
}
```

In this case the $ip address is hard coded but in a real alert it would usually come from the alert tags using something like `{{ $ip := .Group.client_ip}}` where client_ip is a tag key whose value is an IP address.

The jsonq results are similar to the results generated by the jq JSON processor, so you can test in a BASH shell using:

```
$ curl -H "Accept: application/json" http://whois.arin.net/rest/ip/8.8.8.8 | jq  ".net.orgRef"
{
  "@handle": "GOGL",
  "@name": "Google Inc.",
  "$": "https://whois.arin.net/rest/org/GOGL"
}
```

Read Templates: HTTPGet and HTTPGetJSON online: https://riptutorial.com/bosun/topic/578/templates--httpget-and-httpgetjson

Chapter 16: Templates: Overview

Syntax

- #See https://golang.org/pkg/text/template/ for Go Template **Action** and **Function** syntax
- expression = alert status {{.Last.Status}} and a variable {{.Eval .Alert.Vars.q | printf "%.2f"}}
- expression = `Use backticks to span
- multiple lines with line breaks
- in the Bosun config file`
- template *name* {
    - subject = *expression*
    - body = *expression*
- }

Remarks

Bosun templates are based on the Go html/template package and can be shared across multiple alerts, but a single template is used to render all Bosun Notifications for that alert. Alerts reference which template to use via the `template` directive and specify which notifications to use via the `warnNotification` and `critNotification` directives (can have multiple warn/crit notifications defined for each alert).

Templates are rendered when an alert instance is triggered and can:

- Use variables defined in the alert to display text or graphs
- Use Go Template Actions and Functions like if, range, and, not, index, and printf
- Access Bosun metadata to display additional details about a system
- Access other Bosun Template Variables and Functions
- Pull information from other systems via HTTPGet and HTTPGetJSON
- Use Images, HTML, and CSS styles for rich notifications (CSS can be inlined for better email support)

The template subject will be displayed as headers on the dashboard, as the subject line of email notifications, and as the default contents of HTTP POST notifications. The template body will be displayed when an alert instance is expanded and as the body of email notifications.

Examples

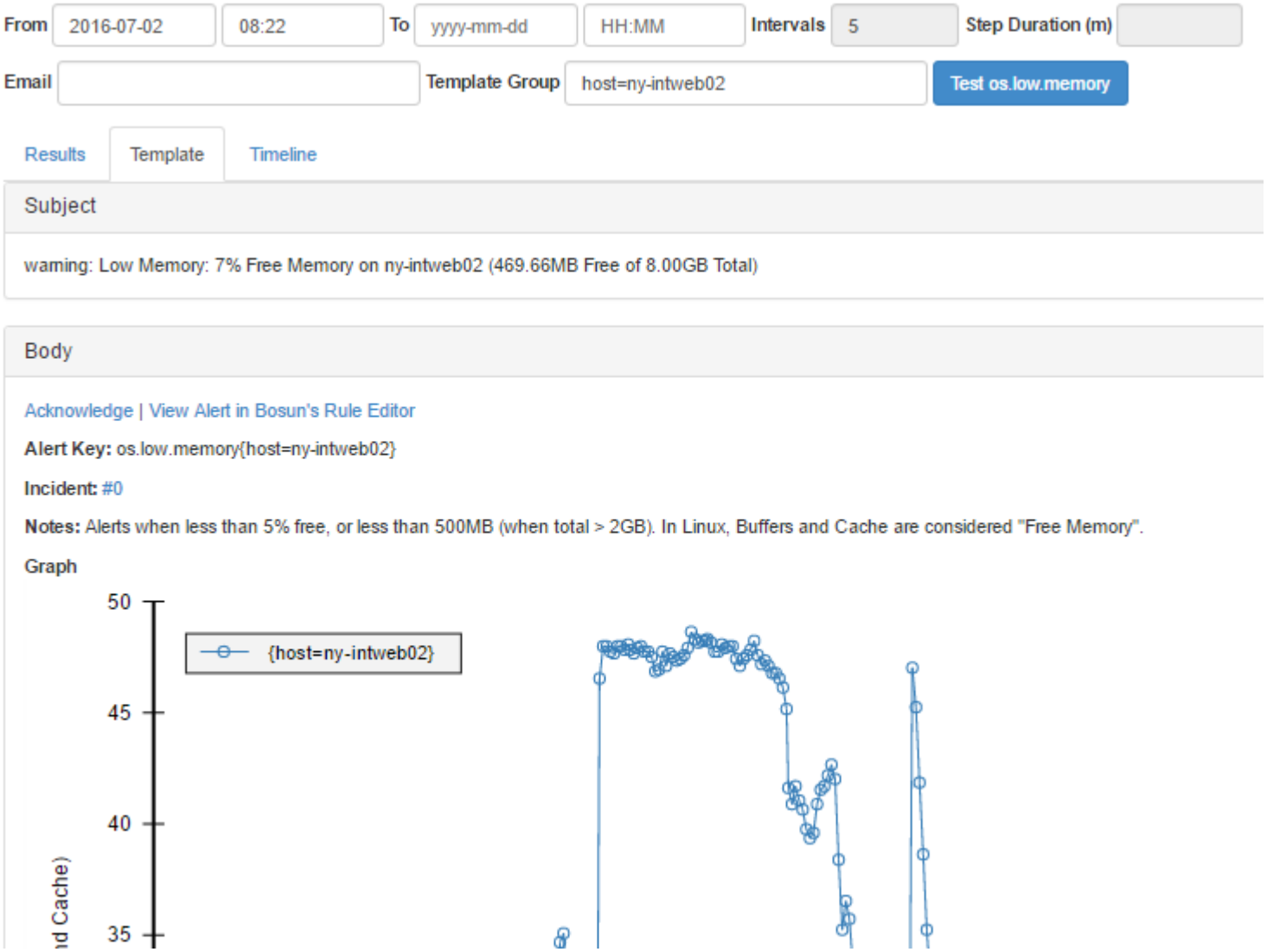
Low Memory Alert and Template

Templates can be previewed and edited using the Rule Editor tab in Bosun. Use the *Jump to* links to select the alert you want to edit, then you can use the *template* button next to *macro* to switch between the alert an template sections of the configuration. If an alert has multiple instances you can use `host=xxx,name=xxx` in the *Template Group* section to specify for which tagset you want to

see the template rendered.

```
template os.low.memory {
    subject = {{.Last.Status}}: Low Memory: {{.Eval .Alert.Vars.q | printf "%.0f"}}% Free
Memory on {{.Group.host}} ({{.Eval .Alert.Vars.free | bytes }} Free of {{.Eval
.Alert.Vars.total | bytes }} Total)

    body = `
    <p><a href="{{.Ack}}">Acknowledge</a> | <a href="{{.Rule}}">View Alert in Bosun's Rule
Editor</a></p>
    <p><strong>Alert Key: </strong>{{printf "%s%s" .Alert.Name  .Group }}</p>
    <p><strong>Incident: </strong><a href="{{.Incident}}">#{{.Last.IncidentId}}</a></p>
    <p><strong>Notes: </strong>{{html .Alert.Vars.notes}}</p>

    <strong>Graph</strong>
    <div>{{.Graph .Alert.Vars.graph .Alert.Vars.graph_unit}}</div>
    `
}

notification sample.notification {
    email = alerts@example.com
}

alert os.low.memory {
    template = os.low.memory
    $notes = Alerts when less than 5% free, or less than 500MB (when total > 2GB). In Linux,
Buffers and Cache are considered "Free Memory".

    $default_time = "2m"
    $host = wildcard(*)
    $graph = q("avg:300s-avg:os.mem.percent_free{host=$host}", "1d", "")
    $graph_unit = Percent Free Memory (Including Buffers and Cache)
    $q = avg(q("avg:os.mem.percent_free{host=$host}", $default_time, ""))
    $total = last(q("sum:os.mem.total{host=$host}", $default_time, ""))
    $free = last(q("sum:os.mem.free{host=$host}", $default_time, ""))

    #Warn when less than 5% free or total > 2GB and free < 500MB
    warn = $q < 5 || ($total > 2147483648 && $free < 524288000)
    #Crit when less than 0.5% free
    crit = $q <= .5
    critNotification = sample.notification
}
```

After you test the alert on the Rule Editor page you can use the *Results* tab to see computations, *Template* to see the rendered alert notification, and *Timeline* to see all alert incidents (only when *From* and *To* dates are specified).



Embedded Templates and CSS Styles

You can embed another template body into your template via `{{template "mysharedtemplate" .}}` to reuse shared components. Here is an example that creates a header template that can be reused at the top of all other template bodies. It also uses CSS to stylize the output so that it is easier to read. Note that any `<style>...</style>` blocks will be converted to inline CSS on each element so that email clients like Gmail will render the output correctly.

```
template header {
    body = `
    <style>
    td, th {
        padding-right: 10px;
    }
    a.rightpad {
        padding-right: 10px;
    }
    </style>
    <p style="font-weight: bold; text-decoration: underline;">
        <a class="rightpad" href="{{.Ack}}">Acknowledge</a>
        <a class="rightpad" href="{{.Rule}}">View Alert in Bosun's Rule Editor</a>
        {{if .Group.host}}
            <a class="rightpad"
```

```
href="https://opserver/dashboard/node?node={{.Group.host}}">View {{.Group.host}} in
Opserver</a>
            <a
href="http://kibana/app/kibana?#/discover?_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(fro
15m,mode:quick,to:now))&_a=(columns:!(_source),index:%5Blogstash-
%5DYYYY.MM.DD,interval:auto,query:(query_string:(analyze_wildcard:!t,query:'logsource:{{.Group.host}}'
{{.Group.host}} in Kibana</a>
        {{end}}
    </p>
    <table>
        <tr>
            <td><strong>Key: </strong></td>
            <td>{{printf "%s%s" .Alert.Name  .Group }}</td>
        </tr>
        <tr>
            <td><strong>Incident: </strong></td>
            <td><a href="{{.Incident}}">#{{.Last.IncidentId}}</a></td>
        </tr>
    </table>
    <br/>
    {{if .Alert.Vars.notes}}
        <p><strong>Notes:</strong> {{html .Alert.Vars.notes}}</p>
    {{end}}

    <p><strong>Tags</strong>
    <table>
        {{range $k, $v := .Group}}
            {{if eq $k "host"}}
                <tr><td>{{$k}}</td><td><a href="{{$.HostView $v}}">{{$v}}</a></td></tr>
            {{else}}
                <tr><td>{{$k}}</td><td>{{$v}}</td></tr>
            {{end}}
        {{end}}
    </table></p>
    `
}
```

After which you can add start your templates with `body = `{{template "header" .}}` to get the following output at the top:

Generic Template with optional Graphs

It often is faster to use a generic template when first creating a new alert and only specialize the

template when you need to display more information. The following template will display a subject with a numerical value, custom formatting, and description string and then a body with up to two graphs. If no graph variables are specified it will instead list the computations used in the alert. The generic template also uses the name of the alert to generate the subject (replacing dots with spaces) and checks for variables to exist before using them to prevent errors.

```
#See Embedded Templates and CSS Styles example for header template
template header { ... }

template computation {
    body = `
    <p><strong>Computation</strong>
    <table>
        {{range .Computations}}
            <tr><td><a href="{{$.Expr .Text}}">{{.Text}}</a></td><td>{{.Value}}</td></tr>
        {{end}}
    </table></p>`
}

template generic_template {
    subject = {{.Last.Status}}: {{replace .Alert.Name "." " " -1}}: {{if
.Alert.Vars.value}}{{if .Alert.Vars.value_format}}{{.Eval .Alert.Vars.value | printf
.Alert.Vars.value_format}}{{else}}{{.Eval .Alert.Vars.value | printf
"%.1f"}}{{end}}{{end}}{{if .Alert.Vars.value_string}}{{.Alert.Vars.value_string}}{{end}}{{if
.Group.host}} on {{.Group.host}}{{end}}

    body = `{{template "header" .}}

    {{if or .Alert.Vars.generic_graph .Alert.Vars.generic_graph_all}}
        <strong>Graph</strong>
        {{if and .Alert.Vars.graph_unit .Alert.Vars.generic_graph}}
            <div>{{.Graph .Alert.Vars.generic_graph .Alert.Vars.graph_unit}}</div>
        {{else if .Alert.Vars.generic_graph}}
            <div>{{.Graph .Alert.Vars.generic_graph}}</div>
        {{end}}
        {{if and .Alert.Vars.graph_unit2 .Alert.Vars.generic_graph2}}
            <div>{{.Graph .Alert.Vars.generic_graph2 .Alert.Vars.graph_unit2}}</div>
        {{else if .Alert.Vars.generic_graph2}}
            <div>{{.Graph .Alert.Vars.generic_graph2}}</div>
        {{end}}
        {{if and .Alert.Vars.generic_graph_all .Alert.Vars.graph_unit}}
            <div>{{.GraphAll .Alert.Vars.generic_graph_all .Alert.Vars.graph_unit}}</div>
        {{else if .Alert.Vars.generic_graph_all}}
            <div>{{.GraphAll .Alert.Vars.generic_graph_all}}</div>
        {{end}}
        {{if and .Alert.Vars.generic_graph_all2 .Alert.Vars.graph_unit2}}
            <div>{{.GraphAll .Alert.Vars.generic_graph_all2 .Alert.Vars.graph_unit2}}</div>
        {{else if .Alert.Vars.generic_graph_all2}}
            <div>{{.GraphAll .Alert.Vars.generic_graph_all2}}</div>
        {{end}}
    {{else}}
        {{template "computation" .}}
    {{end}}`
}


alert puppet.last.run {
    template = generic_template
    $timethreshold = 60
```

```
    $timegraph = 24h
    $notes = Checks if puppet has not run in at least ${timethreshold} minutes. Doesn't
include hosts which have puppet disabled.

    $generic_graph = q("sum:300s-max:puppet.last_run{host=*}", "$timegraph", "") / 60
    $graph_unit = Minutes since Last Puppet Run
    $generic_graph2 = q("sum:300s-max:puppet.disabled{host=*}", "$timegraph", "")
    $graph_unit2 = Puppet Disabled=1 Enabled=0

    $value = last(q("sum:puppet.last_run{host=*}", "6h", "")) / 60
    $value_format = It has been %.0f
    $value_string = ` minutes since last run`
    $disabled = max(q("sum:puppet.disabled{host=*}", "60m", ""))
    warn = ($value > $timethreshold) && ! $disabled
    warnNotification = default
    runEvery = 15
}
```

Which will produce a subject like "warning: puppet last run: It has been 62 minutes since last run on co-lb04" and include a graphs of last_run and disabled for that host. If you want to graph all results for a query instead of just the matching tagsets you can use **$generic_graph_all** and **$generic_graph_all2** as the variable names.

Read Templates: Overview online: https://riptutorial.com/bosun/topic/715/templates--overview

Credits

S. No	Chapters	Contributors
1	Getting started with Bosun	Community, Greg Bray, RamenChef
2	Alerts: Advanced Scoping	Kyle Brandt, Whisk
3	Complete Examples	Kyle Brandt, Trent Scholl
4	Expression Tips and Tricks	Kyle Brandt
5	lscount	Mark Henderson
6	lsstat	Mark Henderson
7	Notifications: Chat Systems	Andy Kruta, Greg Bray
8	Notifications: Overview	captncraig, Greg Bray
9	Packages and Initialization Scripts	Greg Bray, Mark V, Vincent Flesouras
10	Scollector: External Collectors	Gary W, Greg Bray
11	Scollector: Overview	Greg Bray
12	Scollector: Process and Service Monitoring	Greg Bray
13	Silencing and Squelching Alerts	Xavier Nicollet
14	Templates: Graph and GraphAll	Greg Bray
15	Templates: HTTPGet and HTTPGetJSON	Greg Bray
16	Templates: Overview	Greg Bray, Vitor