Riemann - riemann.io

- Stream (event) processing library/framework.
- DIY alerting system.
- Clojure (JVM)
- Processes our metric stream from Kafka.
- Metrics are piped directly into Riemann (comes with a OpenTSDB listener)
- RH primitives to make building alerts easier.
- Can alert into log files, email, Slack, Opsgenie.
- Our system is meant to be used in conjunction with OpenTSDB.
- ELK front end.

Riemann's philosophy: "**mostly correct information right now is more useful, than totally correct information only available once the failure is over.**"

You look at timeseries charts in OpenTSDB and once you are familiar with your data, you define Riemann rules to trigger alerts when “bad things happen”

Clojure is a LISP - (Normal Polish) prefix notation

```
user=> (/ (- 5 2) (+ 3 4))
3/7
user=> (defn sum [a b] (+ a b))
#'user/sum
user=> (sum 31 11)
42
user=> (def mylist (list :a :b :c))
#'user/mylist
user=> mylist
(:a :b :c)
user=> (conj mylist :d)
(:d :a :b :c)
user=> mylist
(:a :b :c)
user=>
```

Riemann rules:

You don’t have to write clojure code for the most part.

Most simple cases have pre-defined templates you can use.

- single-metric-rules: Act on a single metric like "cpu.utilization" and trigger alerts based on the filters/conditions you specify. Uses the current metric value in rule comparisons.

- multi-metric-rules: Act on a group of metrics like “nginx.rate status=200”, “nginx.rate status=201” etc.. These metrics are combined using a transformation function. This result is then used in rule comparisons.

Riemann filters:

A Riemann rule allows you to pick/match specific events based on the **metric-name** and **tags**. You then ask Riemann to push (**thread-through**) these matching events through some **filters** - if events make it all the way through the **filters**, Riemann will **alert** you about them. **Filters thread right to left**. The rightmost filter is applied first, then the one to it's left and so on..

Filters let you specify conditions like:

- When metric (event) value is over 50.
- Aggregate events over 5 minutes and when avg is less than 10.
- Let me know only if this happens like 5 times in a row.

In addition, you can say things like:

- Only during pre-market hours.
- Only on weekdays.
- Only when markets are open (trading days).

single-metric-rule “CPU steal”

```
(def rules
  {:single-metric-rules
   [
    ;; During trading days, when the mean cpu steal (on any box) is higher
    ;; than 20 over a 5 (default) minute interval - log an info message and
    ;; e-mail ops right away (single occurrence). This is a per host check.
    {
     :metric-name "cpu.utilization"
     :metric 20
     :type "steal"
     :thread-through [when-higher mean trading]
     :alert [log-info (email "operations")]
     :metric-modifiers {:description "CPU steal"
                        :flap-threshold 1}
    }
  ]})
```

Quiz

```
(def rules
  {:single-metric-rules [
    {
     :metric-name "df.bytes.percentused"
     :metric 85
     :mount "/"
     :roles ["elk-data", "kafka-server"]
     :thread-through [when-higher minimum extended]
     :alert [log-warning (opsgenie "operations")]
     :metric-modifiers {:description "Low disk space on /"
                        :flap-threshold 5
                        :interval 600
                        :by-host false}}
]})
```

filters for final comparison - REQUIRED

These **filters** **terminate* a top level** **thread-through** **chain**. You will place these filters at the very end of the chain (leftmost).

- when-lower
- when-higher
- When-equal
- When-not-equal
- When-lower-r** (takes a range [lwm hwm]) - alerts when metric goes under low water mark and recovers when metric goes above high water mark.
- When-higher-r** (takes a range [lwm hwm]) - alerts when metric goes over high water mark and recovers when metric goes below low water mark.
- When-in-range (takes a range [min max] - inclusive)
- when-out-of-range (takes a range [min max] - exclusive)

*This means that you can't have a chain that reads "when mean is less than 20, then calculate the max in that group, and if that max is over 50, alert me!

**These filters help cut down flapping and noisy alerts.

modifiers - OPTIONAL

Filters take modifier that change their behaviour slightly. The default value is shown in parens.

- by-host (true) - splits stuff by host
- interval (300) - usually used in aggregate rules, how long do you group for.
- description - friendly name to use in alerts
- exclude-roles ([]) - ignore this rule on these hosts
- flap-threshold (3) - trigger notification after x attempts.
- half-life (7) - used to dampen events in ewma calculations.

You should always add a description modifier that tells you what the rule/alert is about.

If you don’t, Riemann will generate one for you that will be super long and confusing.

filters based on time - OPTIONAL

These **filters** are the simplest of the lot. They pass events only during the specified intervals. The default is to pass all events through all the time. These filters don’t take any modifiers.

I tend to use them as the first filter (rightmost) since they cut out the amount of work Riemann has to do. These filters are pretty handy, if you don’t want to wake up in the middle of the night..

- market - active during market hours only
- pre-market - active during pre-market (extended) hours only
- after-market - active during after market (extended) hours only
- extended - active during extended hours (pre + market + post)
- trading - active on days when NYSE is open/trading
- non-trading - active on days when NYSE is closed
- weekend - active on weekends
- weekday - active weekdays

filters for aggregation - OPTIONAL

These **filters** mostly group events and calculate stuff about them. They are optional. When you aggregate events, you usually specify a role tag as a part of the tags.

If you want to aggregate events by role, remember to set the :by-host modifier to false. The interval (default 300s) modifier usually goes along with these filters.

- mean
- minimum
- maximum
- num-events
- std-dev
- sum
- rate
- exp-moving-avg (uses half-life)
- median
- percentile-99th
- percentile-95th

Riemann rule definitions - code organization.

As you create new sets of rules for your metrics, it's a good idea to create your own namespace for your rules. Look at existing consumers in the riemann repository for examples. If you create a new rule file, add it to consumers/all.clj to start pushing events through the rules.

- All rules are defined in a (def rules) section
- A rule is either a :single-metric-rule or a :multi-metric-rule
- Most rules you will define will be :single-metric-rules
- Submit a diff with your new rules
- Good idea to e-mail stuff to yourself before you enable paging
- Use OpenTSDB (or Elasticsearch's search) to discover metrics

Riemann Elasticsearch/Kibana dashboard

We use Kibana (and Elasticsearch) to provide a dashboard into the current view of the world (from Riemann's perspective). The default dashboard that is recommended by Riemann is pretty bad, so we rolled our own.

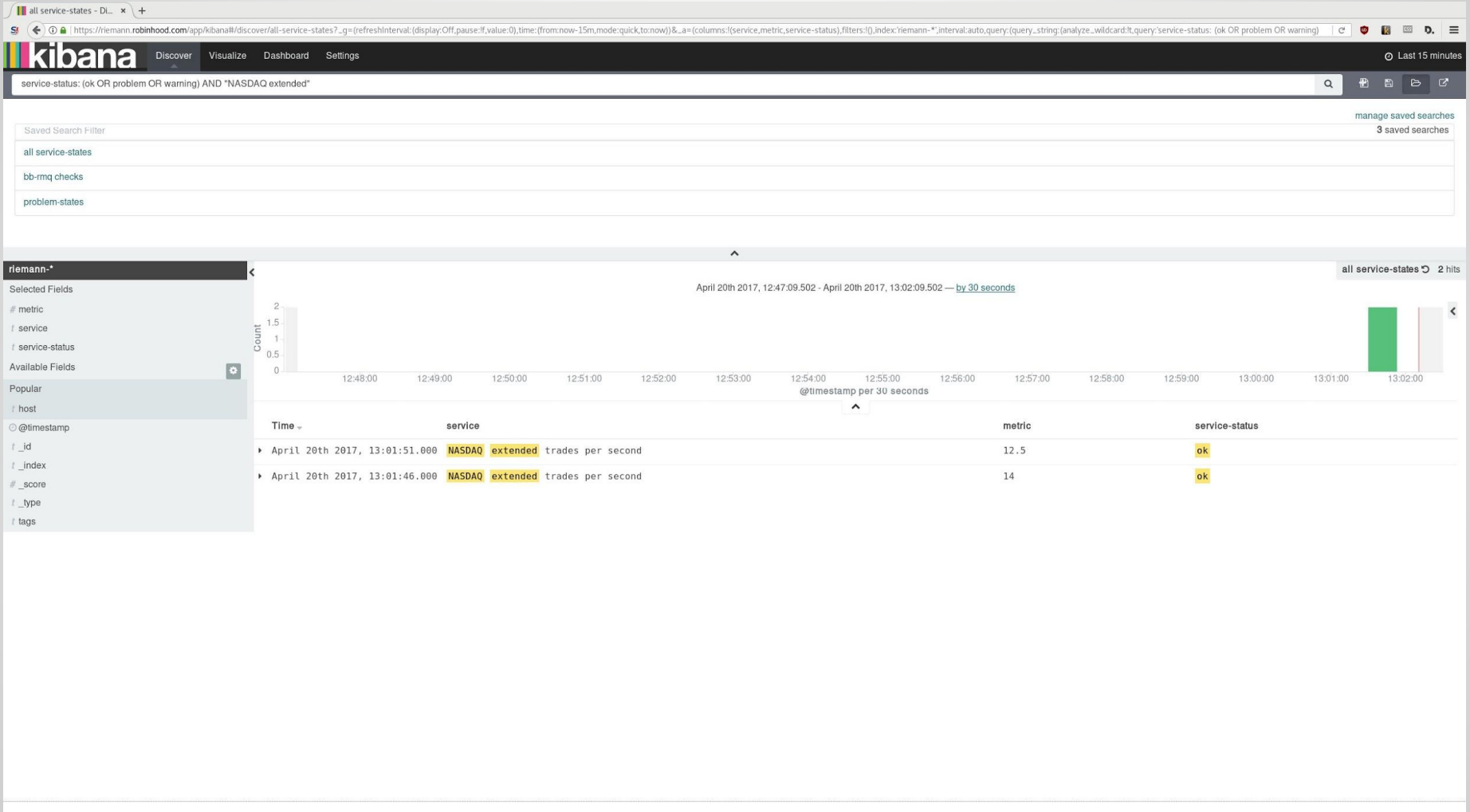
50% of all events flowing into Riemann are indexed into a local Elasticsearch.

All rule output/status is indexed into the local ES.

We have pre-defined ES searches to help you look at the current state of rules.

This is vanilla Elasticsearch/Kibana.. All your leet kibana skills should apply.

We **do not maintain historical charts or trends of rule output in ES**. Use OpenTSDB for that.


all service-states - Di...
https://riemann.robinhood.com/app/kibana#/discover/all-service-states?_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:now-15m,mode:quick,to:now))&_a=(columns:!(service,metric,service-status),filters:!(),index:'riemann-*',interval:auto,query:(query_string:(analyze_wildcard:!t,query:'service-status: (ok OR problem OR warning)
kibana
Discover
Visualize
Dashboard
Settings
Last 15 minutes
service-status: (ok OR problem OR warning) AND "NASDAQ extended"
manage saved searches
Saved Search Filter
3 saved searches
all service-states
bb-rmq checks
problem-states
riemann-*
Selected Fields
metric
t service
t service-status
Available Fields
Popular
t host
@timestamp
t _id
t _index
_score
t _type
t tags
all service-states
2 hits
April 20th 2017, 12:47:09.502 - April 20th 2017, 13:02:09.502 — by 30 seconds
Count
2
1.5
1
0.5
0
12:48:00
12:49:00
12:50:00
12:51:00
12:52:00
12:53:00
12:54:00
12:55:00
12:56:00
12:57:00
12:58:00
12:59:00
13:00:00
13:01:00
13:02:00
@timestamp per 30 seconds
Time
service
metric
service-status
April 20th 2017, 13:01:51.000 NASDAQ extended trades per second 12.5 ok
April 20th 2017, 13:01:46.000 NASDAQ extended trades per second 14 ok

Riemann - combining multiple metrics

Single metric rules look at a single metric :-)

How do you define a rule for “Alert me when the overall API error rate goes above 5%” ?

We currently collect metrics for http 2xx, 3xx, 4xx and 5xx status codes.

You can pre-compute this “API error rate” on every single nginx box - this means you have to modify the nginx collector, push it out to every single nginx box.

You’d then have to aggregate them in riemann using aggregate functions we saw earlier.

Or you can use multi-metric rules to compute this on the fly.

multi-metric-rule “API error %”

```
(defn http-err-rate [r2xx r4xx r5xx]
  (* 100 (/ r5xx (+ r2xx r4xx r5xx))))

(def rules
  {:multi-metric-rules
   [
    ;; percentage of 500 status code to the other status codes for api
    {
     :filters [
               {:metric-name "nginx.hits.rate"
                :status "2xx"
                :roles ["loadbalancer-marketdata"]
                :thread-through [exp-moving-avg sum]
                :modifiers {:by-host false
                            :interval 60}}
               {:metric-name "nginx.hits.rate"
                :status "4xx"
                :roles ["loadbalancer-marketdata"]
                :thread-through [exp-moving-avg sum]
                :modifiers {:by-host false
                            :interval 60}}
               {:metric-name "nginx.hits.rate"
                :status "5xx"
                :roles ["loadbalancer-marketdata"]
                :thread-through [exp-moving-avg sum]
                :modifiers {:by-host false
                            :interval 60}}]
     :metric [2 5]
     :transform http-err-rate
     :thread-through [when-higher-r]
     :alert [log-warning (email "operations")]
     :modifiers {:description "Marketdata HTTP error percentage"
                 :by-host false}}}])
```

multi-metric-rule “Free memory”

```
(def rules
  {:multi-metric-rules
   [
    ;; Alert when free memory, caches and buffers are all low
    {
     :filters [
               {:metric-name "proc.meminfo.memfree"
                :thread-through [mean trading]
                :modifiers {:interval 60}}
               {:metric-name "proc.meminfo.cached"
                :thread-through [mean trading]
                :modifiers {:interval 60}}
               {:metric-name "proc.meminfo.buffers"
                :thread-through [mean trading]
                :modifiers {:interval 60}}
               ]
     :metric [104857600 524288000]
     :transform max
     :thread-through [when-lower-r]
     :alert [log-warning (email "operations")]
     :modifiers {:description "Available memory"}}
    ]})
```