Riemann tool for Real-time Distributed System Monitoring

Bhoomi A. Thaker

Student
MCA Department,
Sardar Patel Institute of Technology,
(Autonomous Institute Affiliated to Mumbai University)
Mumbai, India.

***Abstract*: The increasing implementation of Distributed systems for faster data accessing, storing and processing brings with it many real time warnings, critical conditions making the system slowdown or in the worst condition to crash, affecting the organisation's market value. Hence, setting up monitoring system in large/small organisation for their complex distributed IT infrastructure is a must. This paper talks about one such open source, event-based, scalable, real-time monitoring of all the system and alert when failure occurs; Riemann. This paper describes briefly Riemann's working concepts, monitoring approaches, protocol used along with its implementation procedure, its advantages and usage pre-requisite.**

***Keywords*: Riemann, open-source, event-based monitoring, distributed systems, data stream, push model, real-time monitoring, scalable.**

I. INTRODUCTION

Monitoring has become the integral part of the distributed system. Appropriate selection and designing of monitoring system makes win half a battle in the race providing fast, efficient and reliable cutting edge technologies. The vantages of monitoring tools/systems like early notification, warning or alert of the failure through emails, texts, or any integrated visualization tool helps early stage fixation of issue. The automation of the solutions to trigger when system deviates from the expected routine or on alerts for system down/failure can further help curb the system downtime and human involvement. The selection of appropriate monitoring tool for a system is very critical and is based on the attributes of the system and how it is being used by the organisation. For example, there are open source and paid tools providing different range of functionalities, with simple but detailed and easy to understand UI or complex UI.

This paper presents an open source tool Riemann which can be configured according to the requirements. It supports wide range of client languages to connect to its server like JAVA, C#, PYTHON, PERL, C++, CLOJURE, GO, NODE.js and wide range of tools/programs/plugins to integrate like Collectd, Chef, Cassandra, Graphite, HBase, InfluxDB, Grafana, Nagios, Pupet, Tensor etc[1]. It also has its collection of API to use easily. It is easy to understand, use, configure, highly scalable and reliable.

Goal of this paper is to describe the functioning of Riemann tool with its components which is covered in first half and an implementation, covered in later half.

II. LITERATURE SURVEY

1.The paper, Tools for Distributed Systems Monitoring by Lukasz KUFEL, 2016[2], describes monitoring approaches for designing monitoring systems tool emerging over time and requirements, viz, *Agent-based approach, Agentless Approach, Hybrid approach and Data Streams approach*.

Riemann Monitoring works on the idea of *Data Streams*. In the shift of distributed system to heterogenous cloud environments, new methodology was created to quantify availability and performance of the application and services called Data Streams approach. This method centres around business transactions execution and approves end to end from the client system initiating the task to the server in the infrastructure.

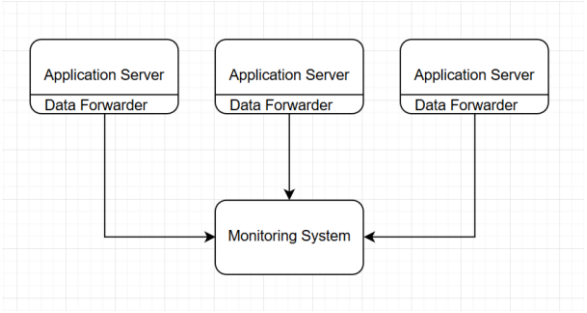


Figure 1: Data Stream Approach Architecture

There is a data forwarder in every application server as agent that is responsible for sending the monitoring metrics to monitoring system as streams(fig.1). This enables IT operations team to vigil application performance in near real time dashboards, generate alerts based on event states and understand user experience with the application or services. Data Streams approach uses Push Model to gather data from the client and send it to monitoring server.

2. The paper, Monitoring Distributed System with Riemann by Simon Obetko, 2016[3], describes Riemann's Push model that ingest the metrics from the client into the monitoring server. The concept of push model is, every client or host initiates the communication to the monitoring server and sends the metrics data without the server to poll from time to time and preventing the communication overhead. It consequently reduces the monitoring system's load to just accept and store the data.

Push models are decentralized, making them easier to scale and focus on quantifying the application's performance and health from the collected data.

III. RIEMANN INTRODUCTION

Riemann is an Open-Source event-based monitoring tool for distributed systems like hosts, servers and applications. It monitors by collecting the events thrown by the systems and feeding it into a powerful Stream Processing Language to be further filtered, processed and summarized. It acts as a routing engine(fig.2) as it routes the events from the distributed systems and further to the integrated storage or analysing or alerting services. Riemann is a stand-alone system, meaning it is not required to be integrated to any other tools to process and view the output though losses the output data fast. Hence, to use it for analysis purpose integrate it with any datastore like influxDB, graphite. It has its own graphing and alerting system that can be configured(exemplified further). Because of its low latency it helps us to get near real time data.

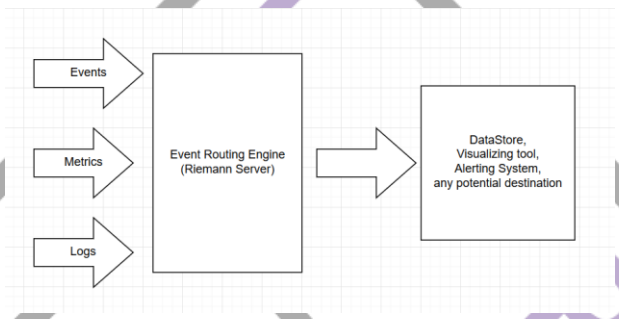


Figure 2: Riemann server as event routing engine[4]

It is fast, easy and highly configurable. Riemann helps to build different checks and alerts on state or metric of incoming events like sending mails for every application exception, latency distribution check of the web application, integrate it with PagerDuty for SMS or phone alerts, combining statistics from every node in a cluster and forwarding it to a datastore, say, influxDB or Graphite, tracking user activity by seconds. Compared to other monitoring systems that run on polling idea, Riemann makes client to "push" their event to the server, which makes them visible within milliseconds, helping to see the problems faster.[5]

Riemann is licensed under *Eclipse Public License*[6]. Riemann is written by *Kyle Kingsbury*, together with *Pierre-Yves Ritschard*, *James Turnbull* in clojure. Its configuration is a clojure program, for its syntax is concise, regular and extendable. Riemann requires JVM to run on.

IV. RIEMANN CONCEPTS

To configure and setup the Riemann monitoring tool, there are some basic keywords and its functions that needs to be understood[7]:

1. Events
   Each event is a struct of the combination of defined fields and custom fields and are immutable maps to the event. These events are sent over Protocol Buffers. The following table describes the defined fields(optional).
   - **host** : A hostname, e.g. "riemannClient.com"
   - **service**: Name of the service, e.g. "read_write_latency_check"
   - **state**: String less than 255 bytes, e.g. "ok" "warning" "critical"
   - **time**: Time of the event in unix epoch seconds
   - **description**: Freeform text describing event
   - **tags**: Freeform list of strings
   - **metric**: A number associated with this event
   - **ttl**: A floating-point time, in seconds, that this event is considered valid for. Expired states may be removed from the index.

2. Streams
   Each event is processed according to the streams defined in the (streams….) riemann.config file. Following is the simple example

```
(streams
  (where (and (service #"^riak")
              (state "critical"))
         (email "delacroix@vonbraun.com")))
```

Figure 3: stream example[6]

The fig.3 describes the stream that sends critical events from services that start with "riak" to the specified email. The "where" in the stream is the predicate expression to filter out services. This streams can be structured as required to process an event, using clojure language.

3. Index
   Each event's state is inserted into an index table by the Riemann server and is uniquely indexed by its host and service. The index table can be queried by riemann clients, dashboard and streams to get the system status. Events entered into the index table have a :ttl field(default value is 60 secs) indicating events validity. On expiration of the event's validity it is again send to the streams with state as "expired" as show in the fig 4.

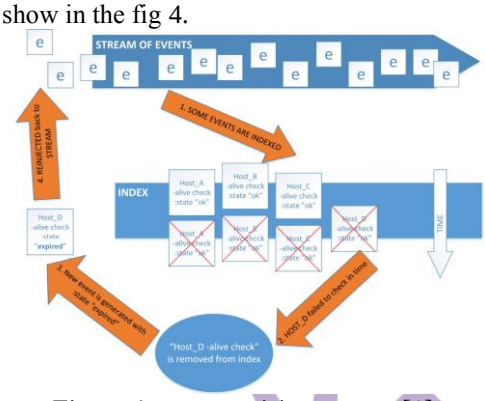


Figure 4: event expiring process[1]

4. Protocols to use
   Recommended protocol is TCP. Here, UDP is designed to drop data for speed, so use UDP where discarding large part of data is ok. Following are the default ports for its protocol which can be changed in the configuration file.

- TCP runs on port 5555
- TLS runs on port 5554
- UDP runs on port 5555
- Websockets run on port 5556

V. IMPLEMENTATION

To demonstrate the working of Riemann I have configured Riemann on Linux Ubuntu that is serving the windows 10 client on which Riemann-java-client component runs and sends its data to the server that can be view on Riemann Dashboard.

Riemann Configuration

1. As mentioned above, Riemann runs on JVM. Hence, install jre/jdk on to the system using the following command,
   - *sudo apt install default-jre OR sudo apt install default-jdk*
2. Download the latest Riemann deb file from its official site http://riemann.io/ and
   - *run dpkg -i riemann_0.3.2_all.deb*
3. As Riemann client tools works on ruby packages, installing ruby is the next step.
   - *sudo apt install ruby*
4. Using gem command we install Riemann tools
   - gem install riemann-tools riemann-dash riemann-client
5. In /etc/Riemann/Riemann.config file replace the default ip""127.0.0.1" to "<your servers ip>"
6. In the same path that of Riemann.config file create a new file config.rb and add "set :bind, "0.0.0.0"". This helps to connect the server via internet.
7. Start the service by, service Riemann start.
8. In the same directory path type, riemann-dash and hit enter. This start the Riemann dashboard at port "4567".
9. On proper execution of Riemann configuration steps above you can view the dashboard on the browser using url- "http://<server ip>:4567" as follows:

Figure 5: Default Dashboard

10. To get the Riemann server default metrics, perform Ctrl+click and 'e' on the configuration page to get the following

Figure 6: Dashboard configuration panel

11. Go the title dropdown and select grid. Enter in the query section "true" and click on apply to get.

Figure 7: Default Riemann server metrics

12. Successful loading of the above page indicates Riemann server is ready to accept metrics from client hosts.

Testing

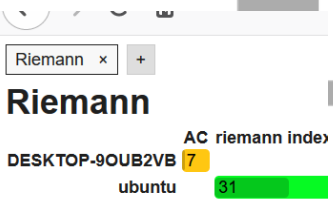
In order to make Riemann server to connect to its client. Riemann provides some language based client plugins as mentioned before[8]. I used Riemann-java-client to demonstrate the implementation. I composed a code for the service name AC that would create events based on its temperature and send to the Riemann server. I used Maven to get the client plugin by adding the following code in the project section of the pom file.

```
<repositories>
    <repository>
        <id>clojars.org</id>
        <url>http://clojars.org/repo</url>
    </repository>
</repositories>

<dependencies>
    <dependency>
        <groupId>io.riemann</groupId>
        <artifactId>riemann-java-client</artifactId>
        <version>0.5.1</version>
    </dependency>
</dependencies>
```

Below is the code that sends, metrics “temperature” of service named “AC” having tags as “applicance and cold/hot” via Object of RiemannClient class.

```
package com.bhoomi.riemannJava;

import java.io.IOException;

public class RiemannEvent {

    public static void getNetworkInterface(RiemannClient c) throws Exception {

    public static void main(String[] args) throws Exception, InterruptedException {
        // TODO Auto-generated method stub
        RiemannClient c = RiemannClient.tcp("192.168.153.133", 5555); // creating connection object
        c.connect(); // connecting to riemann server
        int temperature = 1;
        while (true) {
            if (temperature > 30)
                temperature = 1;
            Thread.sleep(1000);
            if (temperature < 23 && temperature > 18) {
                c.event(). // creating event
                        service("AC").state("ok").metric(temperature).tags("appliance", "cold").send(). // sending event
                        deref(5000, java.util.concurrent.TimeUnit.MILLISECONDS);
            }
            if (temperature > 23) {
                c.event().service("AC").state("critical").metric(temperature).tags("appliance", "hot").send()
                        .deref(5000, java.util.concurrent.TimeUnit.MILLISECONDS);
            }
            if (temperature < 18) {
                c.event().service("AC").state("warning").metric(temperature).tags("appliance", "hot").send().deref(5000,
                        java.util.concurrent.TimeUnit.MILLISECONDS);
            }
            temperature++;
        }
    }
}
```

On executing the code we get the metrics on Dashboard as:



=>warning-state(orange)

=>ok-state(green)

=>critical-state(red)

Here, column indicates the service name and rows indicate the host name. The grid display can be changed in its configuration panel (fig. 6)

Advantages

- Push model and event based stream processing architecture.
- Easy to implement, configure and integrate.
- Highly scalable.
- Wide range of tools, plugins and client languages support.

Further Integration

The captured metrics by Riemann tool can further be sent to the responsible team/individual via email or phone message by configuring its supported plugins in the riemann.config file. Riemann can be integrated to any datastore namely, influxDB or graphite to store the generated events data and then get it onto a visualization tool like Grafana in forms of graphs. Since, Riemann is open source, it can be configured using its various supported plugins according to the organisations monitoring norms. To manipulate riemann.config file, clojure[9,10] language is required to be understood.

VI. RIEMANN TOOL, A RIGHT CHOICE!!

If your organisation plans to setup distributed environment that processes real time data, monitoring such environment is a huge, critical, complex and expensive task as you need to understand what to monitor from the large pool of metrices of the many system used and how to get that metrices that is to be monitored. Moreover being a distributed environment, collecting the data from a geographically dispersed locations adds up the cost of communication overhead making it a critical factor for real time data analysis. Riemann, through its event based polling and stream processing technology have helped gathering data convenient. Hence, the organisation needs to focus on what metrices to monitor and not how to get it. Due to its low latency characteristics from using clojure, it helps us to process real time data and get earliest alert/warning to the system's failure/critical health. Riemann is an open source tool, helping the organisation to manipulate its usage easily according to the infrastructure design.

VII. CONCLUSION

Riemann monitoring system is easy to understand and work and implement in small/large/medium size organisation. Importantly it is flexible and scalable as its distributed environment it is implemented for, which is an significant factor in the designing of an IT infrastructure of the organisation. And its near real time failure alerts or warnings helps the organisation, to keep the system health in check; up and running, preventing "system down" situations. Being open source in nature its support for wide range of API's, plugins, tools and clients have further made the Riemann reliable, available and adaptive to the environments need.

REFERENCE

[1] https://github.com/riemann/riemann-java-client

[2] Tools for Distributed System Monitoring by Lukasz KUFEL, 2016. https://content.sciendo.com/view/journals/fcds/41/4/article-p237.xml

[3] Monitoring Distributed Systems with Riemann by Simon Obetko, Brno, Spring 2016. https://is.muni.cz/th/fia5e/fi-pdflatex.pdf.

[4] The Art of Monitoring James Turnbull, March 25, 2019. http://ns2.1888.spb.ru/TheArtOfMonitoring.pdf.

[5] http://riemann.io/

[6]https://github.com/riemann/riemann/blob/master/LICENSE

[7] http://riemann.io/concepts.html

[8] http://riemann.io/clients.html

[9] https://clojure.org/

[10] Using Clojure in Linguistic Computing by Zoltan Varju, Richard Littauer, Peteris Erins. https://pdfs.semanticscholar.org/e86f/e337b79437429e3a55db2e45c6481a071c1d.pdf.