Monitoring Stack for Distributed Systems

by Nikola Ivancevic MVB · Jun. 06, 16 · Performance Zone · Opinion

Microservice architecture on the one hand and distributed systems on the other brought new challenges to the IT world. Here are two of them that are becoming increasingly noticeable: *management of distributed systems with many nodes which involves deployment* and *orchestration and monitoring of those systems*.

Back in the days, you had a single machine and you could scroll down the single log file to figure out what is going on. Now, you have either many small services or a distributed system, and you need to combine a lot of logs together to figure out what is going on. Combining this information from many machines is impossible to do manually, there are a lot of moving parts, so there is a need for a centralized monitoring platform which will aid the engineers operating the systems, and serve the right information at the right time.

When working with distributed technologies which can be deployed across availability zones or regions on AWS, we need information on how each of the components is functioning in order to obtain knowledge on how the whole system works. This is a must in order to do some heavier tuning, and monitoring all parameters is not an easy task.

Our Distributed System Case

We were hired as Cassandra consultants to deal with tight SLA on read requests. We needed to achieve latency on Cassandra cluster under a threshold on five nines (meaning 99.999% of request must be under a threshold). Without a good monitoring stack, we are blind and do not have a good base for tuning. We started our research and decided to build our own solution on open source software out there. Basically speaking, we decided to connect the dots, reuse already built components and bundle them together so we can perform powerful monitoring.

Monitoring Stack

When thinking about monitoring, there are two sub-domains of the problem: **metrics** and **logs**. Each of these is connected to a stream of data coming from the target system.

Speaking more precisely, there is the third part of the monitoring problem: **alerting**. But as the purpose of our monitoring stack is to help us understand and tune an existing system, and not provide a production deployment setup (yet), we left out alerting from the equation for now.

Metrics data stream contains real-time data about application (system components) performances. Metrics data can be seen as a real-time stream of scalar values with timestamps that represent the activities of different system parts. For example, the rate of input requests processed successfully by an HTTP server, the number of malformed packets received by an application module, or the current usage of a disk partition. Metrics values are usually accompanied with tags that provide additional information (e.g. host name of the machine the value is coming from).

Log data stream contains information about different activities and events within the system. Those data are usually in the form of free-form text records or, in some cases, JSON messages.

It is important to note here that log data can also be used as a source of metrics-like data. Log entries could be processed and, for example, certain textual patterns can be counted over time and presented as new metric data (e.g. the number of exception appearances in an application log file).

Because of the differences in their nature and different tools used to handle metrics and logs, it is reasonable to examine them separately.

Metrics

Talking about metrics data handling implies three things:

- collecting
- storing/maintaining/querying and
- visualizing metrics data

Metrics Storing

We focused ourselves on selecting the storage for metrics data first. Somehow it seemed as the central part of the stack and, therefore, the most logical place to start. As the metrics data streams are time series, we started looking for time-series optimized storage engines used for monitoring systems.

We chose InfluxDB as our weapon of choice. It was not just good-with-time-series, it was built specifically as storage for time series data. Some of the key features that were important to us:

- Purpose-built for time series data, no special schema design or custom app logic required
- Time-centric functions and an easy-to-use SQL-like query language
- Simple to install with no dependencies
- A native HTTP API

The integration support for various metrics collecting tools and protocols, as well as the support for InfluxDB in visualisation tools were very promising.

Metrics Collecting

We were ready to switch to the collecting part. InfluxDB came with its native, so-called *line protocol*—an HTTP-based API for collecting measurements. Ganglia and Graphite compatible inputs were also offered as plugins. So far so good. But we wanted more. We wanted to try a tool that can send various metrics from a lot of moving parts with minimal resource impact on the originating nodes. We heard good things about Riemann, a tool that aggregates events from many nodes using Riemann clients installed and collects them in a centralized place—Riemann server. It speaks Protocol Buffer over TCP and UDP which makes it lightweight and fast. Just the perfect tool for our case. There was a number of out-of-the-box clients and it was quite easy to connect it to Cassandra. Using the metrics-reporter-config and RiemannReporter, internal Cassandra metrics were shipped from every Cassandra node to the Riemann server. On the other side, Riemann server can be easily configured to process/aggregate and forward metrics data towards InfluxDB.

Here is the Ansible role we use to install reporter along with Cassandra on each node.

Metrics Visualization

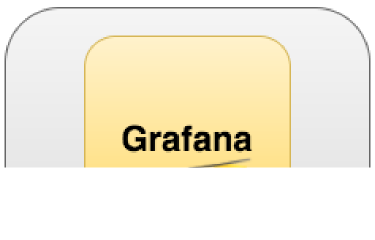
The final, missing step in building metrics stack was to select a tool for visualization. We chose Grafana, for obvious reasons:
it was popular those days, it had a powerful and, at the same time, easy configuration interface, and, finally, it had very good support for InfluxDB (and for Elasticsearch too, which will be explained later).

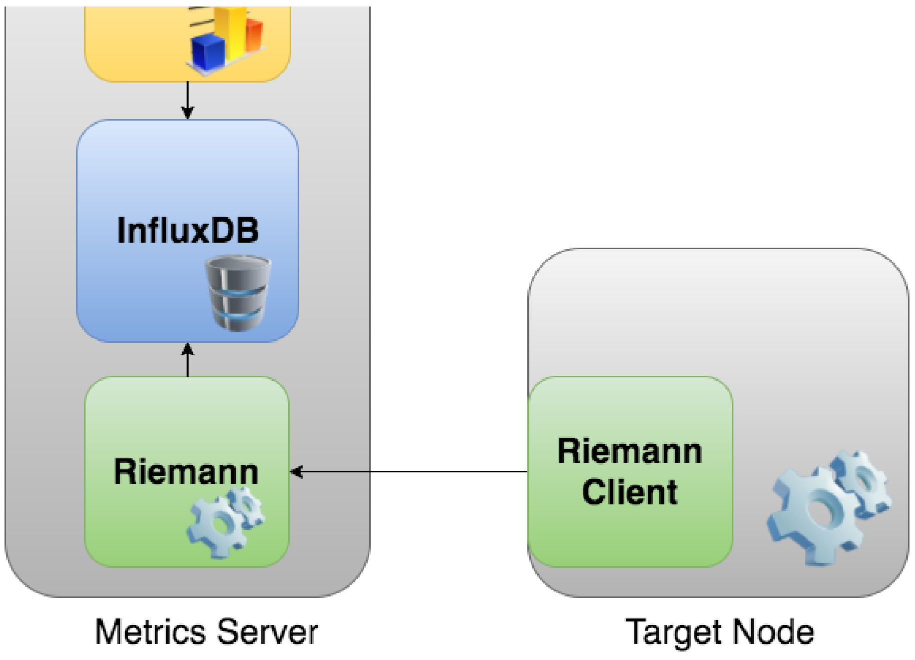
We added Ansible provisioning details in smartcat-ops repository so you can see more details about it.

Now that we had centralized monitoring server accepting Riemann events we could send all sorts of information that we needed to it. Up to this point, we had had all internal Cassandra metrics but we had been missing detailed information on how the system was performing. Cassandra exposed some of them but not detailed enough for us to understand what is going on. We used Riemann tools to send cpu, disk, memory and network statistics so that we could see how the OS was performing at the time we had peaks in Cassandra performance. Details are in Ansible riemann-tools role for metrics which is using riemann-tools project.

Metrics Stack

The following picture gives an overview of the implemented stack:





Logs

Up until this point, we had nice and shiny dashboards with Cassandra metrics and OS metrics. However, one piece was missing to be in full control and that was information in logs. Every time we saw a drop in performance, we associated it to problematic instances and manually investigated log records. Needless to say, it was time-consuming and frustrating—we needed a more efficient way of correlating logs with problematic events.

Keeping the logs of all system parts in a single place is equally important as having all system metrics. Having all log records in one place allows for easy system auditing. Proper tools can then be used to correlate logs coming from different parts of the system with a single search query.

In order to implement a full log monitoring stack, the following three parts have to be realized:

- collecting
- storing, maintaining and querying and
- visualizing log records

Fortunately, there is already an open source solution that implements all three things: ELK stack (Elasticsearch - Logstash - Kibana) - the best open-source stack in this domain. Elasticsearch solves the problem of storing, maintaining and querying log records. Logstash offers a number of ways to collect, process and forward log records towards Elasticsearch. And, finally, Kibana is here to help with visualization.

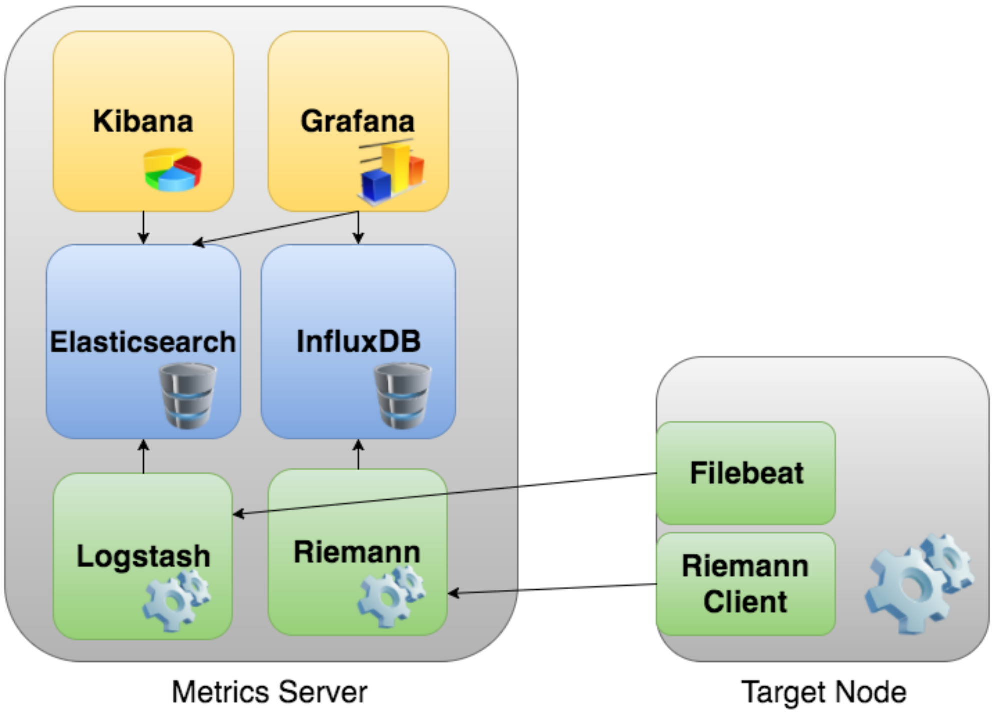
Without any hesitation, we decided to install a log shipper agent (Filebeat) on each node having Cassandra. It was easy enough, we just had to point the agent to the log file location, do some brief configuration and we had a working collector of log events. After that, we needed a centralized place where all log events would be collected.

Now we have a fully functioning ELK stack which runs on the same server as Riemann-InfluxDB-Grafana and we can browse logs on the central place using Kibana. This is nice enough but you still need to jump from Grafana to Kibana and use both tools. It would be perfect if you could see everything on a single dashboard in a single tool. You actually can—Grafana has a great support for Elasticsearch. That means Elasticsearch can be used as a data source for Grafana graphs, specifying search queries directly in Grafana. It's just like all of the pieces have come together. Here you can find a detailed explanation on how to achieve this.

Now we can, for example, plot read request latencies as time series from InfluxDB, and correlate that with things in Cassandra logs by query Elasticsearch. We have obtained a powerful centralized monitoring tool which is flexible and extensible. We also saved a huge amount of time going to each machine and checking what is going on.

Log Stack

The following picture gives an overview of the implemented log stack (together with the metrics stack):



Conclusion

When you are dealing with distributed systems, monitoring is a hard part. Today the trend is to pick up the next hot open source technology and wrap it up into a paying solution with deployment, monitoring and support included. We are working as consultants for Kafka, Spark, Cassandra and, a lot of times, our clients use only open source solutions. Also, you frequently need information from many parts of the system to figure out what is going on. An example of this can be data which started as a request from web UI, went through Kafka with Spark Streaming to be stored in Cassandra and we need to log each step our data went through.

With centralized monitoring (as the one we described here), we are flexible enough and in control. You have a powerful centralized monitoring platform which, for now, can accept Riemann and Logstash events but can be extended in future.

Feel free to check out our smartcat-labs Github account and give us your feedback about this and other useful tools we have there.

Topics: ELK STACK , DEVOPS , GRAFANA , ELASTICSEARCH , DISTRIBUTED APPLICATIONS , CASSANDRA