influxdata®

AN INFLUXDATA CASE STUDY

How RingCentral Created Metrics & Alerts As A Service (MaaS) to Help All of Their Developers to Be Data-Driven

Yuri Ardulov
Principal System Architect, RingCentral

NOVEMBER 2018

Company in brief

RingCentral is an IP telecommunication company that works with its customers to reimagine the world of business communications and collaboration. In a demanding industry where customers have zero tolerance for downtime, RingCentral is providing reliable solutions. RingCentral is headquartered in Belmont, California and has offices in the U.S., the U.K. and China.

With its flexible, cost-effective cloud communications and collaboration solutions, RingCentral has created the ideal workplace, where business can be done more efficiently and effectively. From an all-in-one cloud phone system with team messaging and video conferencing to a complete contact center and more, RingCentral builds solutions for every business, no matter how big or small.

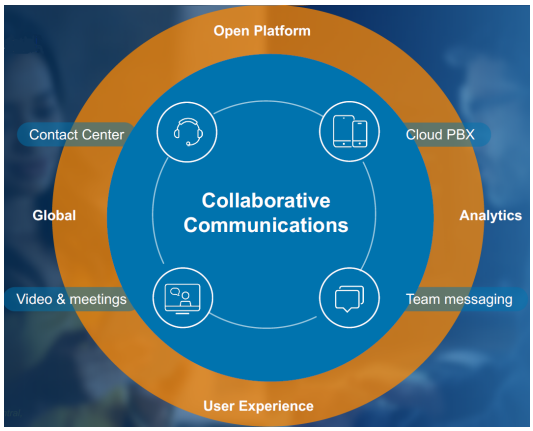
Technology breaks down barriers and unlocks potential, making it easy for people to do their best work together. In today's mobile world, this means giving teams, partners, and customers the ability to communicate, collaborate, and connect the way they want on any device, anywhere, anytime. It's what RingCentral calls collaborative communications, and it's at the heart of everything the company does.

Case overview

RingCentral was faced with the challenge to build a scalable monitoring solution to keep pace with their business and infrastructure growth. They had outgrown the Zabbix monitoring tool set capacity and needed to replace it with a solution that provides high availability and metrics granularity. Additionally, RingCentral established a goal to streamline their processes to more effectively manage development, configuration alterations, as well as metrics and events collection of their ever-growing application environment, currently about 400+ different "homemade" components continuously developed by a team of 1,500 developers.

RingCentral chose to migrate to open source InfluxDB Stack. After the evaluation phase, they deployed InfluxDB Enterprise to handle their metrics and event volume growth, Telegraf as the agent installed in every host (physical or virtual) to collect monitoring data, a Kapacitor pool for no downtime (so no trigger event would pass unnoticed), and an in-house built Kapacitor manager to manage their pool of Kapacitor instances.

Using InfluxDB Enterprise, Kapacitor, and Telegraf, RingCentral's monitoring solution today supports visibility, integrated configuration and alerting for operations efficiency, and quick DevOps cycles for the four pillars of its product (Cloud PBX, contact center, video and meetings, and team messaging) as well as the functionalities built on top of these pillars (open platform, global presence, analytics, and user experience).



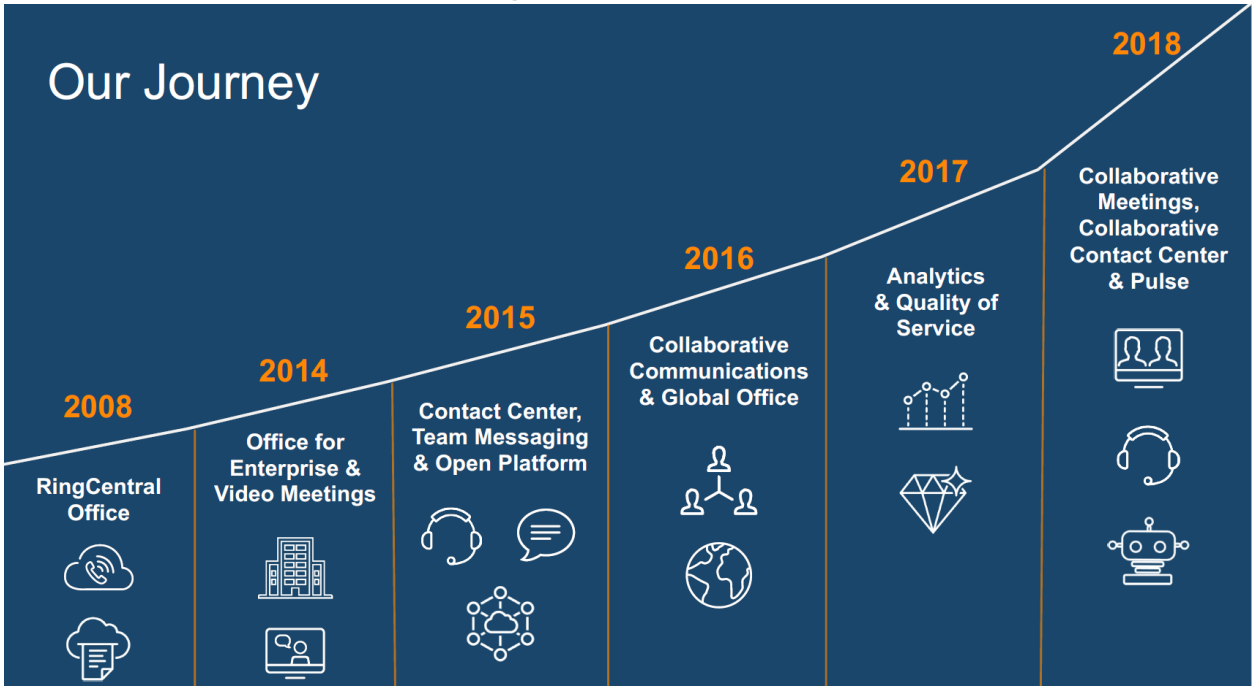
RingCentral's product monitoring solution powered by InfluxDB Enterprise and Kapacitor

> *"Our monitoring team, currently, is very small versus the engineering team which is growing because we have to address all our business needs and all new features."*
>
> ***Yuri Ardulov,** principal system architect*

The business problem

RingCentral's growth was outpacing its monitoring infrastructure year after year. When the company's journey began in 2008, it had just a few thousand customers, a main offering of only phone services and faxes, and a few hundred staff members (engineering, data center operations, and management). Their service consisted of roughly 150 components fully written in-house. Over the next 10 years, they continued adding services and components.

RingCentral Timeline



There were around 60 engineers and roughly 10 operations staff in 2008, but by 2018, their development teams comprised roughly 1,500 developers and an engineering team of about 50 people in charge of all RingCentral infrastructure. The team dealing with the monitoring system itself consisted of only around 10 people.

The monitoring infrastructure had to cope with the company's growing operations and telecom services business encompassing:

- 500,000 business customers
- 10 data centers across the globe (US, Europe, APAC)
- 20K+ servers
- 30K simultaneous phone calls
- 100K faxes per day
- 20M calls per day

As their IT infrastructure becomes more complex, the importance of tracking and understanding the information within the IT environment increased.

The technical problem

Back in 2016, the RingCentral monitoring team started building their CMDB (configuration management database). On top of that, they integrated their CMDB with Jira — their ticketing system. Their monitoring tool at the time, and since inception, was Zabbix.

They initially viewed Zabbix as a Swiss army knife of monitoring since it handles collecting metrics, setting up triggers, managing hosts, and setting up the maintenance node. As RingCentral began to grow its infrastructure, Zabbix lack of high availability and its inability to gain metrics granularity made clear that they need to replace it with a new solution.

Their major presence is in North America, where they have two major data centers spanning over 10,000 hosts each (combination of bare-metal and virtual), over 2.5 million metrics collected with Zabbix, and over 700,000 triggers defined and existent in their Zabbix installation. Their other data centers, located in Europe and beyond, also handled a large number of hosts, metrics, and triggers. They sought to achieve a smooth migration to another database that would be viable for the company.

Also limiting monitoring capability was the very small size of the monitoring team versus that of the engineering and development team which was growing to address new business needs and product features quickly. Matching that growth rate and keeping the system live, and most importantly, for telecom sector: maintaining service quality, was almost impossible, and certainly, unsustainable. They decided to take the delegation approach — "a common strategy of upper management."

RingCentral took the path to delegate or, in other words, do-it-yourself (DIY) framework, by providing a programmable way for developers and operations engineers to self-service their monitoring needs: monitoring of "homemade" systems (RingCentral has 400+ in-house build service components) and operational layer (monitoring CPU, memory, disk space etc.), respectively. In summary, the monitoring team provided a platform and toolsets for the other teams to send metrics and set up alerts of their interest.

To achieve their DIY framework, they defined the technical requirements:

- Alert as code
- Send application metrics without structure requirements
- Provide dashboard as code
- Horizontally scalable infrastructure with high-availability clusters (HA clusters) for metric collection and alerting
- Sandbox for graduation (to test new code before it gets released)

- No hard limitations on cardinality or type of metrics. This requirement revealed itself more challenging than expected: developers initial adopters started sending them — in one of the applications—16,000 metrics every 10 seconds, which quickly consumed storage. Signaling that they would need a scalable storage solution.
- Fully automated service integrating with deployment systems
- No single point of failure, in other words, HA clusters implementation (this was relevant for all components but especially for making sure that no alert trigger event would pass unnoticed).

Solution

> *"Our intention to move from Zabbix to InfluxDB wasn't just because we decided one day we needed to move. It was a necessity of the business because as a company, we very much outgrew the Zabbix capability."*

Why InfluxDB?

RingCentral's decision to migrate from Zabbix to InfluxDB wasn't random — it was a business necessity because the company had outgrown Zabbix capabilities. They first migrated to the open-source version of InfluxDB, and in 2017, started moving to the Enterprise version. InfluxDB Enterprise provided the high availability, scalability and metrics granularity that Zabbix lacked.

Their infrastructure today consists of a broad range of solutions, tools and components. They use Kubernetes and run all their virtualization on VMWare. They have a presence in AWS and in GCP . For monitoring, they have a logging infrastructure where they collect their services' logs during runtime using Elasticsearch, and BMC as a complex event processing (CEP) system to build their log analysis process.

RingCentral's Tools Landscape

RingCentral decided to introduce, to all their engineering team, what they called service manifest following the technical requirements and project goals set for their DIY framework:

- Structure independent
- Collect metrics with high granularity
- Fully automated service
- Integration with deployment systems
- Engineering as self-service (alerting as a code, dashboards as a code, CD support)
- HA implementation

Through this functionality, developers and operations engineering team would be able to express their metrics and alert requirements as a code. The manifest gets compiled by RingCentral's system and promulgates along with the code itself. It successively goes through each stage and then enters the production system automatically without any separate installation or configuration.

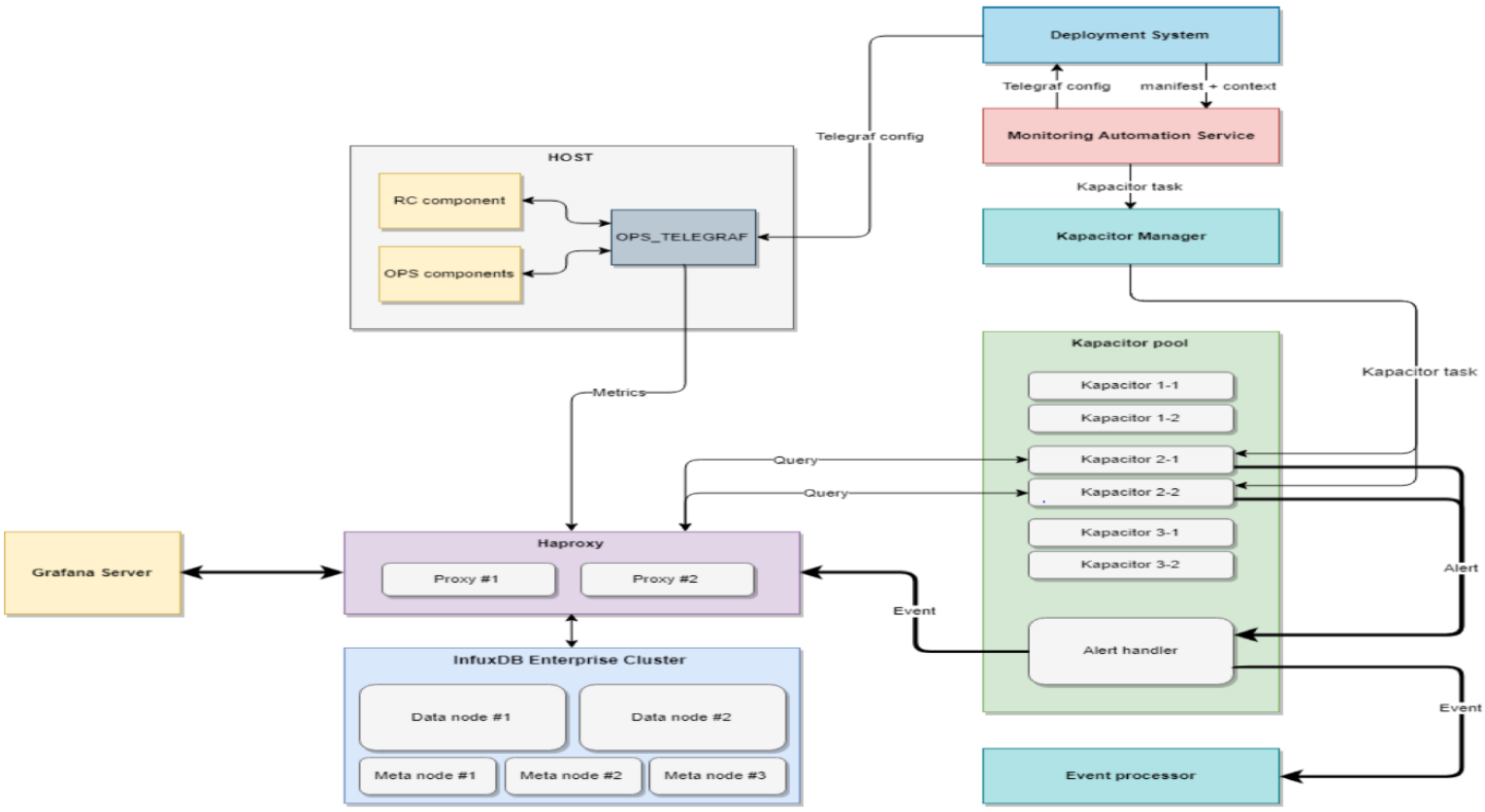
Trigger Manifest Example

```
"measurements": {
"myapp_stats": {
   "telegraf_input": "myapp_input",
   "declaration": {
      "values": ["b_e"]
   },
   "triggers": {
      "trigger_1": {
         "level": "critical",
         "period": "1m",
         "value": ["mean", ["b_e"]],
         "threshold": [">", 1]
      }
```

Technical architecture

> *"One of the requirements for us was no single point of failure. It means we couldn't afford ourselves to have just one Kapacitor task servicing one metrics because if something goes wrong and this particular Kapacitor dies or something, we could not afford to have these metrics overlooked."*

Design of Proposed Solution



Like many engineering lead teams, RingCentral has separate development, performance testing, and staging environments. The above chart shows the design of their system which they planned to install in each of these environments:

- InfluxDB Enterprise cluster and the HAProxy are in front to ensure that all incoming metrics will be balanced there.
- Telegraf is installed on each of the hosts (virtual or physical) for collecting all metrics.
- Through the deployment system, service manifest compiles during the release cycle all of the appropriate configuration that will be delivered on the host on one side.
- All the data is supposed to be delivered for alerting to Kapacitors.

Because one of the requirements was no single point of failure, a two-Kapacitor task runs on the different Kapacitors to duplicate the alert if necessary. As a result, they have a Kapacitor pool to satisfy 50K+ triggers per environment.

Problems with existing Kapacitor (v1.3)

With existing Kapacitor (v1.3), they encountered the following problems:

- Low efficiency (Tasks per Instance)
- Not responsive under high load (API stops functioning)
- Streaming tasks (1000+) cause high CPU load
- Batch tasks (if not grouped by InfluxDB, they consumed all the CPU)
- Low internal concurrency (The alert node stalls on writing into an internal topic, thereby stopping the creation and delivery of alerts)

Test cases

Prior to settling on a solution, they ran different types of test cases. Below are the four test cases they used, and the solution resulting from them.

Test case 1: They used the alert node in Kapacitor using the TCP publisher, with Logstash as a TCP listener, and then pushed the alert into Kafka:
Alert node -> .tcp() -> Logstash tcp listener -> Kafka

Test case 2: They used the same setup as in test case 1, but with an http post on the alert node:
Alert node -> .post() -> Logstash http listener -> Kafka

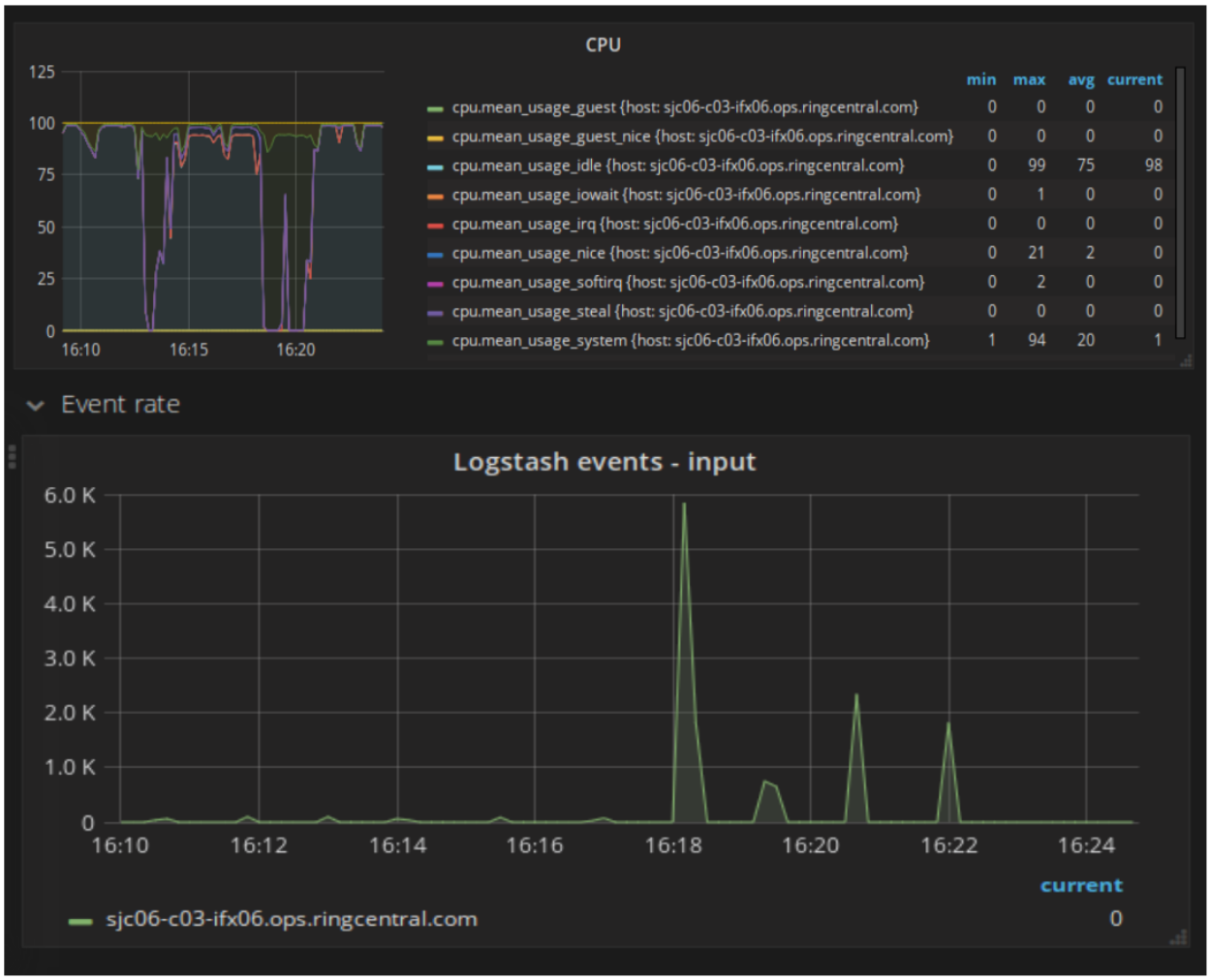
Test case 3: They used the alert node with the InfluxDB output into the Logstash:
Alert node -> InfluxDBOut() -> Logstash http listener as InfluxDB cluster -> Kafka

Test case 4: They used an eval node inside Kapacitor and pushed the data into InfluxDBOut and then pushed data into the InfluxDB cluster:
Eval node -> InfluxDBOut() -> InfluxDB cluster

Each test involved a Kapacitor instance (of 24 CPU cores and 256 GB RAM) and 1000 tasks generated in the following flow:

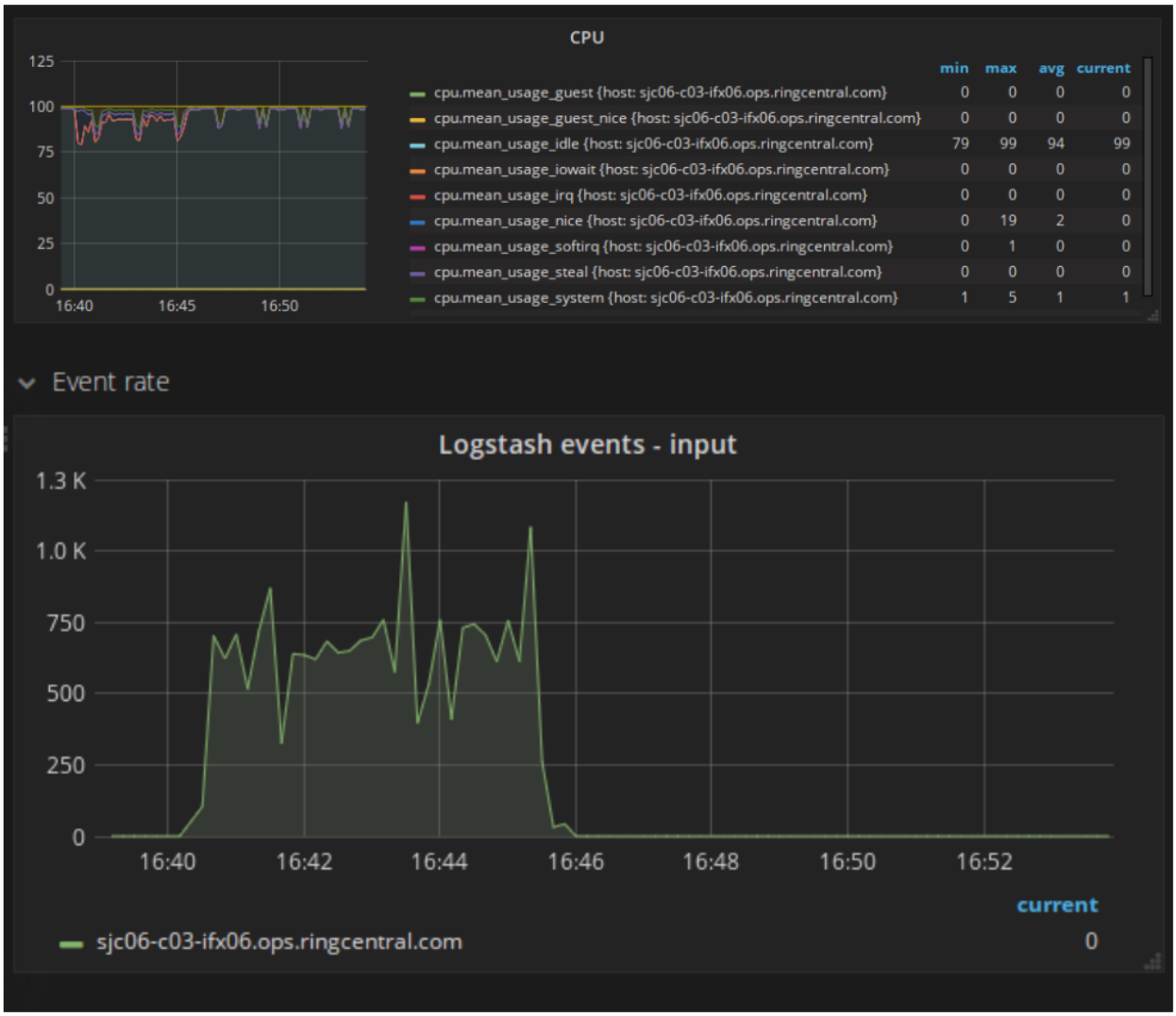
1. Put value=1 to Kapacitor /write API.
2. Wait for 0.5 seconds
3. Put value=0 to Kapacitor /write API.
4. Wait for 0.5 seconds
5. Repeat 1-4 for 100 times

Results: Case 1



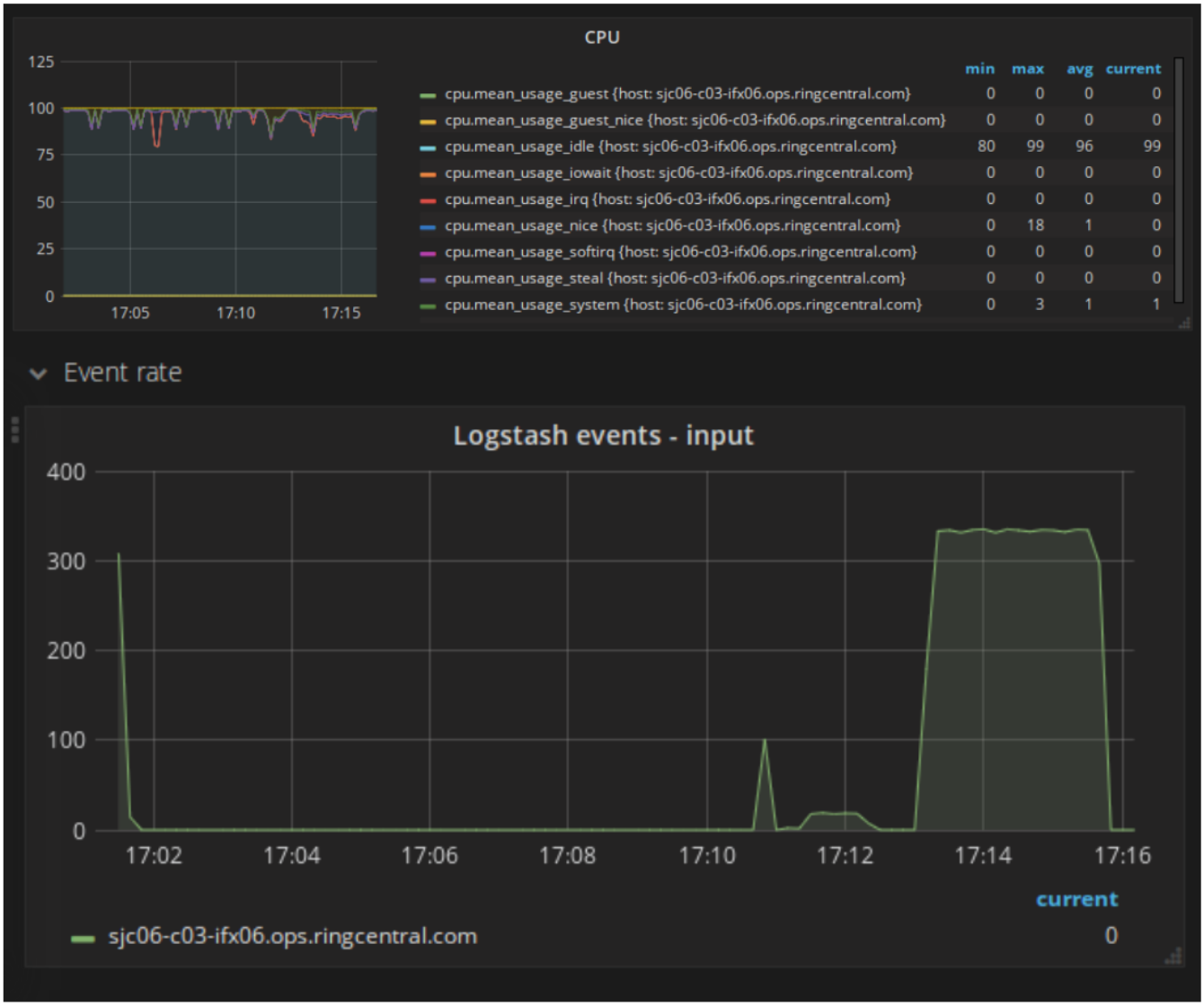
- Delay, between event generation time and time when event received by Logstash, of up to 7 minutes
- Root cause of delay is .tcp method which opens TCP connection for every event

Results: Case 2



- Delay between event generation time and time when event received by Logstash up to 30 seconds.
- Looks much better but delay increases on increased number of tasks

Results: Case 3



- Delay between event generation time and time when event received by Logstash ~3 seconds.

Results: Case 4

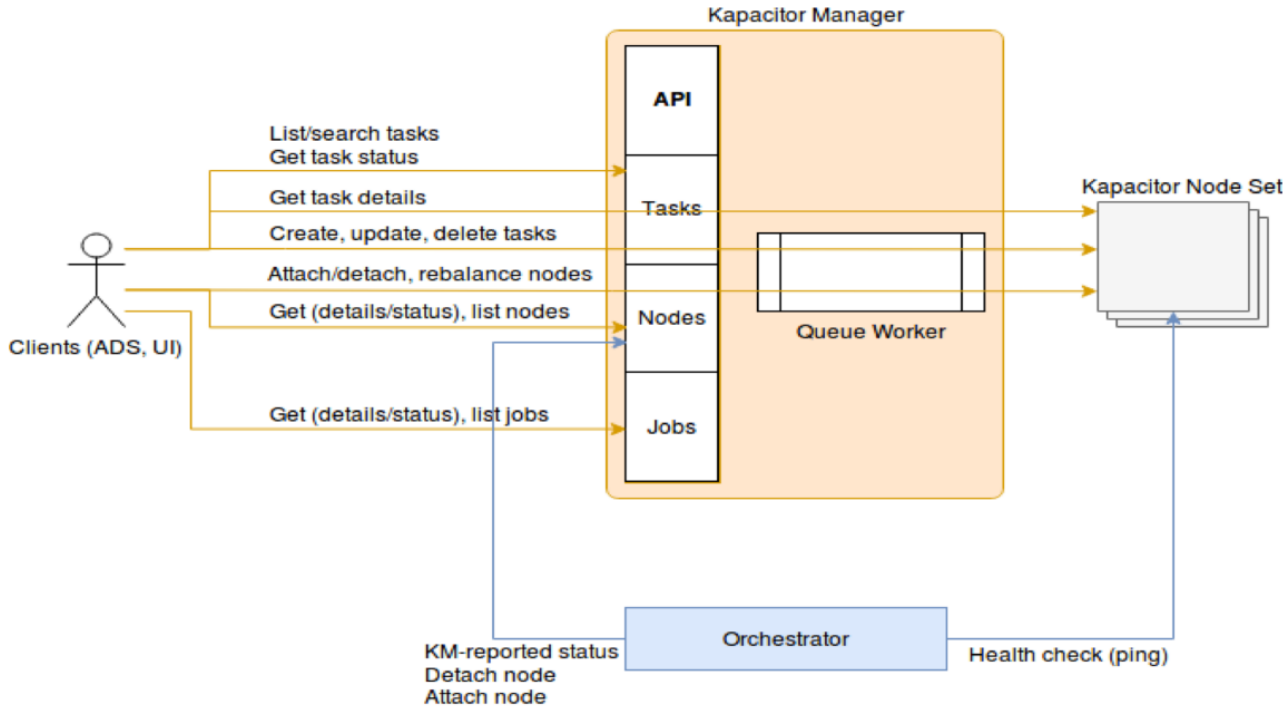
- Serviced per node about 5,000 tasks which were generated without encountering any problems with the CPU or overloading the Kapacitor node
- Events received by InfluxDB cluster in near real-time.

Building Kapacitor manager in-house

RingCentral designed Kapacitor manager to meet their alerting requirements. When they first set up the project, their goals in the initial stage and pertaining to alerting were:

- Scalable solution: 50K+ alerts per location
- Increase efficiency –> Maximize the ratio of tasks per instance
- Manageable solution -> Dynamically change task allocation and balancing
- No single point of failure
- Housekeeping capabilities
- Task sandboxing as a part of the service
- RESTful service

Kapacitor Manager (KM): Functional Description



RingCentral's Kapacitor manager consists of a few instances which are combined and based on the persistent queue. It provides three types of APIs:

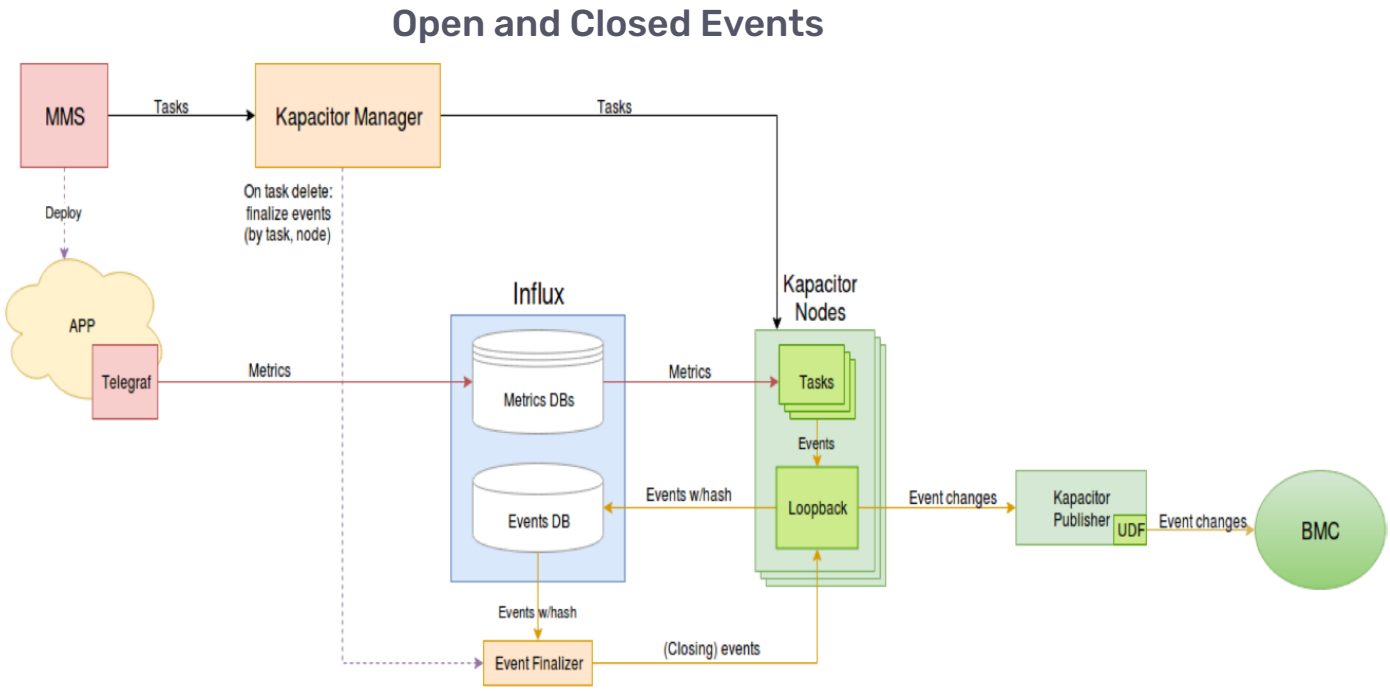
1. **Task management API** - Can install new tasks or retire previous task associated with the previous deployment
2. **Kapacitor nodes management API** - Can add a pair of nodes to the Kapacitor pool or remove them from the pool, and do that independently of the load of the system.
3. **Jobs API** - Assigns each job (any request viewed as potentially taking over five seconds) the job ID, and this call will be asynchronous. One type of job is potentially the rebalancing. Adding the new Kapacitor instances into the pool and therefore rebalancing the system is a dynamic job which may take quite some time. As a result, for the rebalancing, they get the task ID and then pull until the task fully implements.

There are quite a few queue workers able to pick up the different tasks from the queue and execute them so the queue is persistent. RingCentral is planning to run Kapacitor manager for each of their locations. Initially, their plan was to create the typical system: telemetry written into InfluxDB, and on top of the telemetry, Kapacitor for generating alerts.

CEP problems

Unfortunately, RingCentral's OCP system is a single-core processing system and has very low performance and very bad scalability (handling only up to 25 events per second). This presented another problem and required that they build, around their Kapacitor pool, a mechanism for detecting the event itself (thereby the change in the event status). They also needed to mollify the load on the event processor and therefore decided to keep the status of events inside InfluxDB.

As a result, they decided to deploy, inside their InfluxDB cluster, two databases: the metrics DB where all the metrics are collected, and the events DB. Their Kapacitor nodes, as a result, split into the two categories.



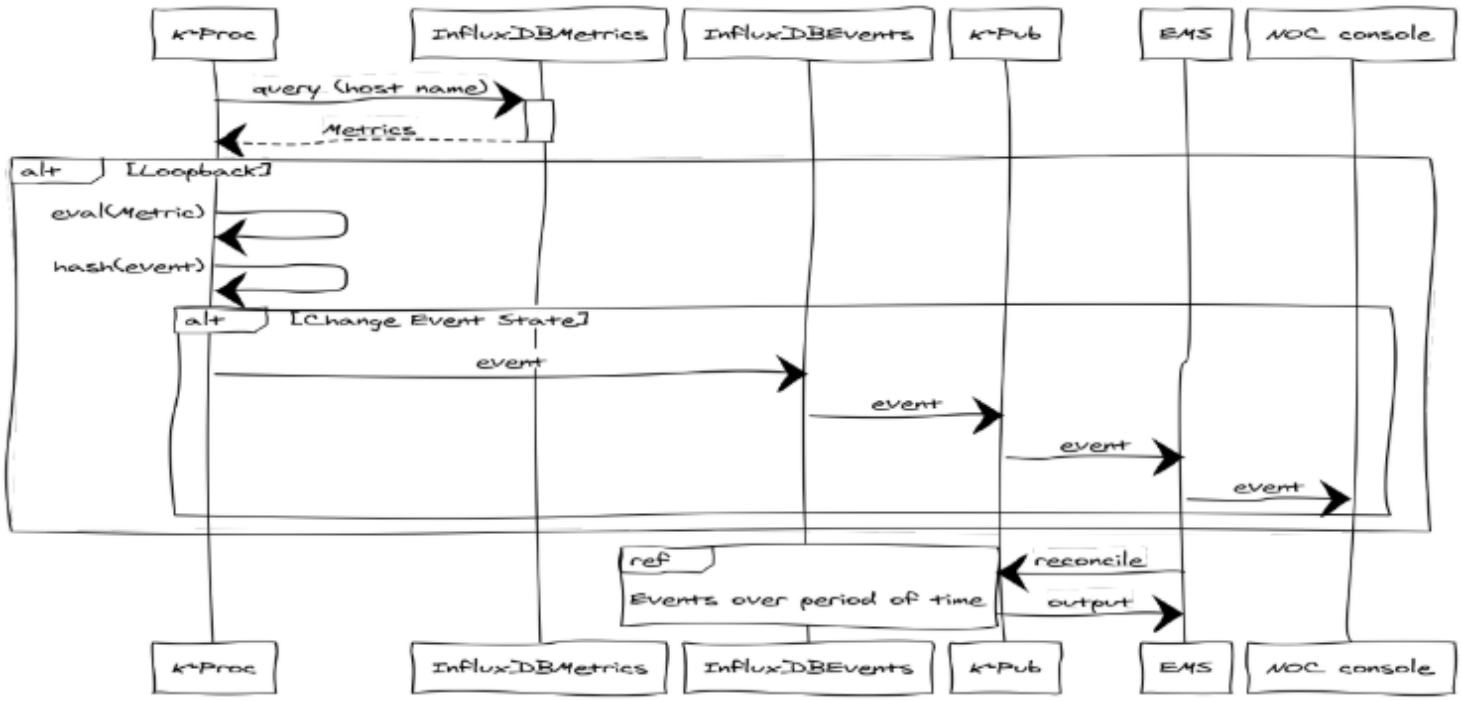
Detecting and generating the change event

The Kapacitor node, in case of a new change event, writes the data into the events DB inside InfluxDB and is able to detect if the change in the event status happened just recently. Then, only changes will be pushed by Kapacitor publisher which is implemented in the user-defined functions (UDF). It will be pushed further into their CEP system and then will be visible on their NOC Console.

In this creative throttling solution, which takes advantage of TICK Stack's versatility, InfluxDB and Kapacitor regulate the event flow to not choke the CEP system and provide a safenet for event recording, since their system can't guarantee delivery.

Below is a very simplified diagram of the event sequence that occurs when RingCentral's K-Plots (Kapacitor processors) make a query against InfluxDB to pick up the metrics. Then, in case the processor detects the change in this state, it will populate this change in the InfluxDB events database and will also push this event into the publisher.

Kapacitor and Event State Change Detection



Yet one pitfall with this approach was their inability to guarantee delivery of all events. So eventually, once in a while, their complex event system came back to them for the reconciliation. For that, they use InfluxDB Events database to grab the old events and ensure they didn't lose anything in the process. Their reconciliation steps are configurable and span roughly five minutes.

What's next for RingCentral?

RingCentral's future plans include:

- **Sandboxing and routing** - Adding a special Kapacitor router which will be able, depending on the metrics, to push metrics for the new tasks (which reside in the specially allocated Kapacitors) to ensure that only new tasks end up there, and after a specific graduation period, they will be able to move it and rebalance the rest of the pool.

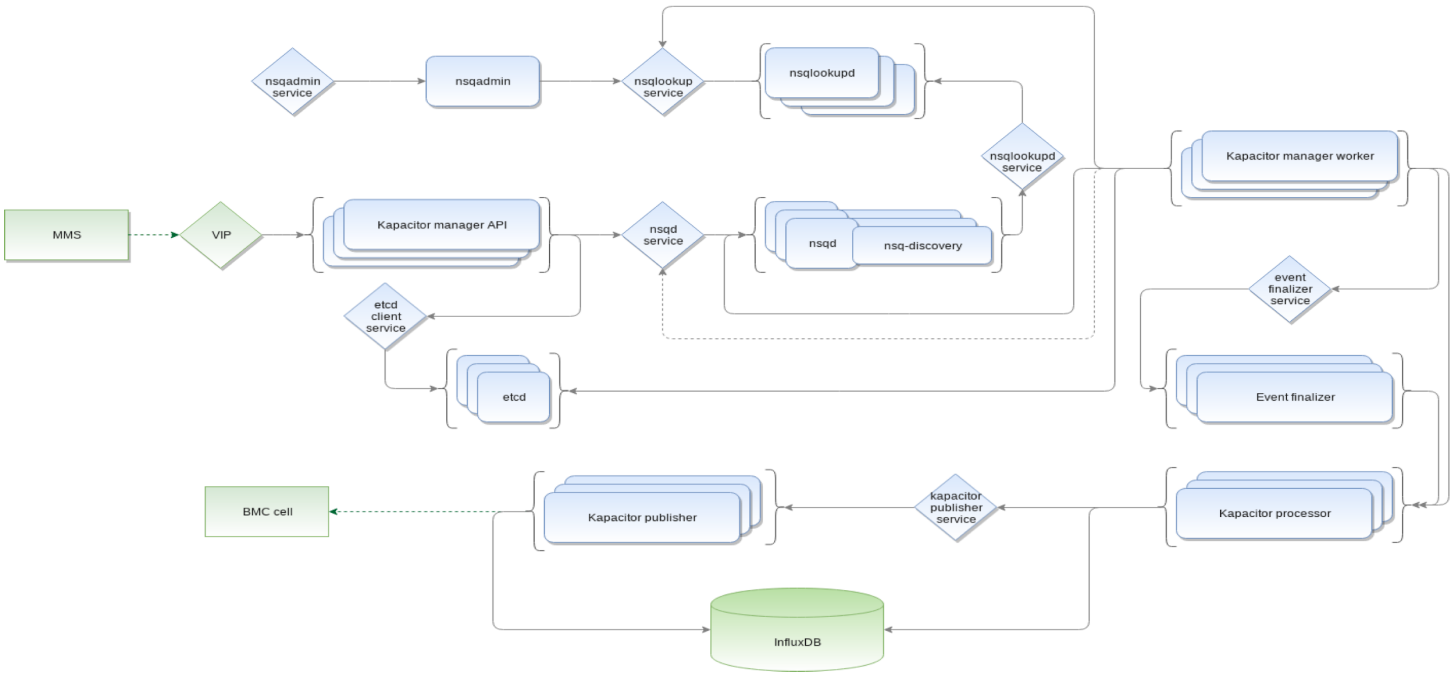
- **Smart rebalancing** - Balancing is currently based on the number of tasks without taking into account real task utilization. Because utilization is very dynamic, smart rebalancing is not an easy challenge and requires some history of utilization of particular tasks.
- **Open source** - They plan to go open source as soon as their legal department completes its review.
- **Align with TICK Stack upcoming releases** - They plan to figure out how the solution will align with InfluxData's new scripting language (Flux) and if it is will be the mechanism through which they will be able to replace Kapacitor.
- **UI** - They need to add some UI for the management to see what they are doing.

Results

> *"Our scalability model matches the ability of Kubernetes: we want to run all of our Kapacitor pools and deploy them inside Kubernetes to ensure we have enough processing and scalability on our Kubernetes cluster just for that purpose."*

Since migrating to InfluxDB, RingCentral's monitoring system has come a long way. Their scalability model needs to match the dynamic nature of their application environment, adopting solutions that ride well along their growth requirements. InfluxData TICK Stack components can be combined with other modern systems, like Kubernetes. In RingCentral's monitoring and alerting architecture, Kubernetes is envisioned to provide deployment flexibility and automatic scalability to their Kapacitor pools as a next step, as depicted in the diagram below.

Kubernetes: Deployment Media for Kapacitor Manager and Kapacitor's Nodes



Choosing InfluxDB Enterprise has freed RingCentral's monitoring infrastructure to keep pace with its business growth since InfluxDB is a high-performance data store common for DevOps monitoring use cases and written specifically to best handle metrics and events in very high granularity and volume. Allows for high throughput ingest, compression and real-time querying of that same data.

This migration, in combination with deploying Telegraf for metrics collection and Kapacitor for alerting, enabled their small-size monitoring team to build the in-house Kapacitor manager and achieve an automated real-time alerting system for their complex operations. This means less time-consuming monitoring activities, while coping with development and growth needs, and furthermore, freeing time and resources to develop knowledge and expertise in workflow automation and process streamlining.

About InfluxData

InfluxData is the creator of InfluxDB, the leading time series platform. We empower developers and organizations, such as Cisco, IBM, Lego, Siemens, and Tesla, to build transformative IoT, analytics and monitoring applications. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by sensors, applications and computer infrastructure. Easy to start and

scale, InfluxDB gives developers time to focus on the features and functionalities that give their apps a competitive edge. InfluxData is headquartered in San Francisco, with a workforce distributed throughout the U.S. and across Europe. For more information, visit influxdata.com and follow us @InfluxDB.

Try InfluxDB

Get InfluxDB