influxdata®

AN INFLUXDATA CASE STUDY

How GO-JEK Uses InfluxDB Downsampling to Avoid Burning the Midnight Oil

Anugrah S.
Product Engineer, GO-JEK
Aishwarya Kaneri
Product Engineer, GO-JEK

gojek

SEPTEMBER 2018

Company in brief

Established in 2010 as a motorcycle ride-hailing phone service, GO-JEK has evolved into an on-demand mobile platform and a cutting-edge app, providing a wide range of services that includes transportation, logistics, mobile payments, food delivery, and many other on-demand services.

GO-JEK is an Indonesian technology company with a social mission to improve the welfare and livelihoods of workers in various informal sectors in Indonesia. GO-JEK champions 3 essential values: speed, innovation and social impact.

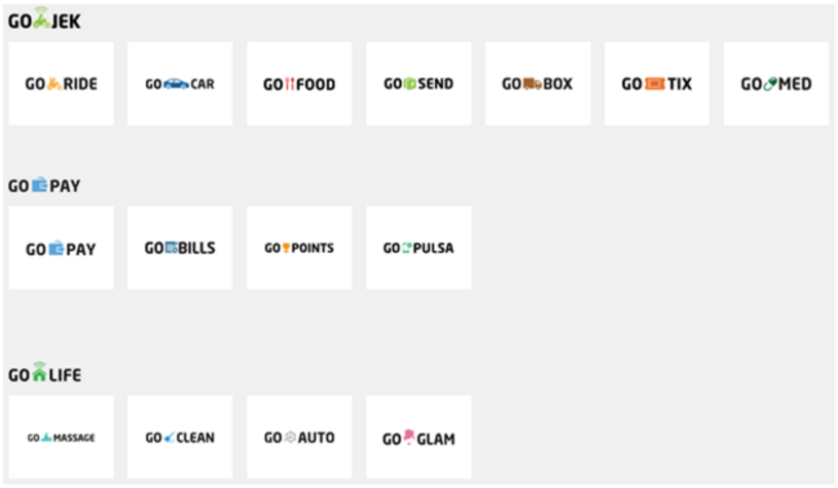
GO-JEK drivers say that since joining the company as partners, they have seen their income increase and reached more customers through the GO-JEK app. They also have access to health and accident coverage, financial services and insurance, as well as affordable automatic payments and many other benefits.

GO-JEK now operates in 50 cities across Indonesia with recent expansion into Vietnam, Singapore, and Thailand.

Case overview

GO-JEK uses InfluxDB for storing and collecting business, application, and system metrics. These metrics are used for monitoring and alerting — gathering 55,153 points per second during peak times, all written into an InfluxDB instance. With such a heavy load, GO-JEK faced the issue of high memory and disk space utilization — and instead of scaling the InfluxDB cluster horizontally — the IT team solved the disk space problem by downsampling their metrics data in InfluxDB.

They used InfluxDB and Grafana to build their monitoring solution — a solution that saved them from downtimes, rising machine costs and countless pages buzzing at night forcing them to burn the midnight oil to address performance issues. They automated this solution using Chef and Terraform for all the InfluxDB and Grafana instances.



One super app providing multiple services supported by InfluxDB

> *“As more and more series data was coming in, we started having a lot of issues with the memory and disk usage…We monitor our own monitoring architecture as well and were getting a lot of PagerDuty alerts during the night, and yes, burning of the midnight oil started for us.”*
>
> ***Anugrah S.,*** *product engineer, GO-JEK*

The business problem

GO-JEK is a fast-moving organization requiring very fast, data-driven decision-making. Since they have multiple services under one app, with some services having associated microservices, they needed to know, in real time, of any performance issues impacting their smartphone app or its underlying systems. Handling such issues was crucial to maintain GO-JEK’s three pillars:

1. Speed - providing fast service and continually learning and growing from experience
2. Innovation - continually offering new technology to make users’ life easier
3. Social impact - creating as much positive social impact as possible for Indonesians

Given the above pillars, and how crucial it was to maintain smooth performance, they needed to understand new feature impact and usage so they could be more efficient at feature implementation and improvement. For that purpose, they used the InfluxDB Ruby client, selecting it because it was easy to use:

1. GO-JEK provided the developers with easy-to-use dashboards with Grafana.
2. They allowed developers to add their own business metrics - without setting any storage limits.
3. Developers didn't need to manage their own instance of InfluxDB.

Giving their developers access to InfluxDB for storing their metrics was so wildly successful that too many metrics were being ingested by InfluxDB, causing issues. The option they were faced with is to limit the amount of data that their developers stored in InfluxDB. But their developers had already felt empowered by storing and visualizing their metrics, and could not be deprived of that. Since there was no turning back at this point, they needed to find a way to manage the data to ensure continued developer empowerment.

The technical problem

Originally, GO-JEK had decided to use InfluxDB to store their system and application metrics. But as they deployed more enhancements like dashboarding with Grafana, and alerting capabilities, on top of InfluxDB, their teams started using it more as well as started adding business metrics as well.

The unexpected popularity of InfluxDB led to an unexpectedly massive amount of stored metrics, which led to the following technical challenges:

- **Disk space and memory consumption** issues, which led them to consider giving their developers separate instances of InfluxDB. This was not feasible as it would become a logistical nightmare.
- A **massive volume of time series data**, which led to consider limiting the number of series, but this also presented a logistics problem, and they realized it would be easier to let their developers add the metrics they needed without such limitation.

- Issues with **horizontal and vertical scaling**

Due to these technical issues, GO-JEK was getting alerts about performance issues throughout the night as the disk space was never sufficient for all the business metrics. So whenever they got a pager, they used to increase the disk size, resize the disk, or have to increase the memory size as well. Turning off the alerts was not possible for GO-JEK since decision-making depended on them. Many of their teams depend on InfluxDB for business metrics, and their alerting stack is built on top of that, so if one service goes down and they don't get alerts, that becomes an issue.

Problems encountered with the current architecture

The GO-JEK application is broken into multiple services. Each of these services faced the issue of high memory usage. To manage this, each service collects its own share of system metrics (CPU usage, process checkers, disk I/O, network I/O, bytes out bytes in) and stores that into InfluxDB for analysis. In addition, as the teams push out new features, they collect specific business metrics about the feature (such as who is using it and what errors they faced) to gauge its success.

The GO-JEK team made sure to “monitor their monitor” and quickly saw that with a massive and fast-growing volume of data, collecting and storing metrics into InfluxDB became an issue. This resulted in a number of short-term fixes such as limiting the number of series and measurements ingested into InfluxDB, the constant need for increasing disk and memory size, and even creating separate InfluxDB instances per team. And finally, trying to scale both horizontally and vertically. Unfortunately, they soon realized these short-term fixes did not solve the problem.

Yet after doing further reading about the type of problem at hand, they realized that the main solution was data downsampling. Downsampling enabled them, since they have granular metrics, to remove details over a specified period of time. And to make downsampling work for the large number of InfluxDB instances they had, they also automated the downsampling process, as discussed further below.

The solution

> *“We had used Grafana, Kapacitor, and InfluxDB, but after we started facing the issues of high memory and disk utilization, that's when we explored the solution of data downsampling.”*

Aishwarya Kaneri, *product engineer, GO-JEK*

Why InfluxDB?

GO-JEK had originally decided to use InfluxDB for their time series data, mainly system and application metrics and started using it for other business metrics they collected with the Ruby client as well. As a time series database, it was simple for GO-JEK to add these new metrics and their developers loved the added insight they gained.

As a scalable time series database with high write and query throughput, InfluxDB could handle a massive volume of time series metrics and provided the flexibility to add ad hoc business metrics. They were able to manage the data ingested and stored in InfluxDB once they became familiar with InfluxDB's built-in functionality for data downsampling, data retention through flexible retention policies, and Continuous Queries.

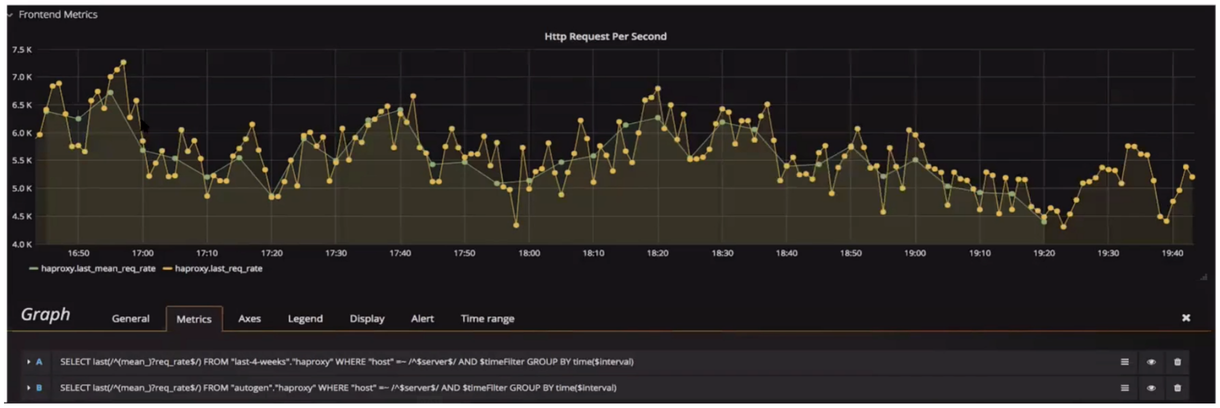
Go-JEK uses InfluxDB to:

- Store application and infrastructure metrics
- Dashboard with Grafana
- Downsample the data to keep storage under control
- Use the Ruby client
- Use Terraform

Data downsampling as the solution

The waveform below shows the actual collection of metrics. The yellow points show how many requests a haproxy is getting while the green points show the downsampled data.

Procedure for Data Downsampling



```
A  SELECT last(/^(mean_)?req_rate$/) FROM "last-4-weeks"."haproxy" WHERE "host" =~ /^$server$/ AND $timeFilter GROUP BY time($interval)
B  SELECT last(/^(mean_)?req_rate$/) FROM "autogen"."haproxy" WHERE "host" =~ /^$server$/ AND $timeFilter GROUP BY time($interval)
```

In the image above, from 16:50 to 17:00 — a 10-minute interval — there are close to 10 metric points via collector, each representing a different end of the spectrum: one is 7.2k requests, and the other is 7k requests. Between 16:50 and 17:00, all of these metrics could be represented by the mean of the metrics collected within the time period of 10 minutes, without losing much information. So downsampling is the process of changing the waveform (or wave structure) without losing much information in the process. This drops the number of requests collected here from 10 to two. Data is aggregated at five-minute intervals and then represented by calculating the mean.

Data downsampling procedure

The data downsampling procedure involves two steps:

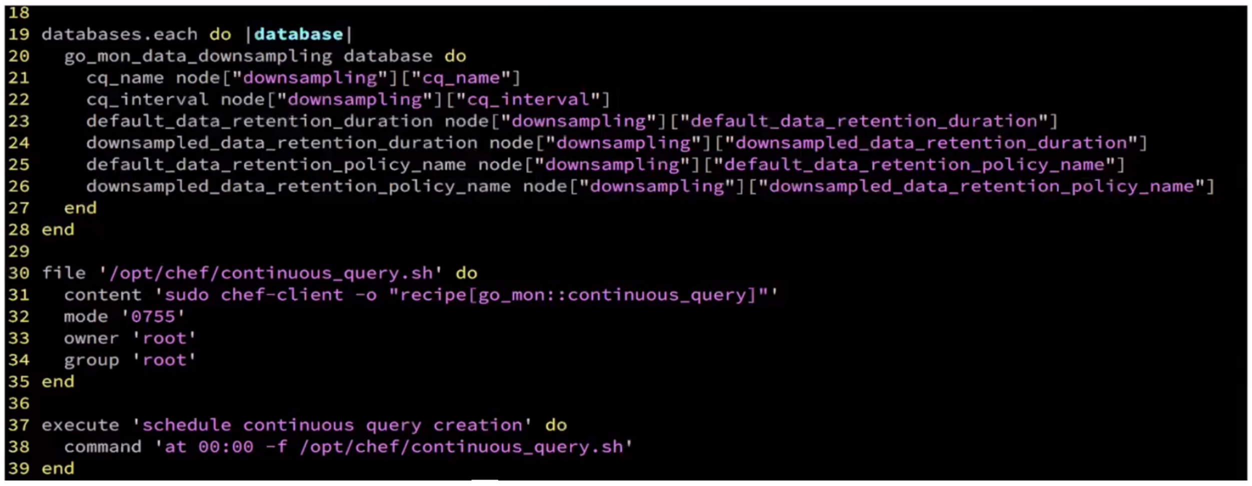
1. **Setting a retention policy:** A retention policy of one month is set for their normal data, collected at the rate of one data point per 10 seconds. They wanted to reduce that retention policy to two weeks, but the issue was that the teams wanted to visualize the normal data within a timespan of at least one month. So they downsample this data and keep the retention policy of the downsampled data of one month but reduce the normal data's retention policy to two weeks, so that the teams can view the normal data at the rate of 10 seconds, as well as get an idea of how the data looks like for one month by visualizing the downsampled data.
2. **Running a Continuous Query:** A periodic query which runs inside InfluxDB, do the data downsampling, gather the results and insert them into the downsampled retention policy measurement.

Data downsampling automation

Data downsampling in each InfluxDB instance of every GO-JEK team could not be done manually due to the large number of instances. A solution of their scale required downsampling to be automated, so GO-JEK used Chef Cookbooks for that purpose:

1. First, they wrote a **data downsampling recipe** for which they use the data downsampling resource. They reduced the retention policy of the normal data (called autogen) from one month to two weeks. They added a new retention policy, which was to be attached with the downsampled data, and this was for four weeks.
2. Second, because they have two InfluxDB instances running for every team, they want to start the data downsampling at the same time, so that it remains in sync in both the DBs. They created a **Continuous Query recipe** for that and scheduled it to run at midnight, using a Continuous Query script.

The Continuous Query Recipe



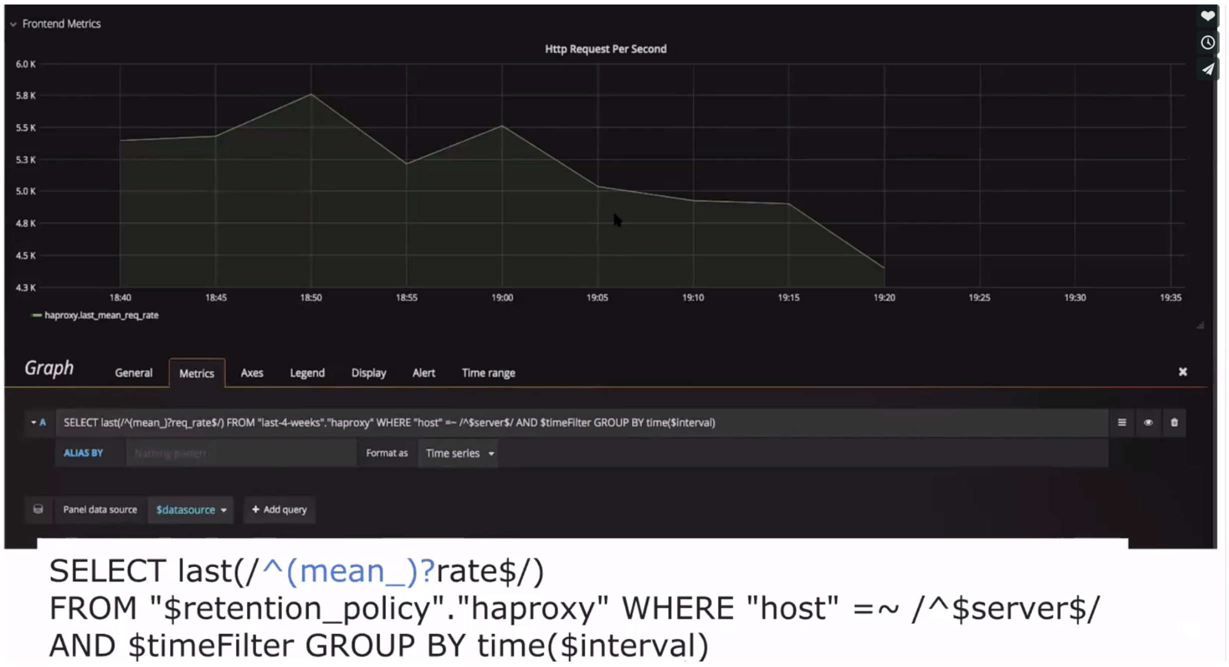
```
databases.each do |database|
  go_mon_data_downsampling database do
    cq_name node["downsampling"]["cq_name"]
    cq_interval node["downsampling"]["cq_interval"]
    default_data_retention_duration node["downsampling"]["default_data_retention_duration"]
    downsampled_data_retention_duration node["downsampling"]["downsampled_data_retention_duration"]
    default_data_retention_policy_name node["downsampling"]["default_data_retention_policy_name"]
    downsampled_data_retention_policy_name node["downsampling"]["downsampled_data_retention_policy_name"]
  end
end

file '/opt/chef/continuous_query.sh' do
  content 'sudo chef-client -o "recipe[go_mon::continuous_query]"'
  mode '0755'
  owner 'root'
  group 'root'
end

execute 'schedule continuous query creation' do
  command 'at 00:00 -f /opt/chef/continuous_query.sh'
end
```

In the data downsampling resource, they first create the retention policy (autogen) for which they use the InfluxDB Ruby library. The second retention policy is for the downsampled data (the Continuous Query recipe). In this query, they aggregate the data by using mean:

- Data is **grouped by five-minute intervals**, so the result of this query is dumped into the same database, but based on the downsampled data retention policy.
- All the **measurement names remain the same** so that after downsampling, for a given measurement, a new retention policy is created and the downsampled data is dumped in that

policy. The only change is that the column names of that measurement get prepended by `mean_`.

Data Downsampling Recipe



```
SELECT last(/^(mean_)?req_rate$/) FROM "last-4-weeks"."haproxy" WHERE "host" =~ /^$server$/ AND $timeFilter GROUP BY time($interval)
```

```
SELECT last(/^(mean_)?rate$/)
FROM "$retention_policy"."haproxy" WHERE "host" =~ /^$server$/
AND $timeFilter GROUP BY time($interval)
```

Issues faced during data downsampling

During data downsampling, GO-JEK faced three issues:

1. The **Continuous Query started increasing the load on InfluxDB** because this query is periodic and will run after the given time interval of five minutes, causing InfluxDB to become quite slow. This issue was fixed by decreasing this query's frequency.
2. GO-JEK's **database names were hyphenated** database names, and changing that hyphen for all the InfluxDB instances would change their legacy infrastructure. This hyphen wasn't supported in the InfluxDB Ruby client, so they added a PR for this to support hyphenated database names.
3. The **downsampled data of the measurement had "mean" prepended** to the field names. All the Grafana dashboards have the column name. But for downsampled data to be visualized on the Grafana dashboard, the column name got changed and the mean got prepended to the column name, so they couldn't use that dashboard. To avoid creating a new dashboard for

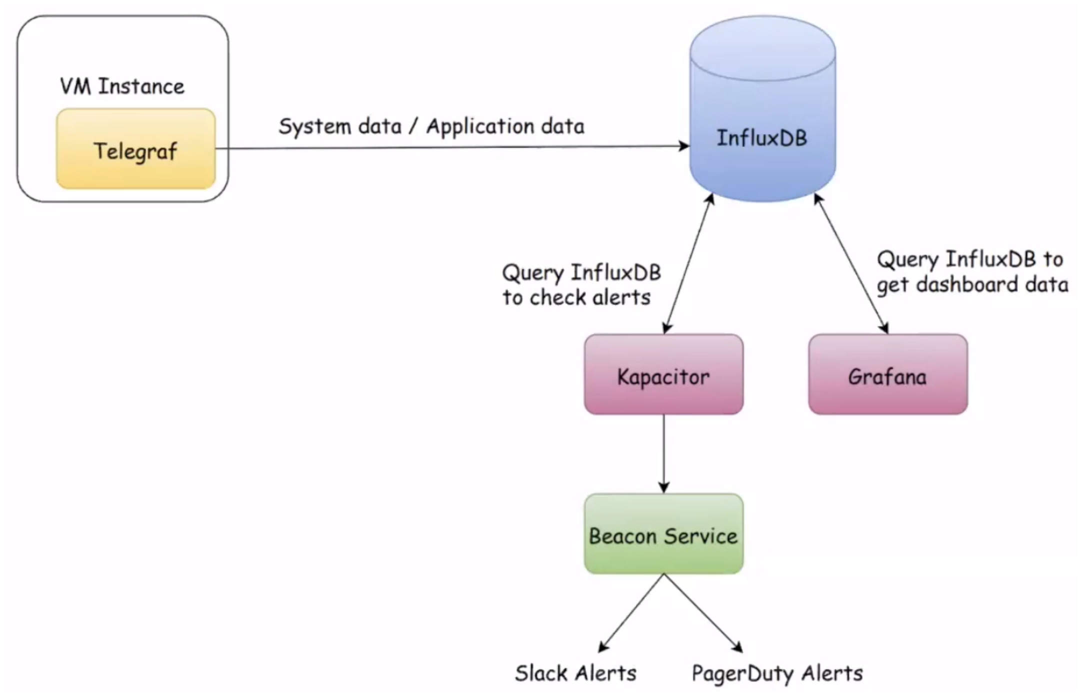
downsampled data, they chose the regex approach, enabling the team to select whether they want to view the downsampled or normal data.

The method used in Grafana to enable the choice of retention policy is transparent to the end user. GO-JEK added, in Grafana, templates on top of the dashboard. The same graph can be used to visualize the HTTP Request for different servers. Similarly, they created a template for retention policy and provided two views within the same dashboard: one for the autogen policy and one for the one-month policy.

Technical architecture

> *“We are using two InfluxDBs for reliability, and every team has been given this entire setup architecture of Load Balancer, the two relays, and two InfluxDBs. All the write queries are sent to this InfluxDB relay, and this relay will redirect them to the two InfluxDBs.”*
>
> ***Aishwarya Kaneri,*** *product engineer, GO-JEK*

Monitoring & Alerting Architecture

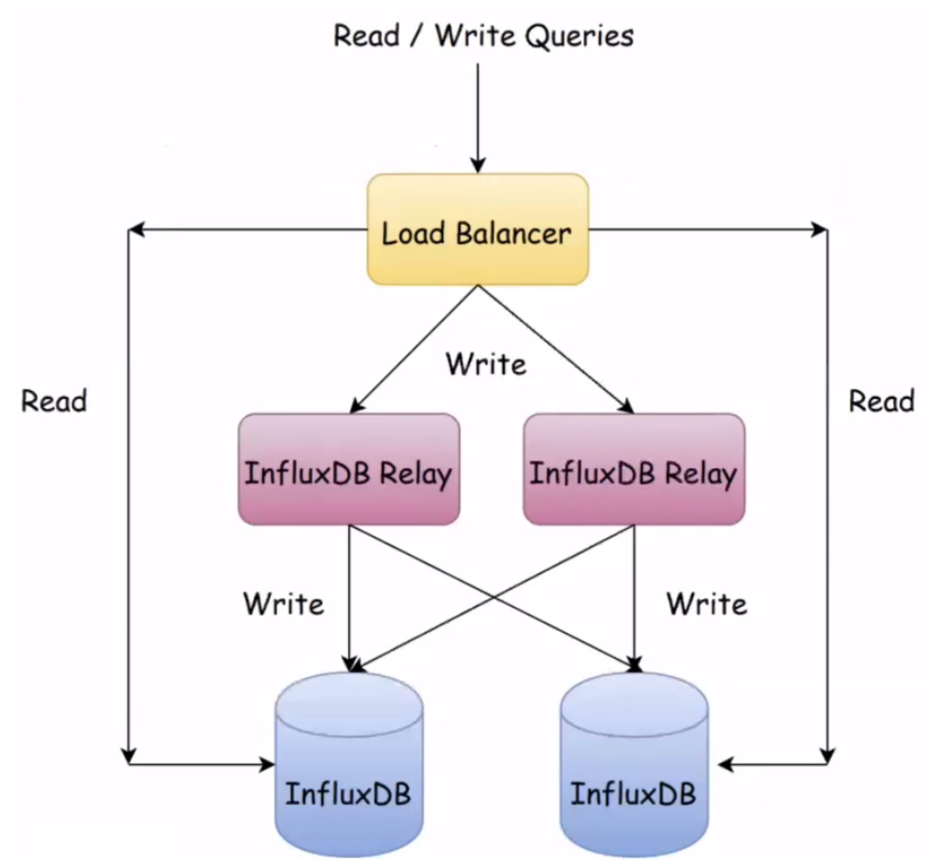

VM Instance
Telegraf
System data / Application data
InfluxDB
Query InfluxDB to check alerts
Query InfluxDB to get dashboard data
Kapacitor
Grafana
Beacon Service
Slack Alerts
PagerDuty Alerts

The GO-JEK monitoring and alerting architecture is as follows:

- **Telegraf** runs on all the VM instances and sends system application and business metrics to InfluxDB.
- **Kapacitor** checks InfluxDB for whether the alerts set are crossing thresholds and sends that to the Beacon Service, written in-house by GO-JEK.
- The **Beacon Service** decides which team will receive these alerts and sends them via Slack and PagerDuty.
- **Grafana** queries InfluxDB to get all the dashboard data.

All the read and write queries first hit the load balancer, which sends **all the write queries to InfluxDB relay** and **all the read queries directly to InfluxDB**.

- Two InfluxDB instances per team are used for reliability, and every team has been given this entire setup architecture of load balancer, two relays, and two InfluxDBs.
- All the write queries are sent to the InfluxDB relay, and the relay redirects them to the two InfluxDBs.



GO-JEK has written a Chef cookbook for their InfluxDB architecture with the two relays:

- They have a tool consisting of a command. They run it and provide the team name, and through that command, all the above architecture gets created.
- They use Terraform to automatically create the architecture, and when that command runs, one load balancer as well as two relays and the two InfluxDBs are created.
- Telegraf gets installed along with plugins being used with it, and then that setup is ready to use. Whenever a VM instance is created, depending on the team name, inside that VM instance, Telegraf is installed. The endpoint for that Telegraf depends on the team name.

What's next for GO-JEK?

When implementing data downsampling, GO-JEK had used InfluxDB v1.4. They then saw that InfluxDB v1.5 onwards had switched to the TSI model. In the TSI model, whenever new data arrives, it gets written into the disk but not in the in-memory structure of InfluxDB. So as the number of series and measurements increase in the disk, memory utilization remains low.

The model's drawback, as GO-JEK saw it, is that the dashboards would take longer to load because there is no in-memory cache present, and the disk has to be repeatedly queried for the data. To switch to the TSI model, they would have to make substantial changes and risk losing the data, but they are planning to adopt newer versions in the future.

In hindsight, they believe they should have started with the newest version of InfluxDB available at the time, and done a lot of research before going for hard fixes since such fixes are not the best approach to deal with the problem and never work in the long run.

Results

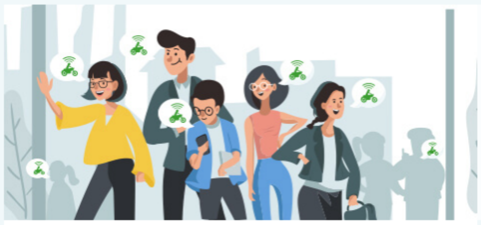
> *"We saw a significant amount of reduction in memory usage and disk usage. Automation helped us scale the solution for multiple InfluxDB clusters without dealing with a lot of errors, which is one of the plus points of having automation. And, yeah. No more sleepless nights."*
>
> ***Anugrah S.,*** *Product Engineer, GO-JEK*

Scalable technology to keep up with rapid growth

By using InfluxDB downsampling to solve disk and memory size problems, automating downsampling, and benefiting from InfluxDB's flexible retention policies and Continuous Query functionality, GO-JEK's infrastructure today can scale with its rapid growth.

- From founding until June 2018, GO-JEK app has been downloaded more than 96 million times.
- In the last 36 months, the startup's total order volume has grown by 6600x and diversified into 18 verticals.
- In 2017 alone, its 200+ engineers completed 100+ million monthly orders.
- GO-JEK's 200 engineers make software decisions that trickle down to affect the lives of about 260 million people in Indonesia.

Work that speaks for itself



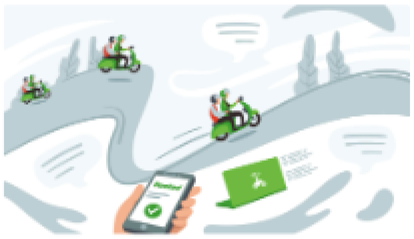
3 Million+
orders everyday

6600x
total order volume growth in 3 years

1 million+ drivers
More daily rides than India's largest ride-sharing service

18+ Products,
200+ engineers

GO-JEK Tech Facts



We process an average of 35 orders every second.

We process more than 350 million internal API calls per second.

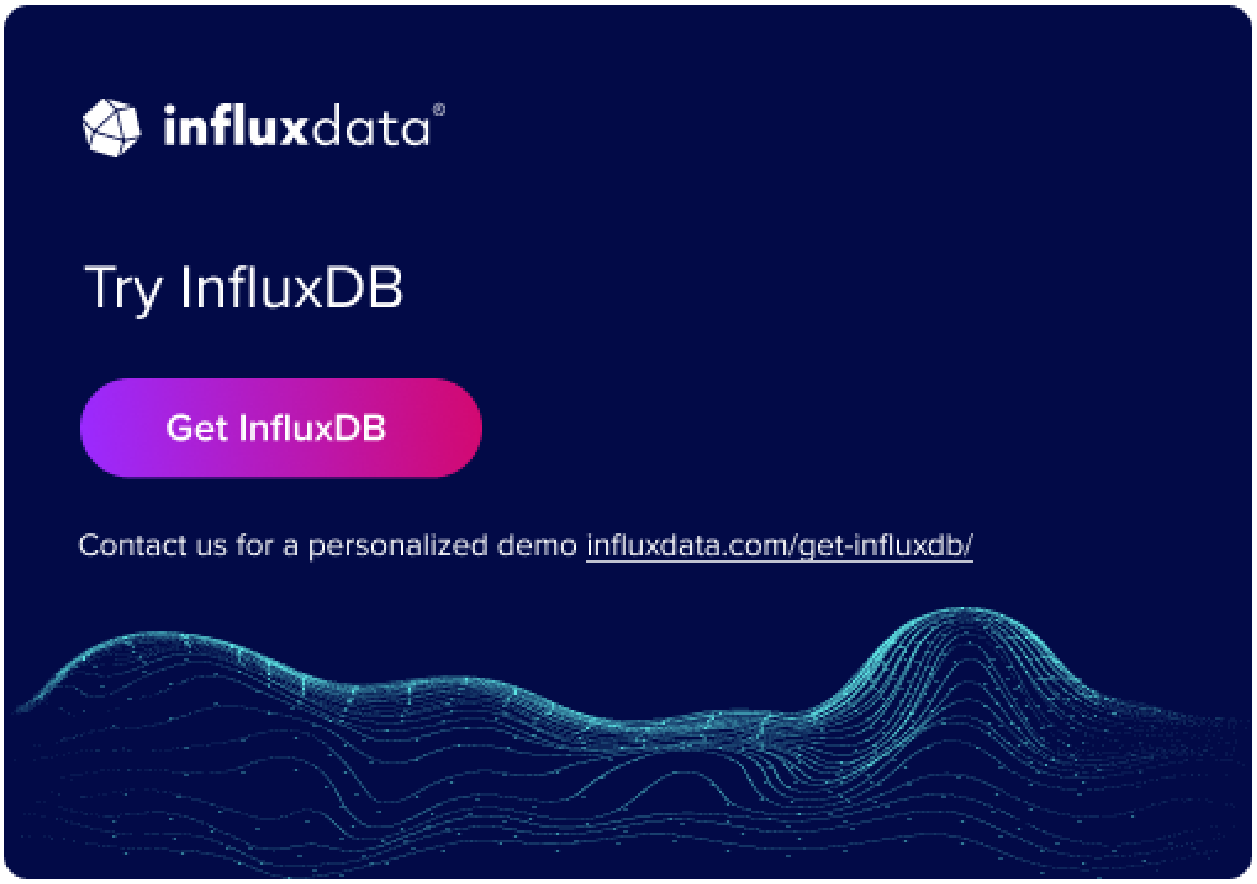
We run one of the largest JRuby, Java and Go clusters in Asia.

GO-JEK's impact in Indonesia

Using technology as a means of social impact, GO-JEK has created jobs, improved livelihoods, and helped nudge micro-entrepreneurs. It has grown to become the largest on-demand service provider in Indonesia that connects users to more than 1 million driver partners, more than 200,000 food vendors, and more than 30,000 other service providers. The new jobs GO-JEK has created address 15% of the country's employment, or about 1 million people in the 3 years since the app was launched.

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.


influxdata
Try InfluxDB
Get InfluxDB
Contact us for a personalized demo influxdata.com/get-influxdb/

548 Market St. PMB 77953, San Francisco, CA 94104