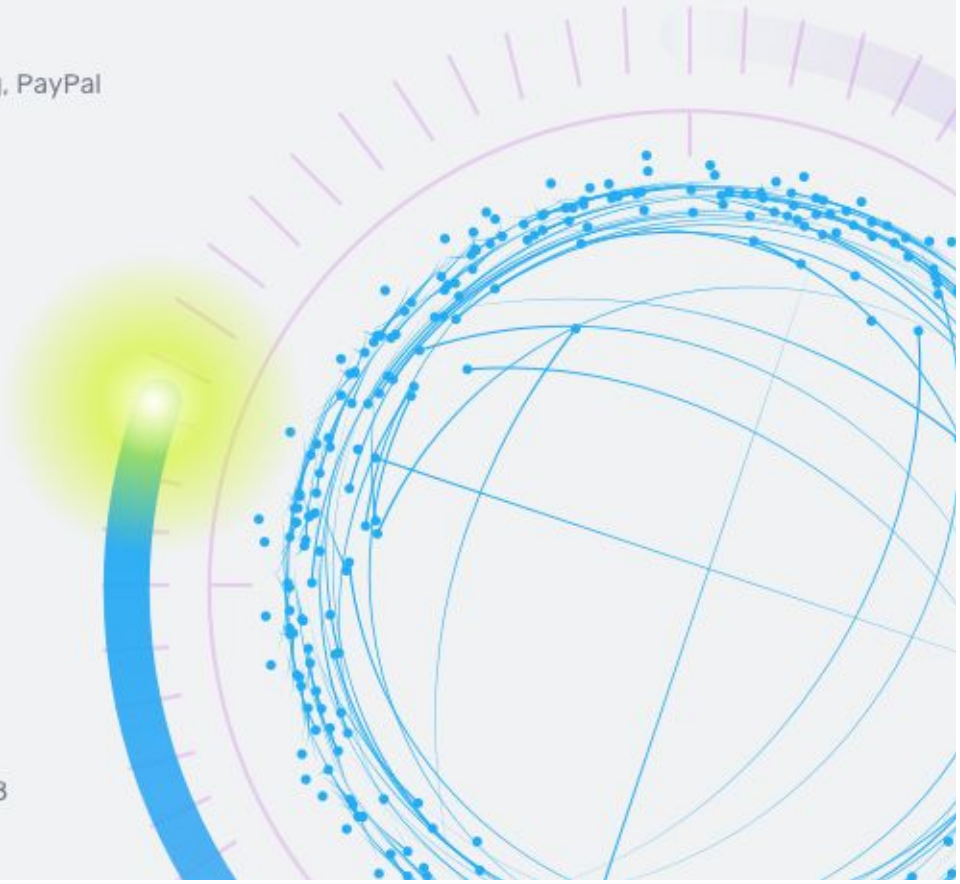
influxdata

# Best Practices for Scaling an InfluxDB Enterprise Cluster

AN INFLUXDATA CASE STUDY

Dennis Brazil

SRE Monitoring, PayPal



November 2018

# Company in brief

Fueled by a fundamental belief that having access to financial services creates opportunity, PayPal (NASDAQ: PYPL) is committed to democratizing financial services and empowering people and businesses to join and thrive in the global economy.

PayPal's open digital payments platform gives PayPal's 254 million active account holders the confidence to connect and transact in new and powerful ways, whether they are online, on a mobile device, in an app, or in person.

Through a combination of technological innovation and strategic partnerships, PayPal creates better ways to manage and move money, and offers choice and flexibility when sending payments, paying or getting paid. Available in more than 200 markets around the world, the PayPal platform, including Braintree, Venmo and Xoom, enables consumers and merchants to receive money in more than 100 currencies, withdraw funds in 56 currencies and hold balances in their PayPal accounts in 25 currencies.

# Case overview

PayPal needed to find a scalable end-to-end host monitoring solution to replace its old one. As the company was modernizing its infrastructure and transitioning many of its applications to a container-based architecture, the new monitoring solution needed to be designed to work with containers and to provide metrics collection, storage, alerting and visualization all at once since the team's preference was to select a single-vendor platform.

PayPal chose InfluxData's InfluxDB Enterprise and leveraged all components of InfluxData's open source core platform to build a solution using Telegraf aggregators, message queues, and publishers in order to control data payload size, manage message flow, and avoid single point of failure (SPOF).

Using the InfluxData platform, PayPal built a resilient monitoring solution that works at scale, and in the process, they derived several conclusions regarding scaling InfluxDB Enterprise clusters.

> *"We had a need for a new host monitoring solution to replace antiquated monitoring solutions and we needed the ability to monitor multiple Docker containers with a single agent... Following with some of the requirements...we realized we also needed a time series database backend for reporting history."*
>
> ***Dennis Brazil,*** *SRE monitoring*

# The business problem

PayPal, whose platform enables digital and mobile payments on behalf of consumers and merchants in more than 200 markets worldwide, was looking for a "Host Monitoring" solution to replace antiquated monitoring systems. The monitoring solution had to be scalable to keep pace with the company's infrastructure.

PayPal has nine data centers, with 30,000 instances each, and they all have their own clusters. The company was migrating all their old applications — some 20 years old and compiled in C++ — into containers and more modern operating systems, with many of their Docker hypervisors hosting 50 to 100 containers at once.

# The technical problem

Following were some of PayPal's technical requirements for its host monitoring solution:

- A reliable and extensible agent sitting on all systems to monitor basic OS system metrics such as CPU, Disk, Memory, third-party applications and databases
- Time series database backend for reporting history
- Ability to monitor multiple Docker containers with a single agent (critical to keep the agent's overhead down across the whole system)
- Smart alerting based on time series data

PayPal wanted to meet those requirements through one vendor to solve all of their host monitoring problems.

# The solution

> *"Time series data helps us make educated, data-driven decisions quickly. It is what keeps us in business. It drives the need for products like InfluxDB. If you can't measure something to get results, you can't possibly get better at it. Worse yet, you won't know what you should be focusing on."*

## Why InfluxDB Enterprise?

InfluxData's InfluxDB Enterprise (the Enterprise edition of InfluxDB) provided an end-to-end solution from one vendor as Paypal was seeking. Here's why they selected all components of the InfluxData platform (the TICK Stack):

- Telegraf provides an extensible plugin-based architecture for monitoring for all OS's, applications, and Docker containers
- InfluxDB provides a fast, scalable time series database.
- Chronograf has an intuitive data explorer and query builder.
- Kapacitor provides smart alerting capabilities.
- Each component had single binaries which provide simplicity in deployment.
- InfluxData's architecture allows for further scalability & customization.
- InfluxData provided helpful technical support.

Below is an overview of each of the components used in designing PayPal's new monitoring solution.

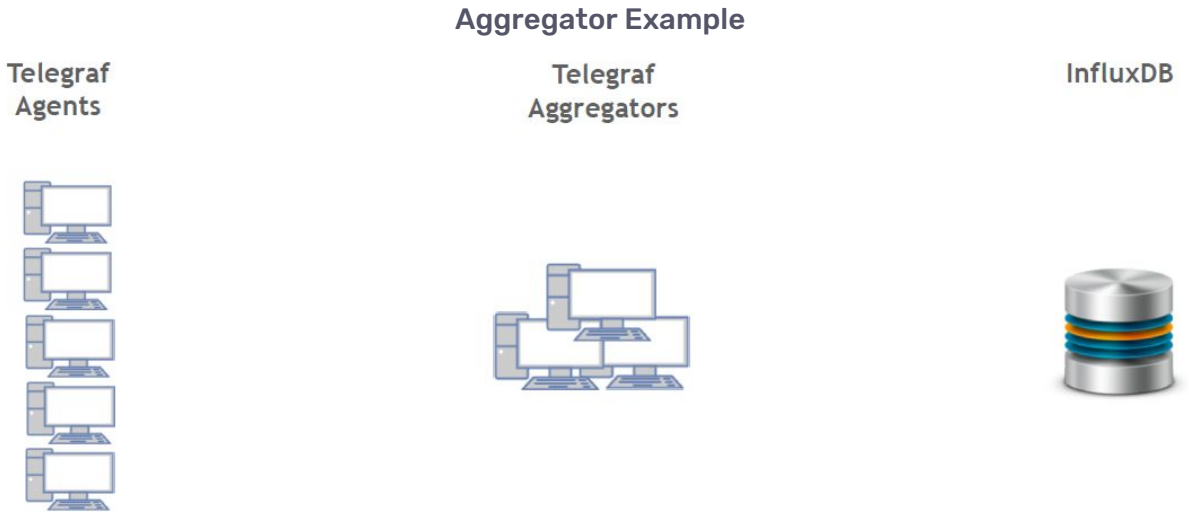
**Telegraf aggregators**

The Telegraf aggregators were chosen because they perform the following:

- Send larger payloads with fewer writes to InfluxDB having better overall performance on the cluster—they reduce the number of processes that need to happen in the database on the backend.
- Send to multiple outputs and formats — Telegraf can transform the line protocol format into JSON to be parsed by a custom consumer publisher that you might want to write yourself for your own environment.

- Filter, sanitize & process at scale quickly (tagpass, tagdrop, namedrop, namepass) — making it possible to change only the five nodes at the aggregator level, without having to change the 50,000 - 100,000 nodes in their infrastructure.

The diagram below shows all the Telegraf agents in PayPal's data center sending to the aggregators' Virtual IP (VIP) load balancer. The aggregators take all those smaller messages from all the agents and send one large payload directly to the database.




**Message queues**

Message queues were used because they:

- Store messages until final destination is available
- Provide a common platform for publishing to just about any other system (such as on to a relational database, MySQL, and machine learning systems for further processing/alerting)
- Use "fan-out" exchanges to multiple queues to replicate to other InfluxDB clusters or nodes
- Manage TTL (time-to-live) for messages, which helps avoid running out of disk space

**Publishers**

As publishers are the smart way to manage message flows, PayPal used them to:

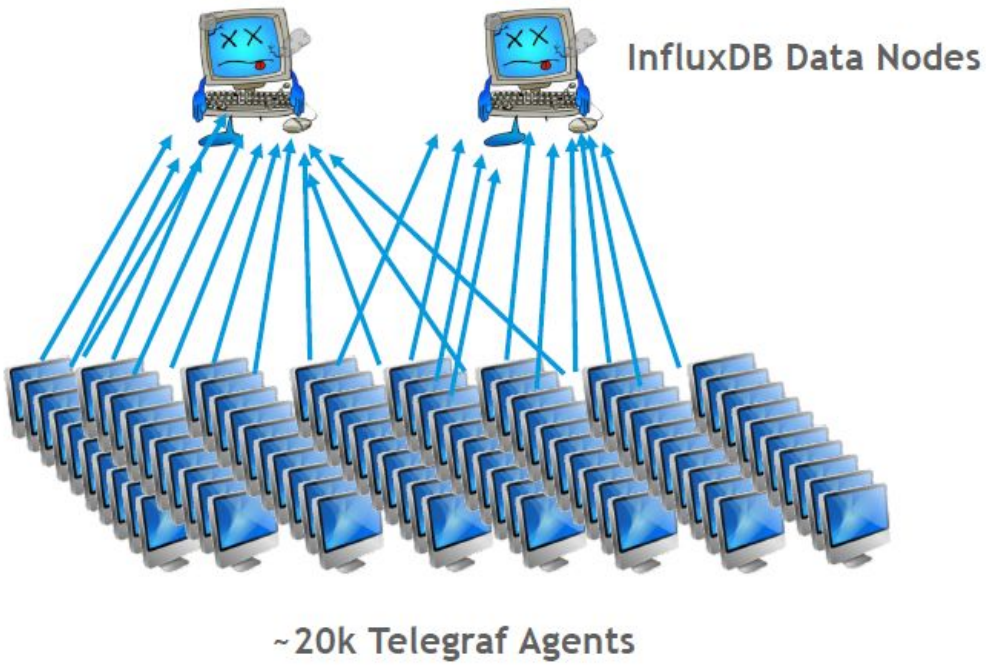
- Consume from message queues — publishers are named as such because in the final step in the flow they push messages and publish to some final destination.
- Publishers send heartbeats to ensure the database is ready to accept messages.
- Publishers are "smart" in that they only deliver messages while the database is up.

# The technical journey

PayPal's journey to scalability has not been straightforward. They had to go through multiple iterations to reach a "nirvana" that is currently working for them.

In Iteration 1, they used three meta nodes and two cluster nodes. They had 20k Telegraf agents to a single cluster. The data nodes, as shown below, were completely overwhelmed.

**Iteration 1**

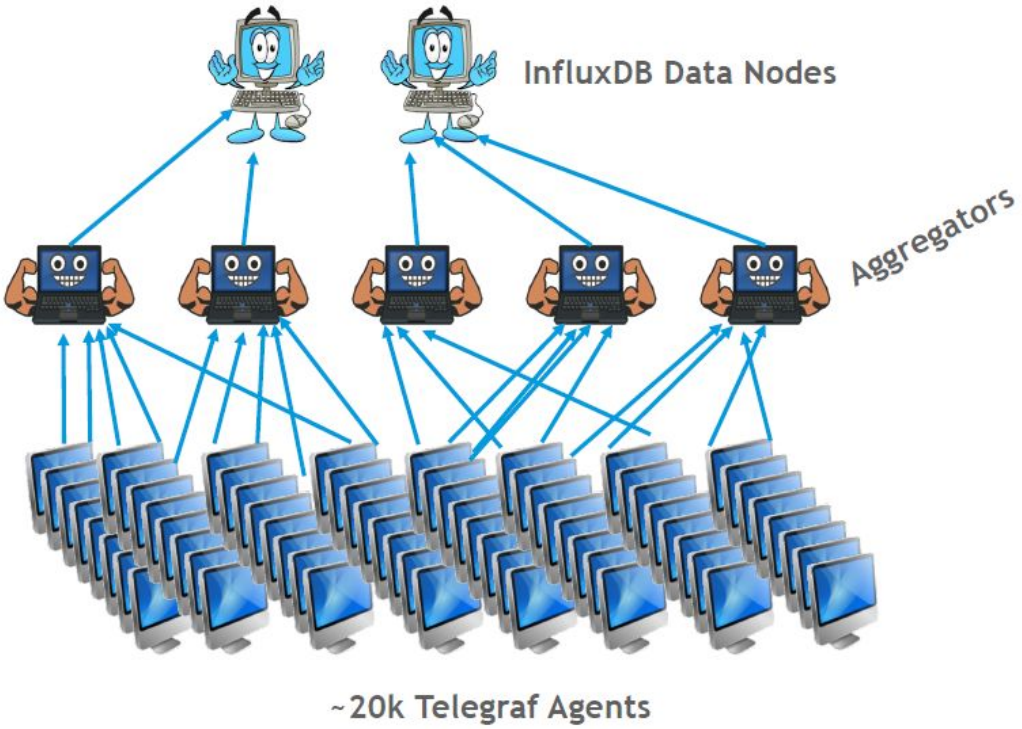



Iteration 1 didn't work well because there were too many writers with small payloads. Every single payload that was sent to the database creates a wall entry. So 30,000 wall entries per minute in a single node is lot to keep up with. Iteration 1 didn't work because it presented:

- No buffering/retention in case there is backpressure from the database. There is some buffering in Telegraf, but it's all in memory, and this means you're constrained to the memory on the system.
- Single point of failure (SPOF) condition as soon as one node fails. When you have two nodes, as soon as one node fails, you are immediately in a SPOF condition.

In Iteration 2, PayPal added Telegraf aggregators between Telegraf agents (in the data center) and data node cluster behind them. Aggregators helped reduce the number of writes by combining multiple tiny payloads into larger ones for fewer writes to the database, enabling the database to better keep up.

**Iteration 2**




The aggregators have the HTTP listener plugin setup emulating what a data node does. It listens on port 8086, so the Telegraf agents think that they are posting directly to the database, but instead, they're posting to the HTTP listener on the Telegraf aggregators. Telegraf aggregators run Telegraf, whose purpose is to ingest metrics and pump data into message queues.

Yet Iteration 2 presented the same two challenges as Iteration 1:

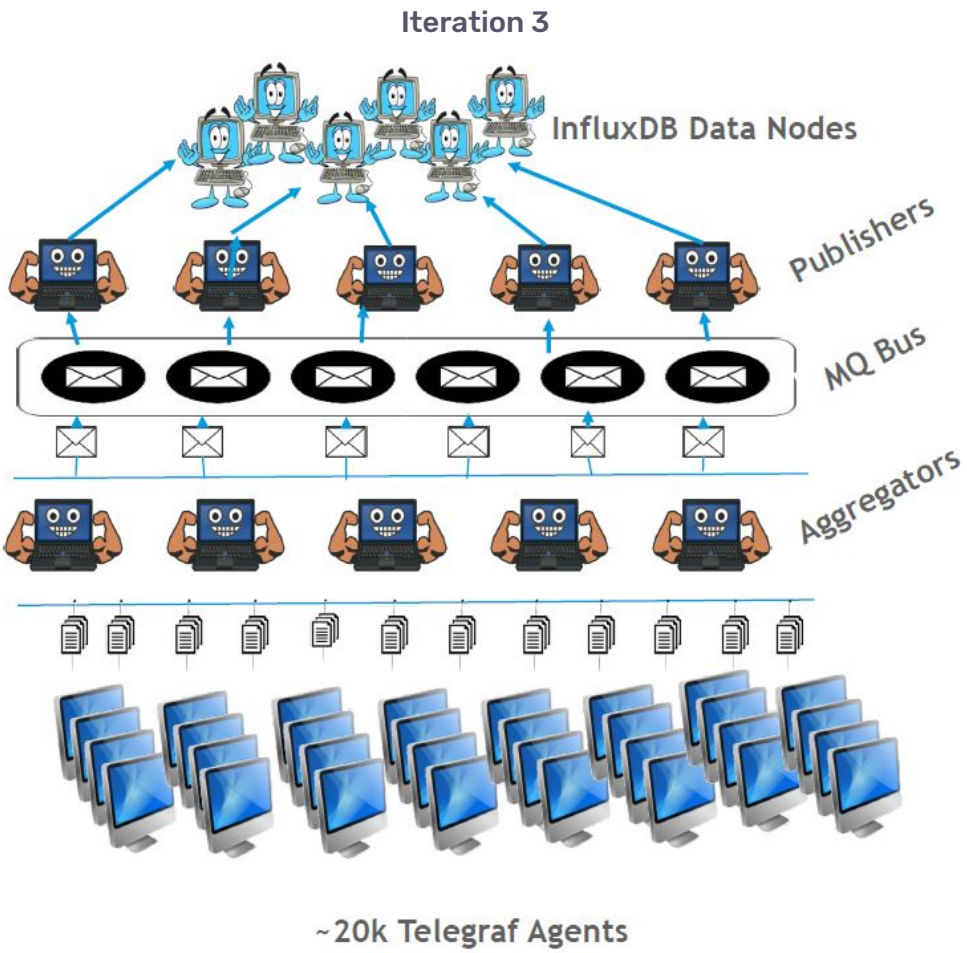
- No buffering/retention in case the database is unresponsive
- Single point of failure (SPOF) upon node failure

In Iteration 3, PayPal added the following:

- Message queues (MQ) for aggregators to send to
- "Publishers" to consume messages from MQ and post to InfluxDB (post the data from InfluxDB to the cluster). These are larger payloads, i.e., less number of writers. So instead of

20,000-40,000 writers to the database, they reduced the number to 5 doing all the writes on behalf of all the agents out in the data center.

- Additional data nodes to the cluster and a replication factor (RF) set to 3

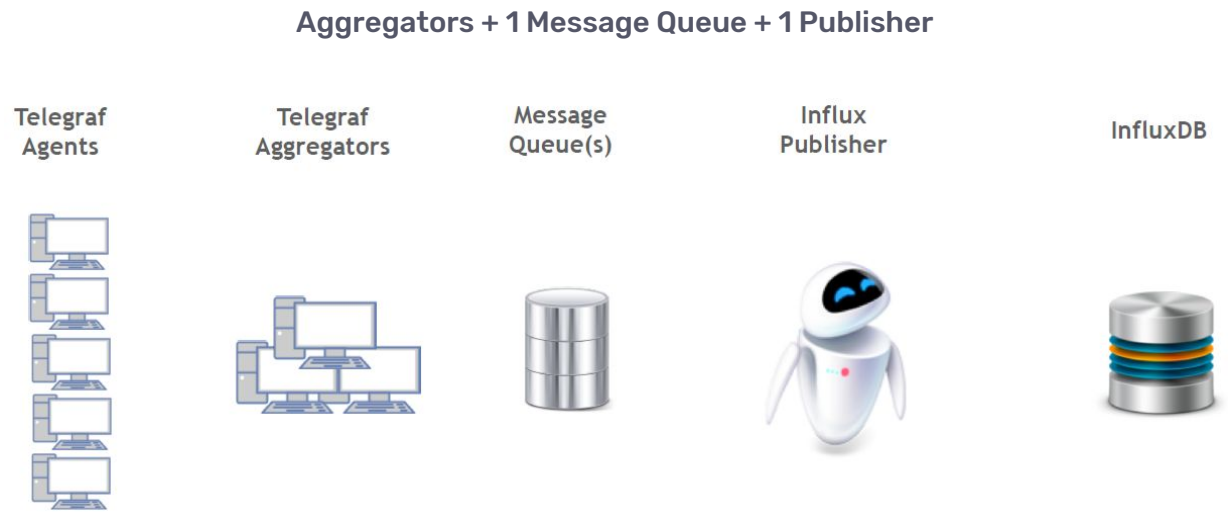



These changes helped because:

- MQ's prevent data loss when the database is unavailable (by retaining data until the consumers consume the data).
- Smart publishers watch for back-pressure and back-off until cluster is available.
- The new setup prevented immediate SPOF condition with RF of 3 with more data nodes (they don't consume messages or publish until the database is made available again).

# Technical architecture

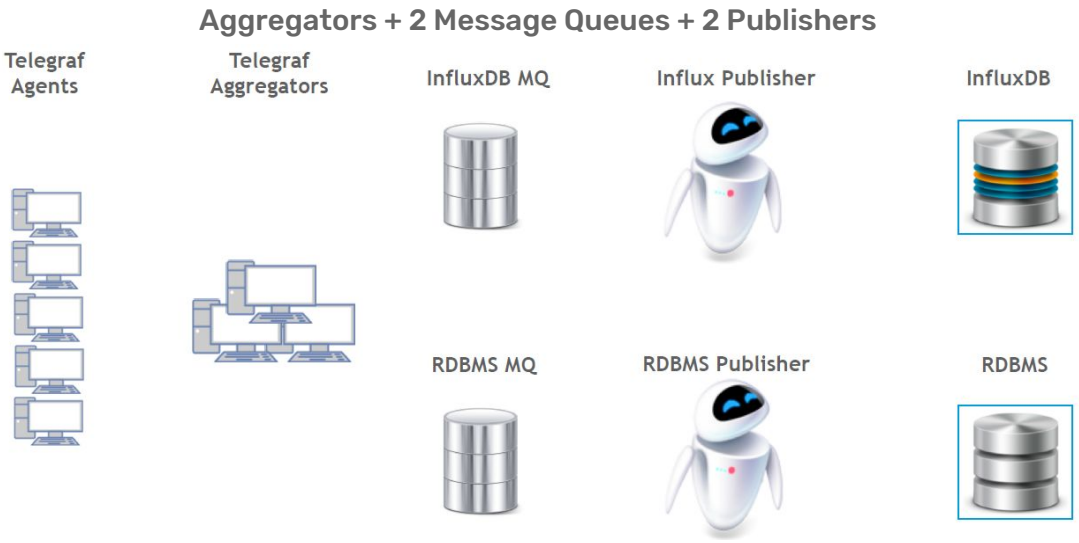
> *"The Telegraf aggregators actually take all these smaller payloads, and we basically forklift like 5,000 metrics at once, into one payload into the database, so that's one huge wall entry to commit to the database."*




The above is a simplified conceptual visual of the dataflow in PayPal's architecture to simply show what the design looks like when you bring all the components together:

- Telegraf agents send to Telegraf aggregators and then go to the message queue.
- An InfluxDB publisher consumes the messages from the message queue and then publishes to InfluxDB.

PayPal's actual architecture is more complex:




Dataflow and its management are as follows:

- In a given data center, they have five aggregators in a pool behind a virtual IP (VIP) - a load balancer which is in every availability zone, or data center. If the five aggregators cannot keep up, more aggregators are added.
- For aggregators, they use the same port as InfluxDB (8086) because they did not want to modify the default configuration for Telegraf.
- All the Telegraf agents publish to the Telegraf aggregators and then go to two different message queues: one destined for InfluxDB, and one destined for a relational database.
- One database will still be accepting messages, but when InfluxDB is not responding, the messages bounce back. Messages remain on the frontend and are still sent to the message queue. Nothing gets queued at the Telegraf agent in the data center. The RDBMS queue still publishes data.
- As soon as InfluxDB comes back online, InfluxDB publisher detects that and starts consuming messages again and publishing. That's how it manages the flow.

# What's next for PayPal?

PayPal plans to migrate to the new release of InfluxDB Enterprise forthcoming in 2019, which will retain all platform components but will package them a little differently.

# Results

> *"I cannot tell you how much we reach out to InfluxData Technical Support, which has been really, really, really helpful to help us out in time of need."*

Following their journey, which led to designing a new monitoring solution that met their needs, PayPal reached some conclusions and recommendations regarding scaling an InfluxDB Enterprise cluster. All these tips and best practices apply to InfluxDB Enterprise, but many also apply to the open source version of InfluxDB:

**1- Avoid being greedy:**

- Collect only the data you need
- Start with a smaller retention policy first
- Check which retention policy is best for you (it is easier to extend than reduce your retention policy because reducing retention policies does not clean up data from the former retention policy). So PayPal recommends to start small, grow slowly, and evaluate.

**2- Ensure all cluster nodes have the same characteristics:**

- Ensure all nodes in your cluster have the same machine characteristics such as RAM, Disk IO/IOPS, and Network IO.
- The above step is critical since a single slower node can impair the performance of the whole cluster.

In this regard, the graph below depicts a real-world scenario that PayPal witnessed. Here all of their cluster nodes show memory usage, hinted handoff queue, and heap size. One node is misbehaving and slowing down the rest of the cluster.

**Slow Node Example**


While the slower node is in the cluster, the hinted handoff was persistent. All of the nodes in the cluster were trying to send to that one slower node. Hinted handoff queue buildup occurred on the rest of the nodes in the cluster. Also, the slower node had a larger heap size, trying but unable to keep up with all the writes coming to it.

Once the decision was made to remove the slow data node from the cluster, the memory usage became slightly higher, but a bit more balanced, on the rest of the cluster. The hinted handoff eventually depleted to zero, across the board, just by removing that one node. Now, they have a much happier cluster as well as heap size balance on the remaining nodes as well.

**3- Where possible – divide & conquer**

- Maintain regional clusters and roll up alerting to a single cluster, or
- Maintain regional Telegraf receivers/aggregators and forward to a single cluster.

**4- Scale up! (Use beefier hardware)**

- More CPU power is important for write throughput: a lot of CPU power is required for compaction, compression, and writes.
- RAM and disk speed are important for query throughput for reading.
- Remember that if you can't measure it, you can't manage it. Everything consists of data. If you can't monitor it, then you won't have metrics and won't know what you need to improve on.

| AWS instance type | vCPUs | RAM (approx.) | Analytical read execution time | Write execution time |
|---|---|---|---|---|
| m4.large | 2 | 4 GB | 157 seconds | 130 seconds |
| m4.2xlarge | 8 | 32 GB | 130 seconds | 34 seconds |
| m4.4xlarge | 16 | 64 GB | 119 seconds | 19 seconds |
| r4.large | 2 | 15 GB | 154 seconds | 125 seconds |
| r4.2xlarge | 8 | 61 GB | 132 seconds | 34 seconds |
| c4.2xlarge | 8 | 15 GB | 120 seconds | 33 seconds |
| c4.8xlarge | 36 | 60 GB | 122 seconds | 13 seconds |

**Miscellaneous tips from PayPal**

- Check for multiple metrics (such as those listed below) to ensure your cluster is healthy and able to scale in the future:
    - Hinted handoff
    - CPU/Mem/Disk
    - WAL/HH directories
    - IOPS_In_Progress
    - Etc.
- Send cluster stats and data to a different instance or cluster
- Avoid deleting measurements, since this is very expensive and taxing to the system. A better method, if you need to remove old data, is to remove the shard altogether, using the meta node remove shard feature.
- Use InfluxDB's native line protocol as much as possible. Perform data transformations only when necessary (such as to RDBMS).
- If you can't revive a dead data node quickly, it's best to remove it from the cluster (so data nodes don't keep retrying to send to that node) to free up some cycles and prevent Hinted Handoff buildup.
- Enable the time series index (TSI).
- Adjust fsync-delay for slower disks.
- Adjust WAL setting.

- Adjust compaction interval to an interval knowing you won't receive data past a certain time so recompaction doesn't reoccur.
- Take advantage of jitter feature in Telegraf to streamline delivery of metrics to your influx cluster.
- Ensure VMs are on different hypervisors (HVs) to avoid the risk of one hypervisor dying and bringing down your whole cluster.
- Use flash (SSD) storage for data nodes
- Use a reliable deployment strategy such as Puppet or Ansible.
- Have a "nanny" like SystemD in OEL7+ or supervisord to ensure your services stay running.

Below is a simple aggregator configuration that PayPal have. They send 5,000 metrics at any given time to InfluxDB. The HTTP listener emulates what InfluxDB does, listening on port 8086. You can apply changes, filters, and data transformation, at scale and at the aggregator levels. So they have a HTTP listener tagdrop.

PayPal's cloud team spin up and spin down instances all the time, testing images. But all of those hosting follow a certain pattern, and they don't want that in their system because it creates cardinality. So they specify the hostname to exclude and then setup the system to not send them information that has this hostname in the host tag. Also shown below are the two different outputs.

### Sample Aggregator Configuration

```
[agent]
  metric_batch_size = 5000

# Influx HTTP write listener
[[inputs.http_listener]]
  ## Address and port to host HTTP listener on
  service_address = ":8086"

  # Don't collect metrics where
  # host tag contains -HOSTNAME-
  [inputs.http_listener.tagdrop]
    host = ["*-HOSTNAME-*"]
```

```
# AMQP Output for Influx Publisher
[[outputs.amqp]]
  brokers = ["amqp://AMQP_UID:AMQP_PWD@AMQ_VIP:5672"]
  max_messages = 10000
  exchange = "influxdb_exchange"

# AMQP Output (in JSON format)
[[outputs.amqp]]
  brokers = ["amqp://AMQP_UID:AMQP_PWD@AMQ_VIP:5672"]
  max_messages = 10000
  exchange = "json_exchange_name"
  data_format = "json"
  json_timestamp_units = "1s"

  # Don't send metrics for any container measurement
  namedrop = ["container*"]
```

- Check/adjust system limits (especially for aggregator & data nodes) to ensure you have enough room for network connections and files.

Below are some of the settings used in PayPal's environment.

/etc/security/limits.d/<filename>.conf:

```
uid        hard    nproc          16384
uid        soft    nproc          16384
uid        hard    nofile         500000
uid        soft    nofile         500000
```

/etc/sysctl.d/<filename>.conf

```
net.core.somaxconn = 1024
net.ipv4.tcp_tw_recycle = 0
net.ipv4.tcp_tw_reuse = 0
net.ipv4.tcp_orphan_retries = 1
net.ipv4.tcp_fin_timeout = 20
net.ipv4.tcp_max_orphans = 8192
net.ipv4.ip_local_port_range = 18000  65535
net.ipv4.tcp_rmem = 4096 25165824 25165824
net.ipv4.tcp_wmem = 4096 65536 25165824
net.core.optmem_max = 25165824
net.ipv4.tcp_congestion_control = htcp
net.core.default_qdisc = fq
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_syn_retries = 2
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_max_syn_backlog = 4096
```

Powered by InfluxData, PayPal's new monitoring solution meets the technical requirements they set out to fulfill.

# About InfluxData

InfluxData is the creator of InfluxDB, the open source time series database. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by IoT devices, applications, networks, containers and computers. We are on a mission to help developers and organizations, such as Cisco, IBM, PayPal, and Tesla, store and analyze real-time data, empowering them to build transformative monitoring, analytics, and IoT applications quicker and to scale. InfluxData is headquartered in San Francisco with a workforce distributed throughout the U.S. and across Europe.

Learn more.

# InfluxDB documentation, downloads & guides

Download InfluxDB
Get documentation
Additional case studies
Join the InfluxDB community

799 Market Street
San Francisco, CA 94103
(415) 295-1901
www.InfluxData.com
Twitter: @InfluxDB
Facebook: @InfluxDB