饿了么 Influxdb 实践之路

刘平 GitChat技术杂谈 2017-06-23

> 文章来自作者：刘平 在 GitChat 上的分享
>
> 点击文末「阅读原文」这场 Chat 看看大家与作者交流了哪些问题

前言

Influxdb（https://github.com/influxdata/influxdb）是一个基于 golang 编写，没有额外依赖的开源时序数据库，用于记录 metrics、events，进行数据分析。这篇文章谈论的 influxdb 版本在**1.2.0**以上。

这篇文章只谈论 influxdb 在监控中的数据存储应用，不会谈论 influxdb 提供的整套监控方案。本文主要谈论五个方面：时序数据库选型、**influxdb** 基本概念、存储引擎、实践、数据聚合。

选型

Influxdb vs Prometheus

- influxdb 集成已有的概念，比如查询语法类似 sql，引擎从 LSM 优化而来，学习成本相对低。
- influxdb 支持的类型有 float，integers，strings，booleans，prometheus 目前只支持 float。
- influxdb 的时间精度是纳秒，prometheus 的则是毫秒。
- influxdb 仅仅是个数据库，而 prometheus 提供的是整套监控解决方案，当然 influxdb 也提供了整套监控解决方案。
- influxdb 支持的 math function 比较少，prometheus 相对来说更多，influxdb 就目前使用上已经满足功能。
- 2015年 prometheus 还在开发阶段，相对来说 influxdb 更加稳定。
- influxdb 支持 event log，prometheus 不支持。
- 更详细的对比请参考：对比（https://db-engines.com/en/system/Graphite%3BInfluxDB%3BPrometheus）。

我们其实仅仅需要的是一个数据库，其他组件都是自己开发的，而且存储的数据类型不仅仅是数字，因此选择了 influxdb。希望上面的比较对大家有帮助。

Influxdb 基本概念

htt 8e6feec4

Database

数据库是个逻辑容器，包含了 measurement、retention policies、continuous queries、time series data，类似于 mysql 的 database。

Measurement

描述了相关数据的存储结构，类似于 mysql 的 table，但是不需要创建，写入数据的时候自动创建。关于 schema 的设计建议参考：设计建议

（https://docs.influxdata.com/influxdb/v1.2/concepts/schema_and_data_layout/）。

Line Protocol

Line Protocol 定义了 influxdb 的数据写入格式，如下：

```
weather,location=us,server=host1 temperature=82 1465839830100400200  |    -----------------
```

Tag

上面的 location 和 server 就是 tag key，us 和 host1 是 tag value，tag 是可选的。不过写入数据时最好加上 tag，因为它可以被索引。tag 的类型只能是字符串。

Field

上面的 temperature 是 field key，82是 field value。field value 会用于展示，value 支持的类型有 floats，integers，strings，booleans。

Timestamp

格式是：RFC3339 UTC。默认精确到纳秒，可选。

Series

measurement, tag set, retention policy 相同的数据集合算做一个 series。理解这个概念至关重要，因为这些数据存储在内存中，如果 series 太多，会导致 OOM。

Retention Policy

保留策略包括设置数据保存的时间以及在集群中的副本个数。默认配置是：RP 是 autogen，保留时间是永久，副本为1。这些配置在创建数据库时可以修改。

Continuous Query

CQ 是预先配置好的一些查询命令，定期自动执行这些命令并将查询结果写入指定的 measurement 中，这个功能主要用于数据聚合。具体参考：CQ（https://docs.influxdata.com/influxdb/v1.2/query_language/continuous_queries/）。

Shard

存储一定时间间隔的数据，每个目录对应一个 shard，目录的名字就是shard id。每一个 shard 都有自己的 cache、wal、tsm file 以及 compactor，目的就是通过时间来快速定位到要查询数据的相关资源，加速查询的过程，并且也让之后的批量删除数据的操作变得非常简单且高效。

存储引擎

概述

TSM Tree 是在 LSM Tree 的基础上稍作修改优化而来。它主要包含四个部分：cache、wal、tsm file、compactor。

Cache

插入数据时，先往 cache 中写入再写入 wal 中，可以认为 cache 是 wal 文件中的数据在内存中的缓存。

WAL

预写日志，对比 mysql 的 binlog。其作用就是为了持久化数据，当系统崩溃后可以通过 wal 文件恢复 cache。

TSM File

每个 tsm 文件的大小上限是 2GB。当达到 `cache-snapshot-memory-size`，`cache-max-memory-size` 的限制时会触发将 cache 写入 tsm 文件。

Compactor

主要进行两种操作，一种是 cache 数据达到阀值后，进行快照，生成一个新的 tsm 文件。另外一种就是合并当前的 tsm 文件，将多个小的 tsm 文件合并成一个，减少文件的数量，并且进行一些数据删除操作。 这些操作都在后台自动完成。

目录结构

InfluxDB 的数据存储有三个目录，分别是 meta、wal、data。meta 用于存储数据库的一些元数据，meta目录下有一个 meta.db 文件。wal 目录存放预写日志文件，以 .wal 结尾。data 目录存放实际存储的数据文件，以 .tsm 结尾。基本结构如下：

```
-- wal   -- test       -- autogen         -- 1            -- _00001.wal         -- 2
```

其中 **test** 是数据库名称，**autogen** 是存储策略名称，再下一层目录中的以数字命名的目录是 shard 的 ID 值，比如 **autogen** 存储策略下有两个 shard，ID 分别为 1 和 2，shard 存储了某一个时间段范围内的数据。再下一级的目录则为具体的文件，分别是 `.wal` 和 `.tsm` 结尾的文件。

更详细的参考

InfluxDB 详解之 TSM 存储引擎解析（http://blog.fatedier.com/2016/08/05/detailed-in-influxdb-tsm-storage-engine-one/）

实践

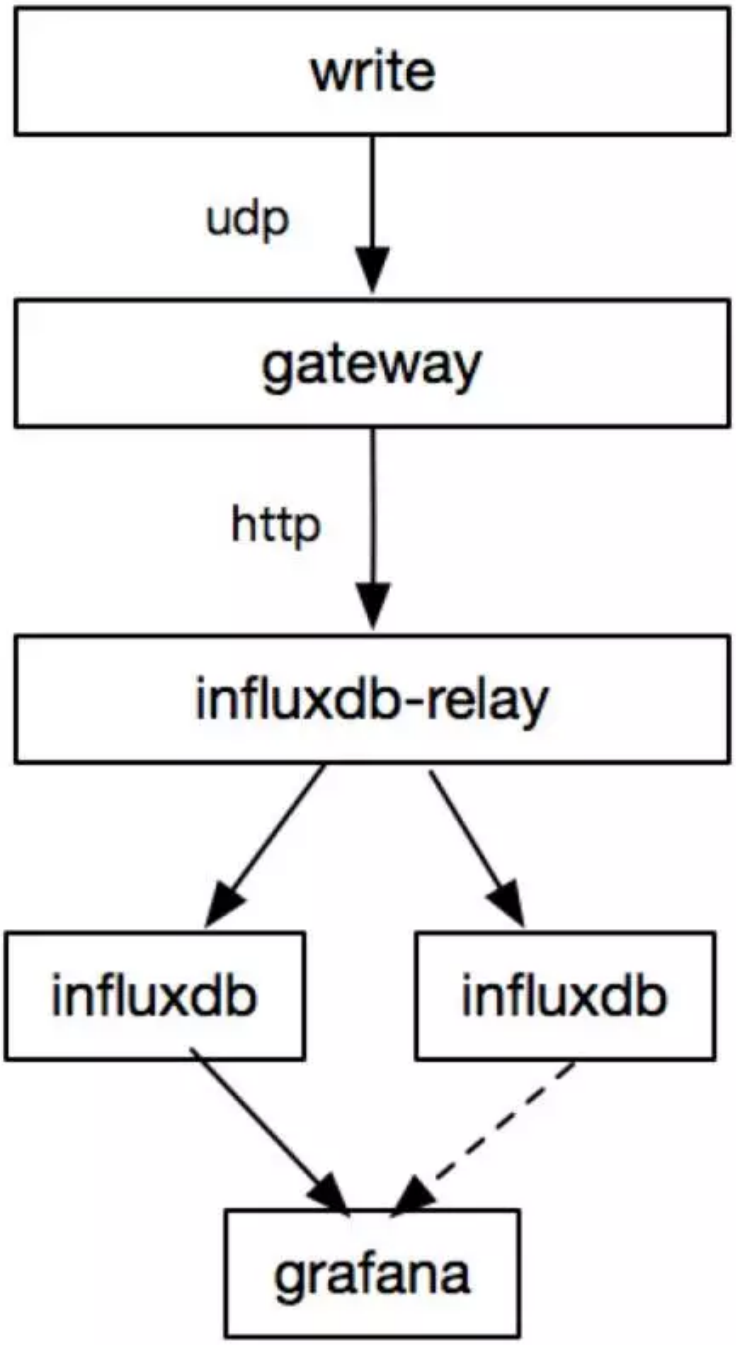
项目介绍

gateway（https://github.com/pingliu/influxdb-gateway）用于检测和压缩influxdb的数据，用于跨机房传输，采用udp接受数据。

influxdb-relay（https://github.com/influxdata/influxdb-relay）是官方提供的高可用方案，但是它只提供简单的写入功能。

influxdb-proxy（https://github.com/shell909090/influx-proxy）是用于替代 influxdb-relay 的高可用方案。

前期架构图



使用问题

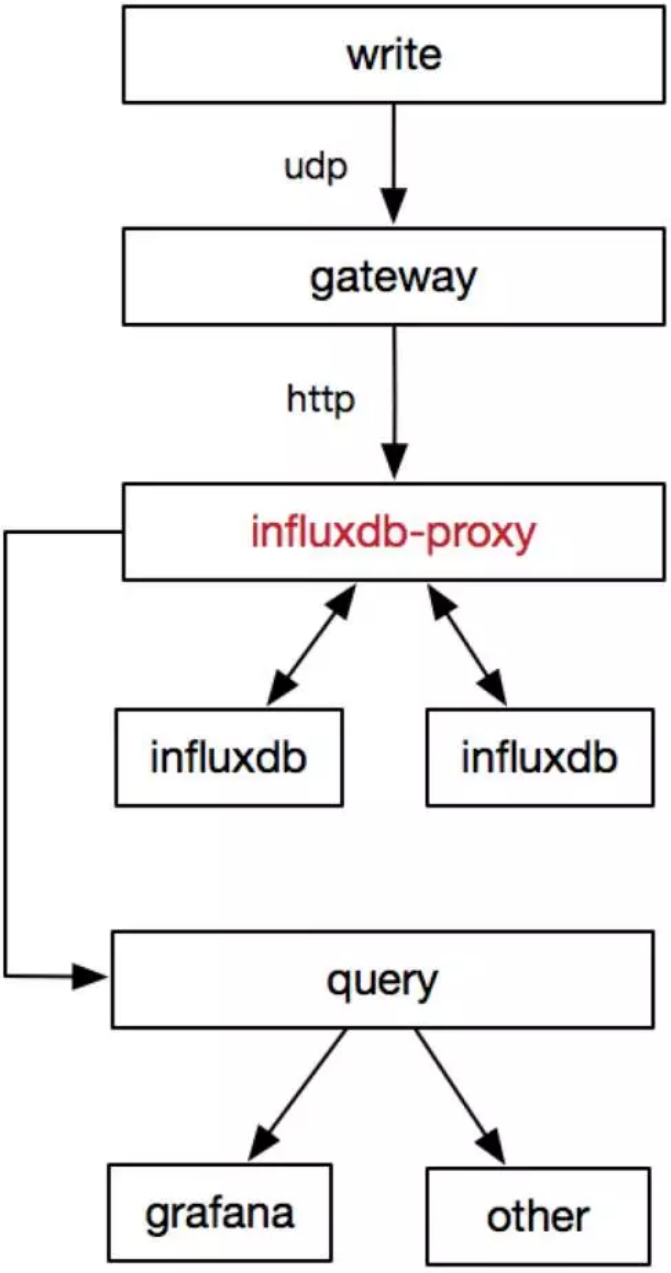
influxdb-relay 是官方提供的高可用方案，但是它只提供简单的写入功能。在初期使用时，并没有多大的问题，随着 influxdb 在公司的推广，接入方越来越多，意味着查询方越来越多，这就带来了以下问题：

- grafana 需要配置很多个数据源。
- 用户不能根据 measurement 来订阅数据。
- 数据库挂掉，就需要修改 grafana 的数据源。
- 维护困难，比如需要新增数据库，用户需要配置多个数据源，不能统一接入点。
- 用户查询直连数据库，用户 `select *` 数据库直接 OOM，数据库会重启。
- relay提供的重写功能，数据是保留在内存中，一旦 influxdb 挂掉，就会导致relay机器内存疯涨。

踩过的坑

- `max-row-limit` 不为0，会导致 influxdb OOM。目前这个问题已经修复，但是 grafana 展示时会存在问题，配置时请设置为0。
- 配置查询限制参数时，会导致一些奇怪的问题，官方是不限制，请保留默认配置。
- 没有制定 schema 规范，接入方把 field写成 tag 了，导致内存疯涨，最后 OOM。理解 series 的概念很重要。
- 写入超时时间默认是10s，有时候数据写入了但返回 500。可以将这个时间设置成大点。

优化后的架构图



influxdb-proxy 是为了解决上面的使用问题而开发出来的。具有以下功能：

- 同时支持写和查询功能，统一接入点，类似cluster。
- 支持重写功能，写入失败时写入文件，后端恢复时再写入。
- 限制部分查询命令和全部删除操作。
- 以 measurement 为粒度区分数据，支持按需订阅。
- measurement 优先精确匹配，然后前缀匹配。
- 提供数据统计，比如 qps，耗时等等。

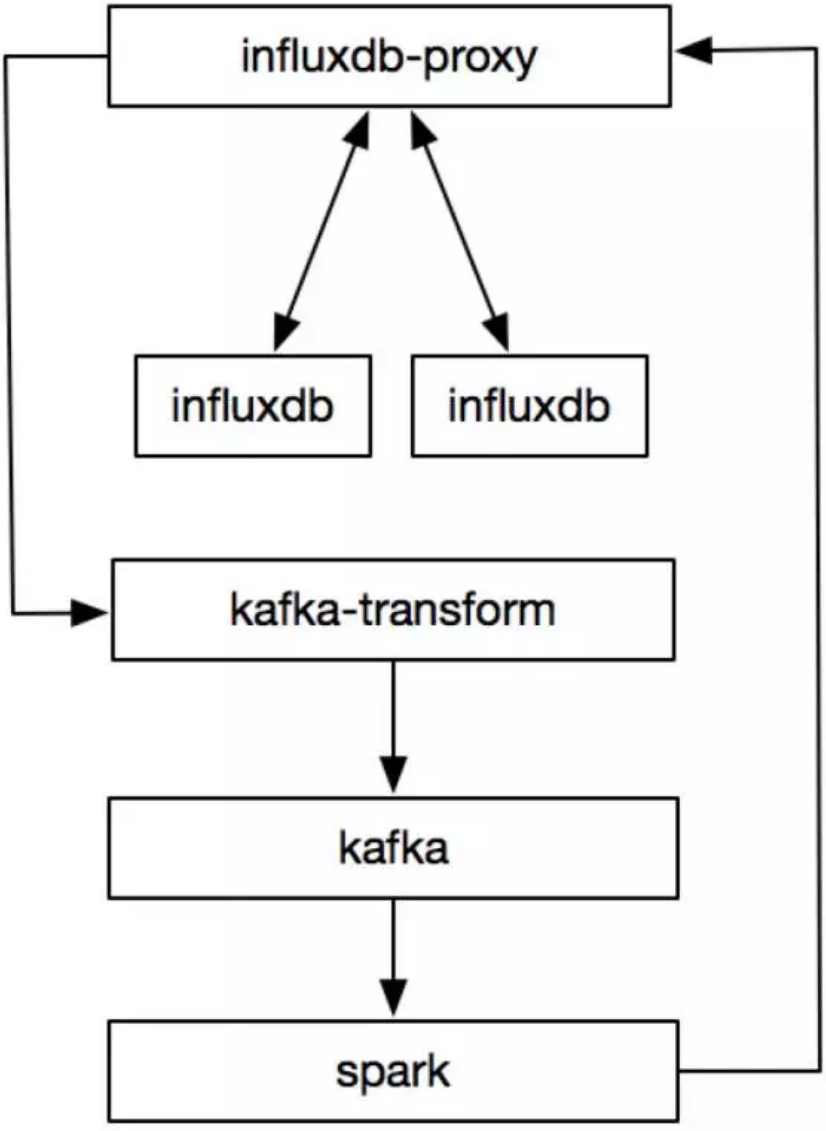
数据聚合

CQ

influxdb 提供数据聚合的功能，就是上面基本概念里提到的 Continuous Query。预先定义好cq，就可以定期根据不同的tag进行聚合数据。目前它有个设计问题：cq 是顺序执行的，cq 越多，数据延迟越高，一般延迟在几分钟内。如果需要更实时的聚合，cq 不能满足，需要引入其他工具，比如spark。关于 cq 的语法请参考：语法（https://docs.influxdata.com/influxdb/v1.2/query_language/continuous_queries/）。

Spark

经过内部调研，发现 spark+kafka 是个更好的聚合方案。spark支持流式处理且支持 sql 功能，我们只需要将cq改成sql就行。目前这个处于尝试阶段，已经上线部分功能。目前的处理流程如下：



总结

上文讲的整套架构已经支撑起饿了么2万台机器的监控，目前每秒写入的点数是300k。后端 influxdb 的机器数量是20台左右，维护成本基本趋于零。我们的焦点目前已经从 influxdb 转移到数据聚合和分析上。

「阅读原文」回顾本次 chat

阅读原文