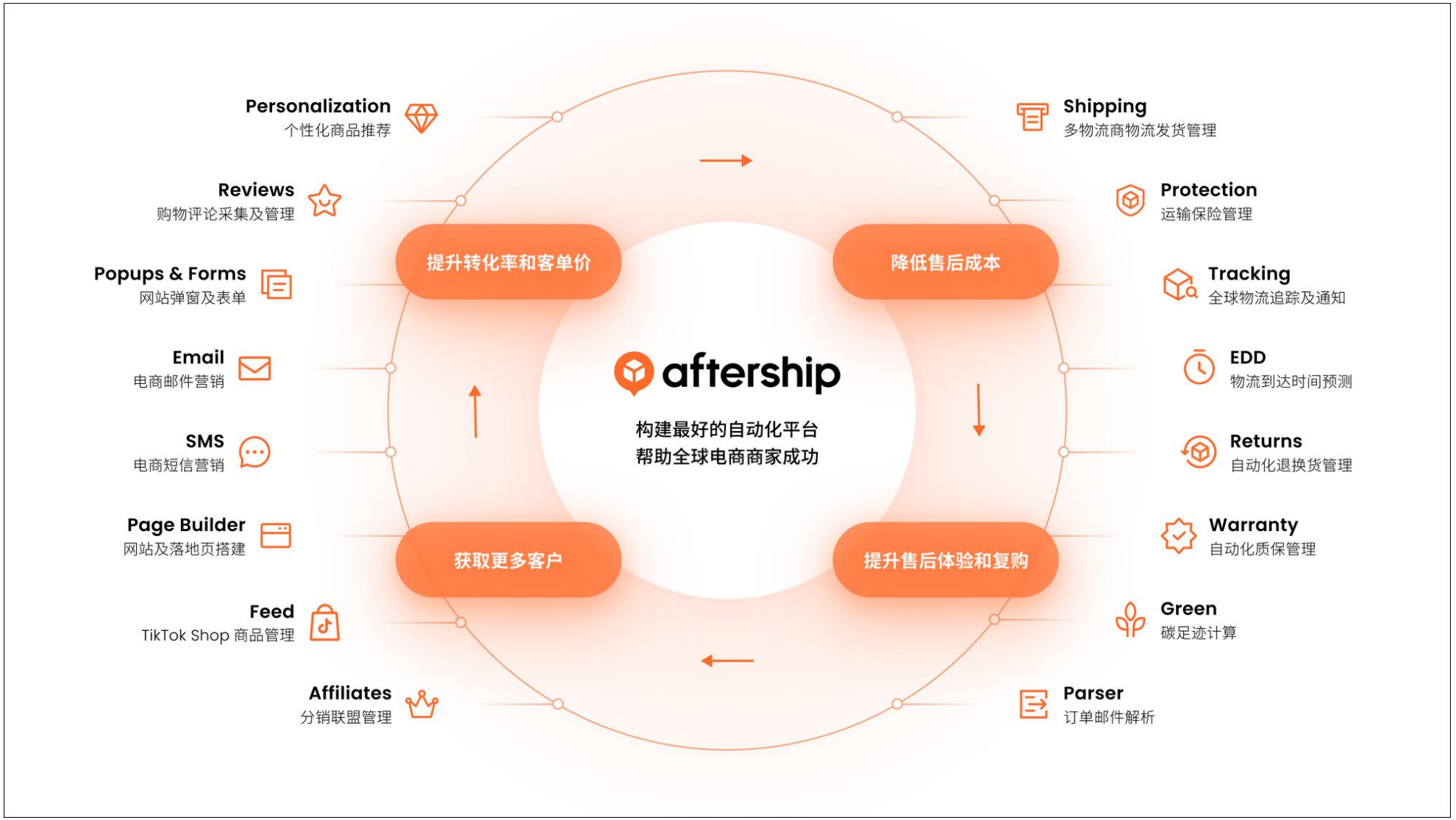
每秒百万数据点 Go 应用监控系统演进

×

张平

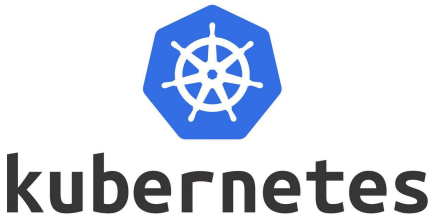
AfterShip
高级 SRE

关于 AfterShip



拥抱云原生和开源系统


CLOUD NATIVE
COMPUTING FOUNDATION


kubernetes


Prometheus


GO


VICTORIA
METRICS


argo


LMSTFY

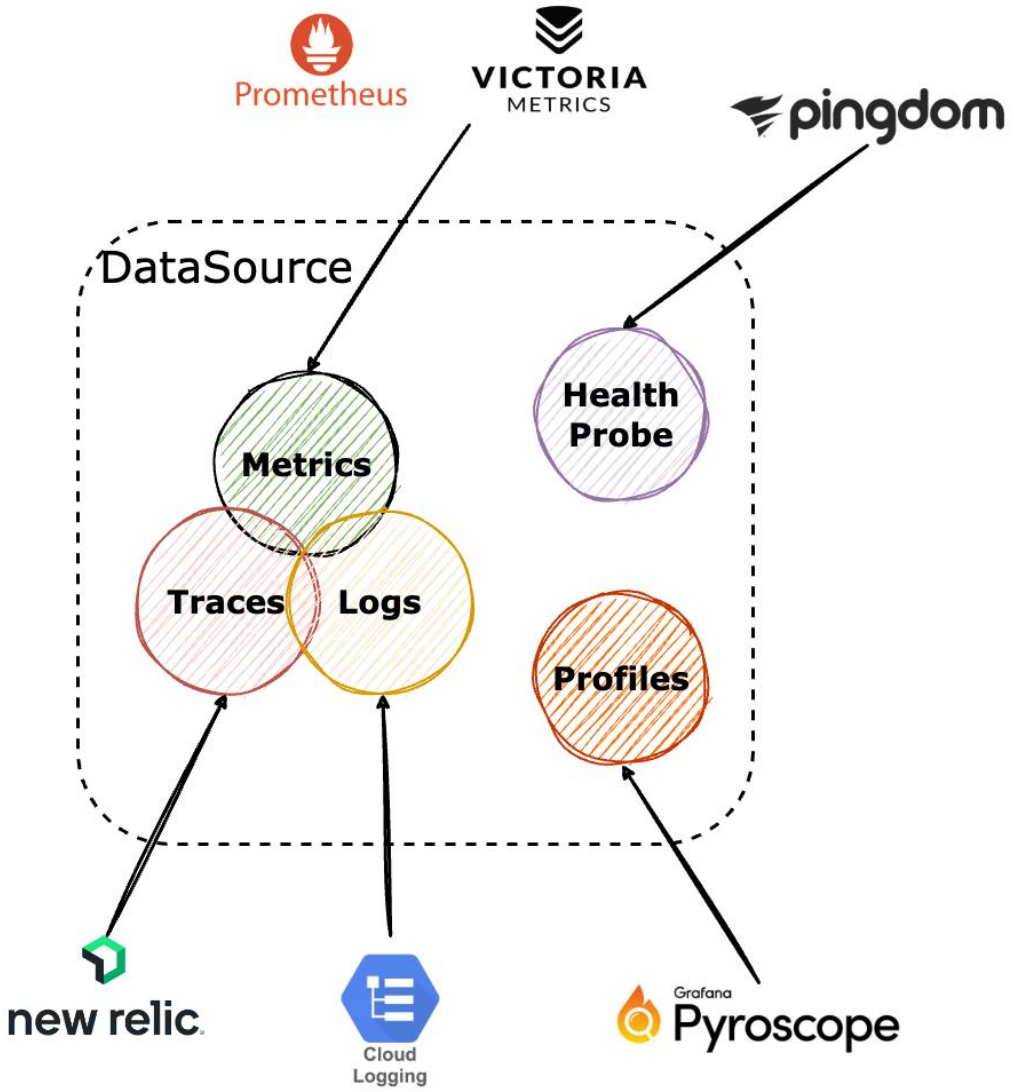

Grafana

目录

第一部分

监控架构概览

监控系统架构概览 -- 数据源



监控系统架构概览 -- 告警配置

监控系统架构概览 -- 告警通道

PagerDuty

第二部分

如何监控 Go 应用？

基于 Prometheus Go 应用监控接入流程

指标类型

- Go 运行时指标
  - Goroutine 数量
- 应用层指标
  - infra_http_request_total
- 业务指标
  - 总 Tracking 查询量
  - Tracking 创建速率
  - 某个 ENT 客户的 Tracking 查询失败率

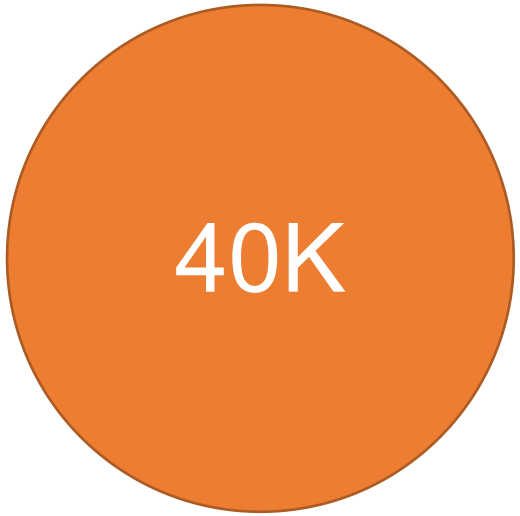
第三部分

Metrics 系统架构演进

2018-2020

2020 年指标数据

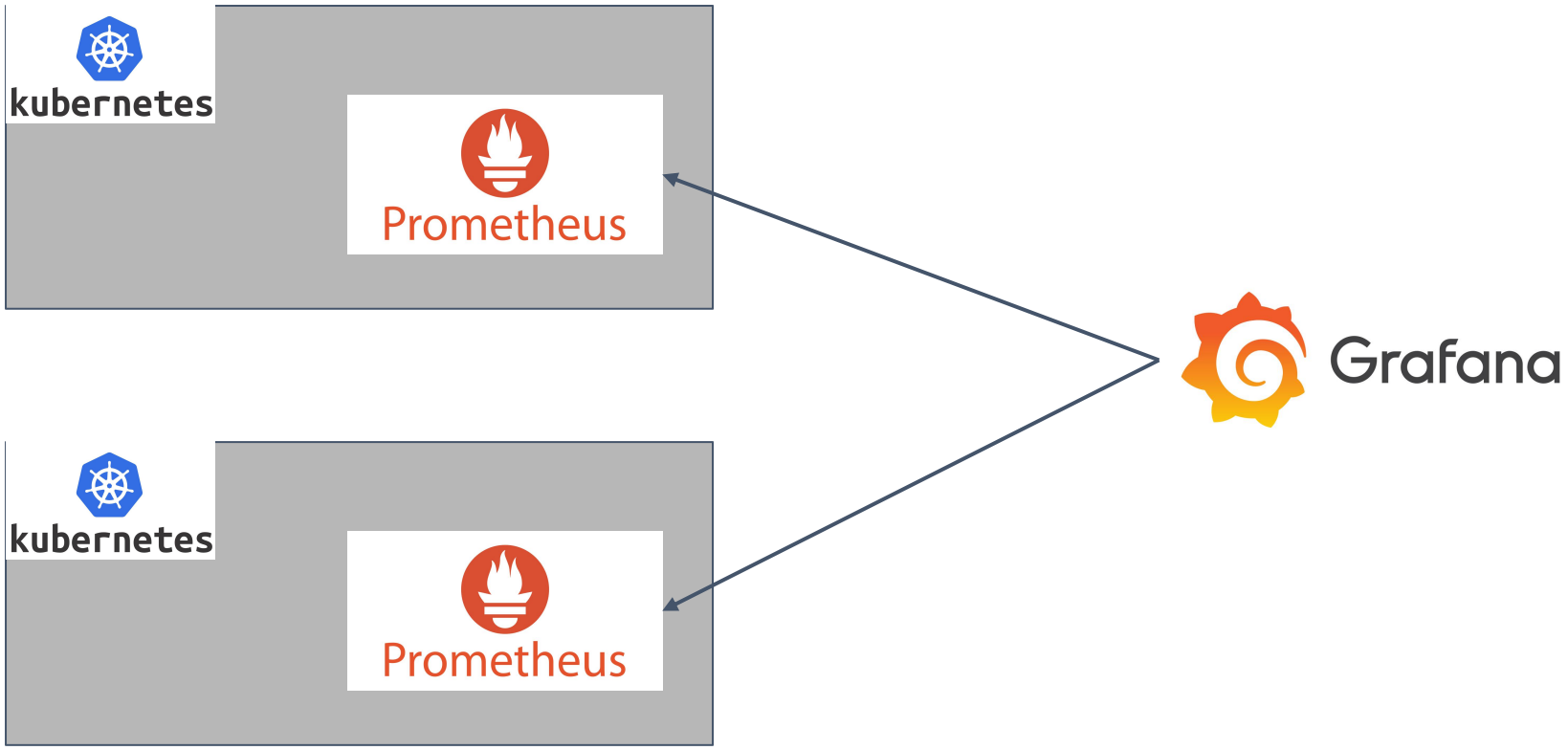
业务指标数量



每秒写入数据点

Active Time Series

2018-2020 年架构



2020 年底面临的问题

- 无法查询超过 30 天的数据
- 查询慢，平均时间超过 2 分钟
- 跨集群指标无法聚合
- Prometheus 集群经常崩溃
- 维护时 Prometheus 会丢数据
- 成本高，需要大容量 SSD 磁盘

2021-2022

核心需求

长期存储

可跨集群查询

兼容
Prometheus

扩展性强

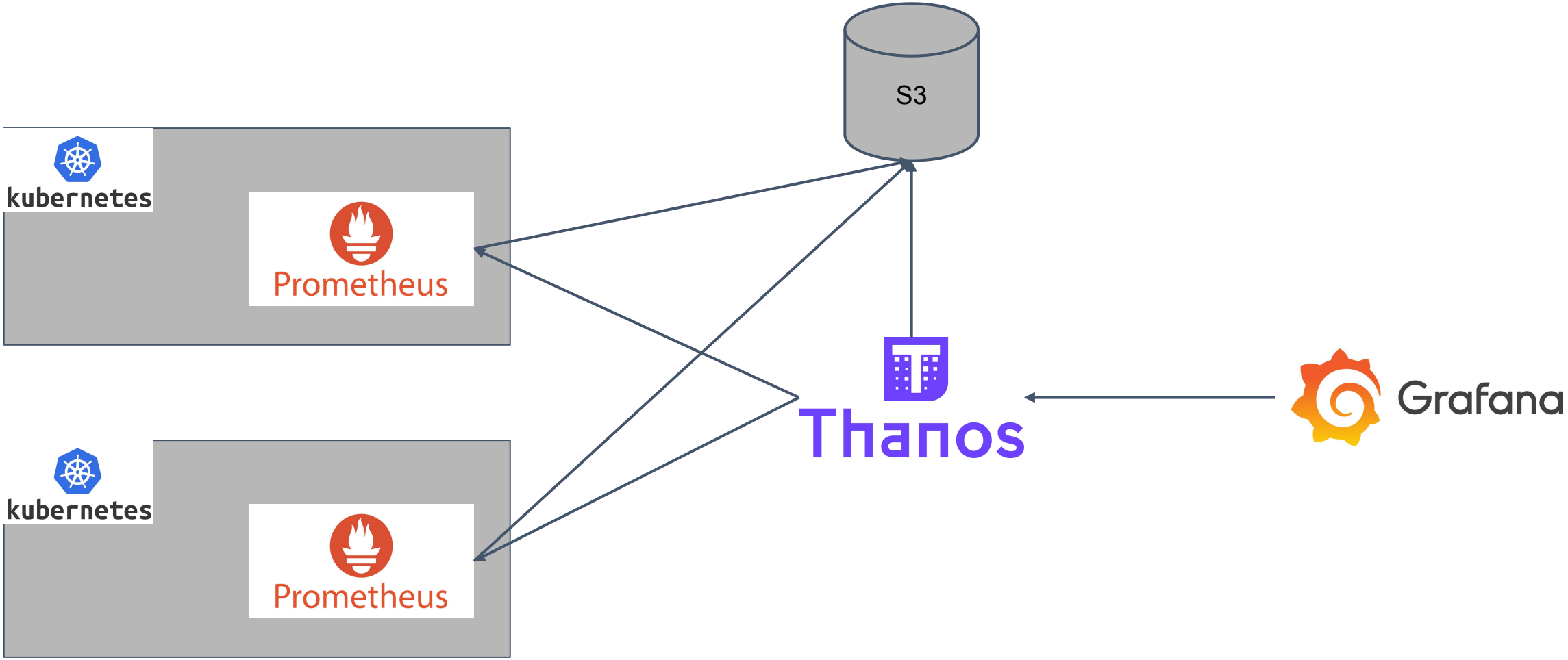
无侵入性

Why Thanos

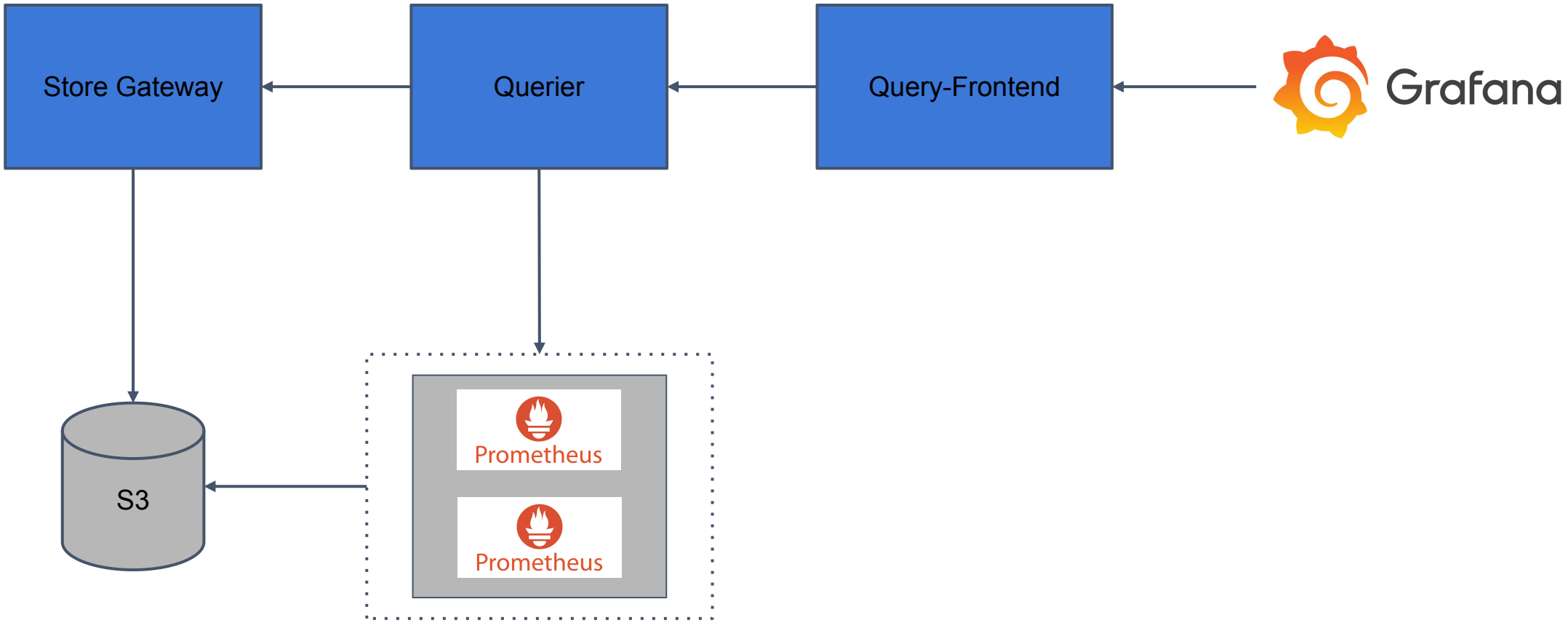
VS

VS

架构


S3
kubernetes
Prometheus
kubernetes
Prometheus
Thanos
Grafana

Thanos 架构


Store Gateway
Querier
Query-Frontend
Grafana
S3
Prometheus
Prometheus

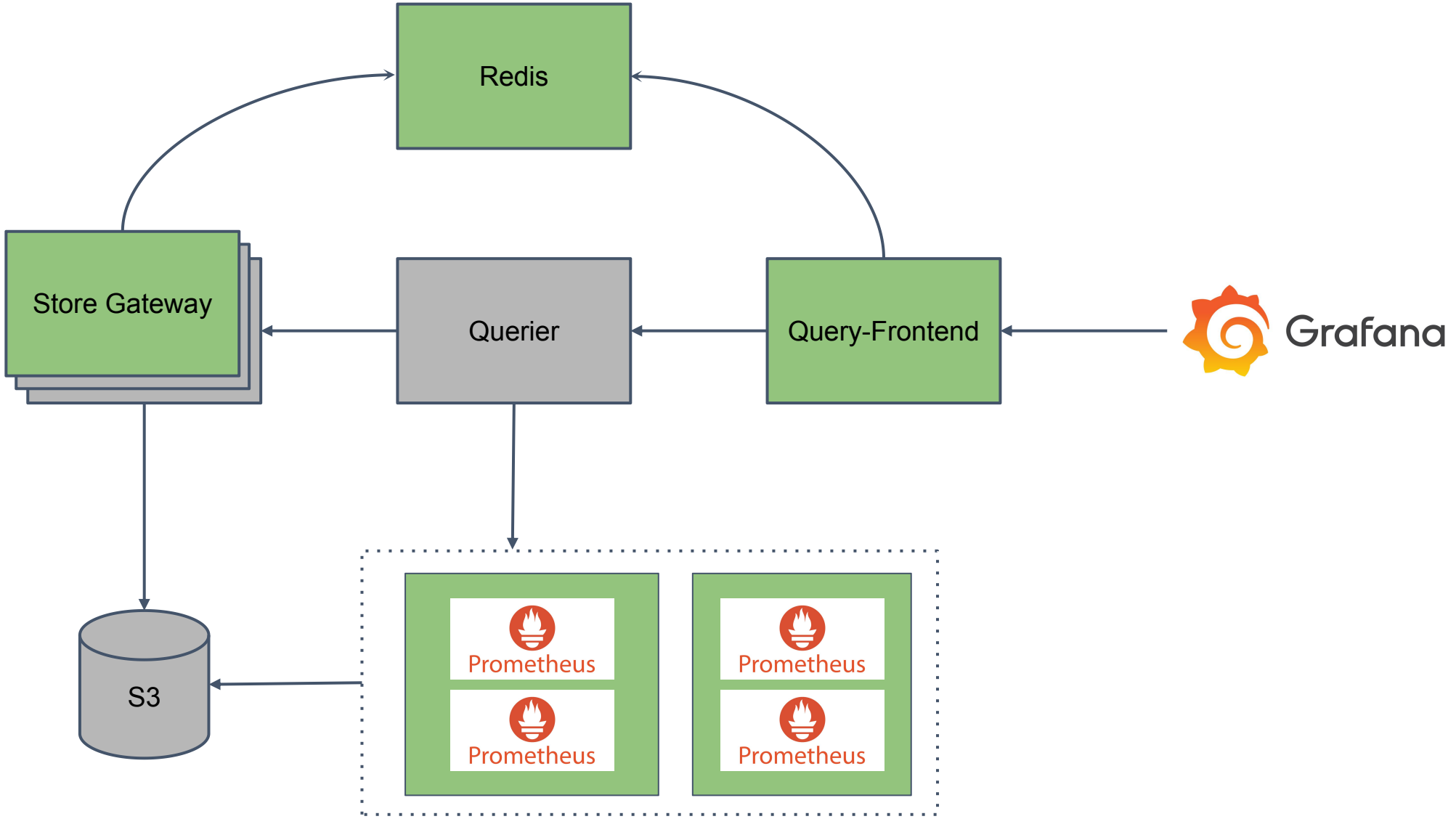
2022 年中指标数据

业务指标数量

每秒写入数据点

Active Time Series

Thanos 架构优化


Redis
Store Gateway
Querier
Query-Frontend
Grafana
S3
Prometheus
Prometheus
Prometheus
Prometheus

2022 年底面临的问题

- 超 100+ 倍数据点增长导致查询缓慢
- 架构复杂，参数调优困难
- 频繁 OOM
- 集群规模受制于 Prometheus
- 集群成本上升

2023

压测结果

- CPU 使用低 1.7 倍
- RAM 使用减少 5 倍
- 存储空间减少了 3 倍

VS

2023 年底指标数据

业务指标数量

每秒写入数据点

Active Time Series

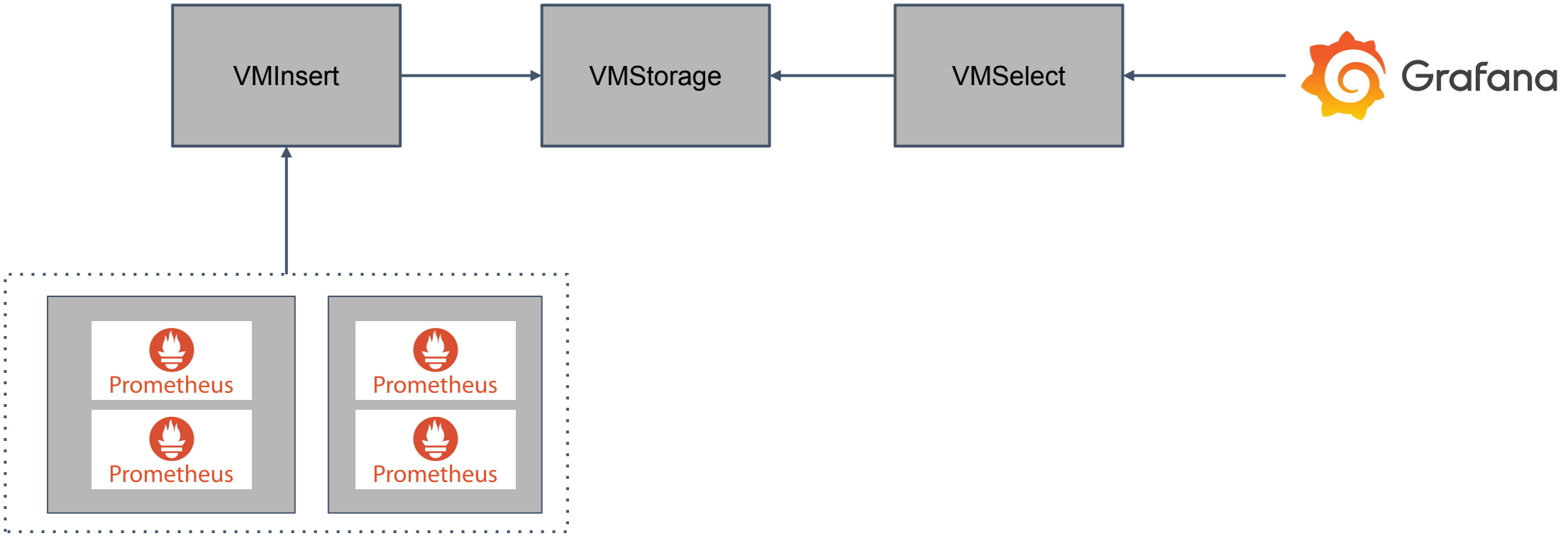
VictoriaMetrics 收益

- 高性能，看板加载时间从 120s 降低到 10s
- 兼容 Prometheus，可以无缝迁移
- 成本更低，只需要 thanos 的 50% 资源
- 扩展性强，所有组件支持水平扩容

2023 年底架构


kubernetes
Prometheus
kubernetes
Prometheus
VICTORIA
METRICS
Grafana

VictoriaMetrics 架构



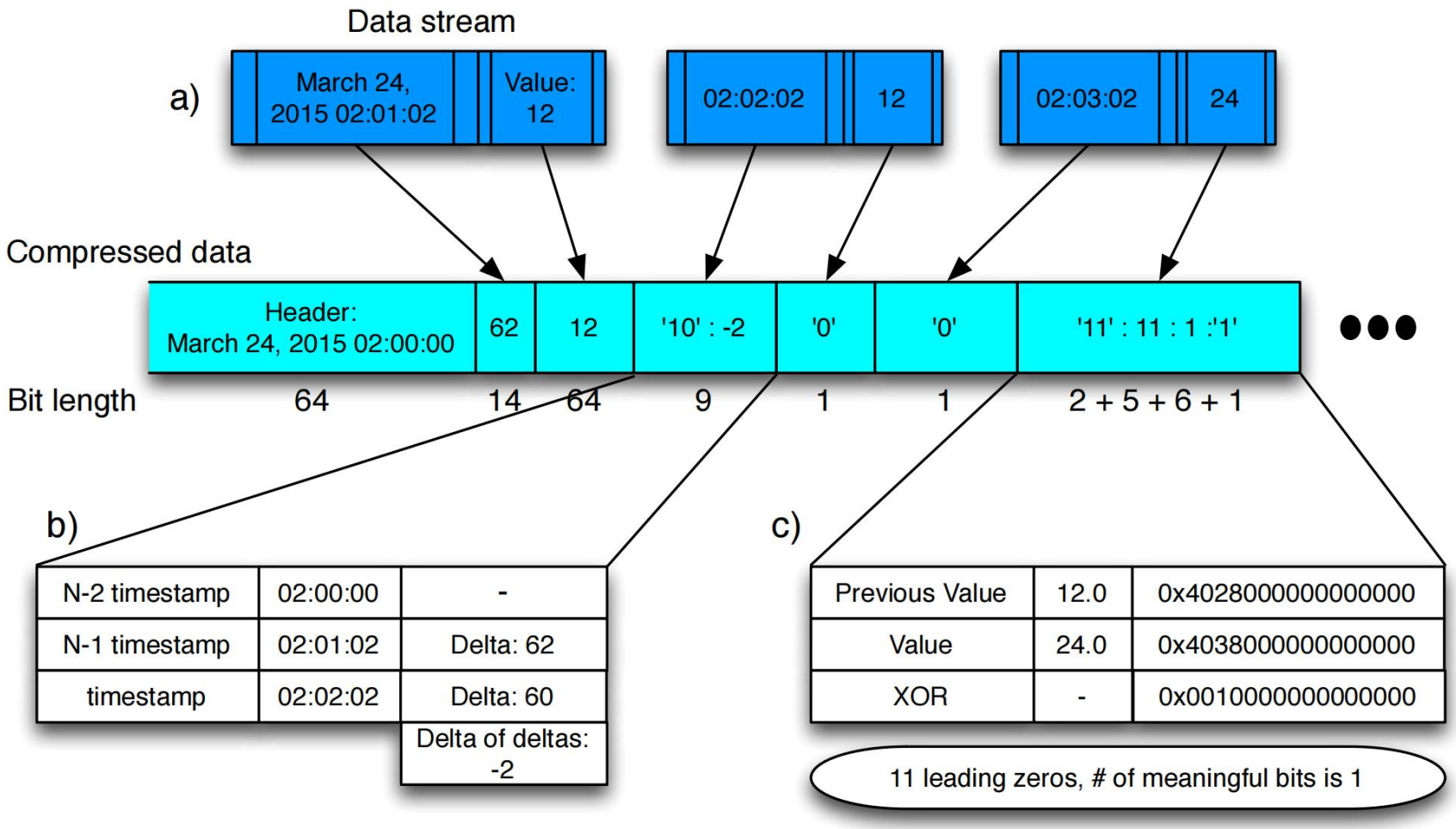
第四部分

Why VictoriaMetrics so good?

极致的设计与优化

- 根据容器可用的 CPU 数量计算协程数量
- 区分 IO 协程和计算协程，同时提供了协程优先级策略
- 使用 ZSTD 压缩传输内容，降低磁盘性能要求
- 根据可用物理内存限制对象的总量，避免 OOM
- 区分 fast path 和 slow path

Gorilla 压缩算法



压缩算法优化

- 改进 XOR，通过应用 10^X 乘数将 value 浮点值转换为整数值
- 根据具体情况选择压缩算法
  - 数值相同，那么只存储第一个值
  - 数值是等差数列，那么只存存储第一个值和 Delta 值
  - Gauges 类型的 value 先用一阶增量编码(Delta)压缩，然后再用 zigzag 算法压缩
  - Counters 类型的 value 先用二阶增量编码(Delta of Delta)压缩，然后再用 zigzag 算法压缩
- 最后再应用 ZSTD 算法进行二次压缩

压缩率对比

	Prometheus	Thanos	VictoriaMetrics
Bytes / Sample	1.2B ~ 1.79B	1.5B	0.69B

不可忽略的问题

- 数据完整性校验缺失
- 可能会丢数据
  - 没有 WAL(Write-Ahead Log)
- 扩容/维护时可能容易崩溃
  - vmstorage 没有服务自动发现
- 兼容性需要关注
  - MetricsQL 与 PromQL 有差异

```
func newEncoder(compressionLevel int) *zstd.Encoder {
        level := zstd.EncoderLevelFromZstd(compressionLevel)
        e, err := zstd.NewWriter(nil,
                zstd.WithEncoderCRC(false), // Disable CRC for performance reasons.
                zstd.WithEncoderLevel(level))
        if err != nil {
                logger.Panicf("BUG: failed to create ZSTD writer: %s", err)
        }
        return e
}
```

第五部分

总结与展望

2023 关键成果

- 查询性能大幅提升，用户体验好
- 稳定性大幅提升，几乎没有 OOM
- 资源成本得到降低，至少降低 30% 的成本

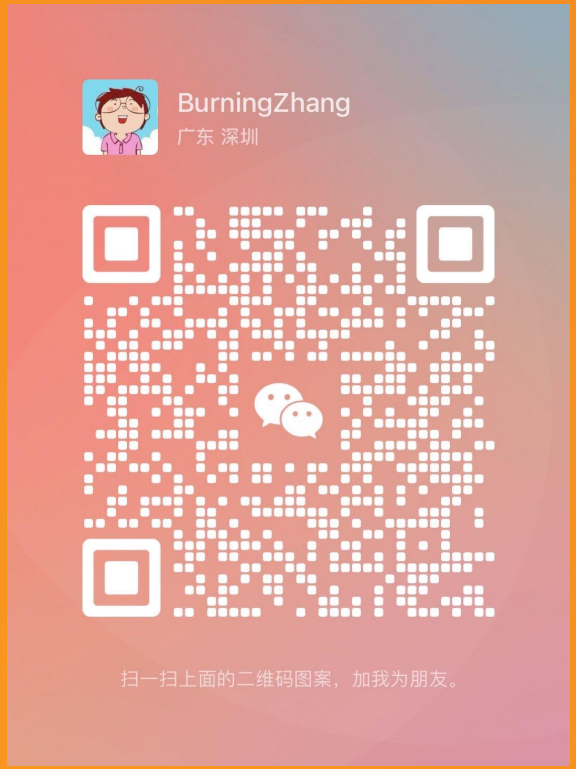
2024 优化方向

- 成本优化
  - 使用 vmagent 替换 Prometheus
  - 根据实际需求调整指标存储周期
- 性能优化
  - 优化高基数指标

Takeaway

- 不同阶段选型侧重点不同
  - 兼容 Prometheus 是首要关注点
  - 随着业务增长，性能和成本需要重点关注
- 不要相信互联网上的报告
  - 你可以并行运行多个方案，然后再决定

扫码关注 AfterShip 公众号，
获取更多技术、职场干货。



扫码添加我的个人微信，
期待与大家进行技术交流。

Q&A