基于 Grafana LGTM 可观测性平台的快速构建

作者：Grafana 爱好者

2022-11-01 • 江苏 • 本文字数：3086 字 • 阅读完需：约 10 分钟

可观测性目前属于云原生一个比较火的话题，它涉及的内容较多，不仅涉及多种遥测数据（信号），例如日志（log）、指标(metric)、分布式追踪（trace）、连续分析(continuous profiling)、事件（event）等；还涉及遥测数据各生命周期管理，比如暴露、采集、存储、计算查询、统一看板等。

目前社区相关开源产品较多，各有各的优势，今天我们就来看看如何使用 Grafana LGTM 技术栈快速构建一个自己的可观测性平台。

通过本文你将了解：

- 如何在 Go 程序中导出 metric、trace、log、以及它们之间的关联 TraceID
- 如何使用 OTel Collector 进行 metric、trace 收集
- 如何使用 OTel Collector Contrib 进行日志收集
- 如何部署 Grafana Mimir、Loki、Tempo 进行 metric、trace、log 数据存储
- 如何使用 Grafana 制作统一可观测性大盘

为了本次的教学内容，我们提前编写了一个 Go Web 程序，它提供 `/v1/books` 和 `/v1/books/1` 两个 HTTP 接口。

当请求接口时，会先访问 Redis 缓存，如果未命中将继续访问 MySQL；整个请求会详细记录相关日志、整个链路各阶段耗时和调用情况以及整体请求延迟，当请求延迟 >200ms 时，会通过 Prometheus examplar 记录本次请求的 TraceID，用于该请求的日志、调用链关联。

下载并体验样例

我们已经提前将样例程序上传到 github，所以您可以使用 git 进行下载：

```
git clone https://github.com/grafanafans/prometheus-exemplar.git
cd  prometheus-exemplar

```

使用 docker-compose 启动样例程序:

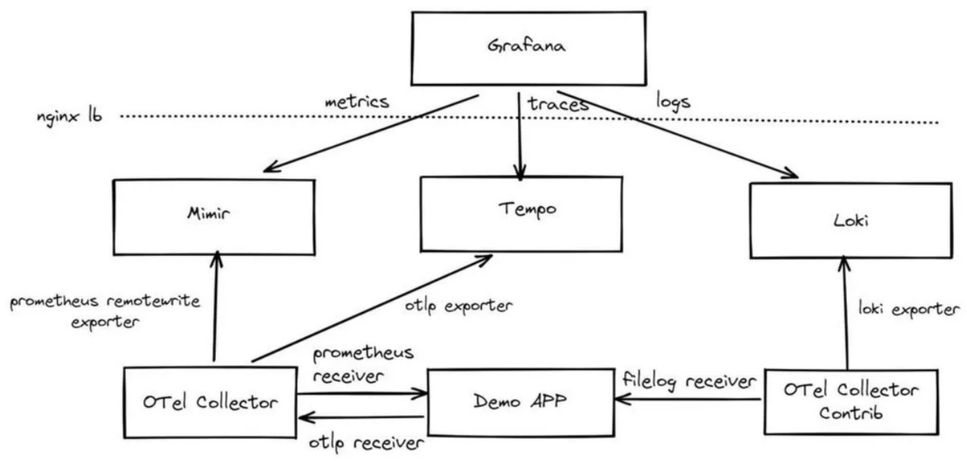
```
docker-compose up -d

```

这个命令会启动以下程序：

1. 使用单节点模式分别启动一个 Mimir、Loki、Tempo
2. 启动一个 Nginx 作为统一可观测平台查询入口，后端对接 Mimir、Loki、Tempo
3. 启动 demo app, 并启动其依赖的 MySQL 和 Redis， demo app 可以使用 `http://localhost:8080` 访问
4. 启动 Grafana 并导入预设的数据源和 demo app 统一看板，可以使用 `http://localhost:3000` 访问

整个部署架构如下：

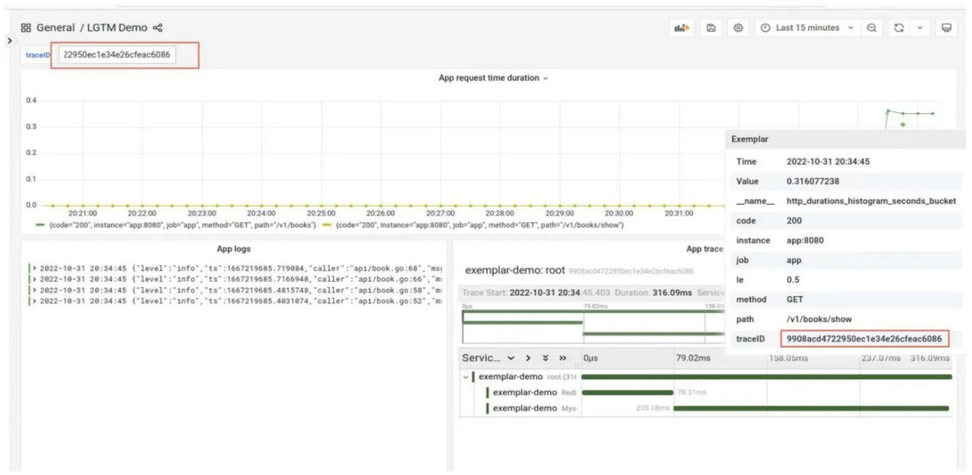


当程序部署完成后，我们可以使用 wrk 进行 demo app 接口批量请求：

```
wrk http://localhost:8080/v1/books
wrk http://localhost:8080/v1/books/1

```

最后通过 `http://localhost:3000` 页面访问对应的统一看板：



细节说明

使用 Promethues Go SDK 导出 metrics

在 demo app 中，我们使用 Prometheus Go SDK 作为 Metrics 导出，这里没有 OpenTelmetry SDK 主要因为当前版本（v0.33.0）还不支持 Exemplar， 代码逻辑大致为：

```
func Metrics(metricPath string, urlMapping func(string) string) gin.HandlerFunc {
  httpDurationsHistogram := prometheus.NewHistogramVec(prometheus.HistogramOpts{
    Name:    "http_durations_histogram_seconds",
    Help:    "Http latency distributions.",
    Buckets: []float64{0.05, 0.1, 0.25, 0.5, 1, 2},
  }, []string{"method", "path", "code"})

  prometheus.MustRegister(httpDurationsHistogram)

  return func(c *gin.Context) {
    .....
    observer := httpDurationsHistogram.WithLabelValues(method, url, status)
    observer.Observe(elapsed)

    if elapsed > 0.2 {
      observer.(prometheus.ExemplarObserver).ObserveWithExemplar(elapsed, prometheus.Labels{
        "traceID": c.GetHeader(api.XRequestID),
      })
    }
  }
}
```

使用 OTLP HTTP 导出 traces

使用 OTel SDK 进行 trace 埋点：

```

func (*MysqlBookService) Show(id string, ctx context.Context) (item *Book, err error) {
  _, span := otel.Tracer().Start(ctx, "MysqlBookService.Show")
  span.SetAttributes(attribute.String("id", id))
  defer span.End()

  // mysql qury random time duration
  time.Sleep(time.Duration(rand.Intn(250)) * time.Millisecond)

  err = db.Where(Book{Id: id}).Find(&item).Error
  return
}

```

使用 OLTP HTTP 进行导出：

复制代码

```
func SetTracerProvider(name, environment, endpoint string) error {
  serviceName = name

  client := otlptracehttp.NewClient(
    otlptracehttp.WithEndpoint(endpoint),
    otlptracehttp.WithInsecure(),
  )

  exp, err := otlptrace.New(context.Background(), client)
  if err != nil {
    return err
  }

  tp := tracesdk.NewTracerProvider(
    tracesdk.WithBatcher(exp),
    tracesdk.WithResource(resource.NewWithAttributes(
      semconv.SchemaURL,
      semconv.ServiceNameKey.String(serviceName),
      attribute.String("environment", environment),
    )),
  )

  otel.SetTracerProvider(tp)

  return nil
}
```

结构化日志

这里我们使用 `go.uber.org/zap` 包进行结构化日志输出，并输出到 `/var/log/app.log` 文件，并在每个请求开始的时候，注入 traceID：

复制代码

```
cfg := zap.NewProductionConfig()
cfg.OutputPaths = []string{"stderr", "/var/log/app.log"}
logger, _ := cfg.Build()


logger.With(zap.String("traceID", ctx.GetHeader(XRequestID)))
```

使用 OTel Collector 进行 metric、trace 收集

因为 demo app 的 metrics 使用 Prometheus SDK 导出，所以 OTel Collector 需要使用 Prometheus recevier 进行抓取，然后我们再通过 Prometheus remotewrite 将数据 push 到 Mimir；而针对 traces，app 使用 OTLP HTTP 进行了导出，所有 Collector 需要用 OTP HTTP recevier 进行接收，最后再使用 OTLP gRPC 将数据 push 到 Tempo，对应配置如下：

复制代码

```
receivers:
  otlp:
    protocols:
      grpc:
      http:

  prometheus:
    config:
      scrape_configs:
      - job_name: 'app'
        scrape_interval: 10s
        static_configs:
        - targets: ['app:8080']

exporters:
  otlp:
    endpoint: tempo:4317
    tls:
      insecure: true

  prometheusremotewrite:
    endpoint: http://mimir:8080/api/v1/push
    tls:
      insecure: true
    headers:
      X-Scope-OrgID: demo

processors:
  batch:

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlp]
    metrics:
      receivers: [prometheus]
      processors: [batch]
      exporters: [prometheusremotewrite]
```

使用 OTel Collector Contrib 进行 log 收集

因为我们结构化日志输出到 `/var/log/app.log` 文件，所以这里使用 `filelog` receiver 进行文件扫扫描，最后再经过 `loki` exporter 进行导出，配置如下：

复制代码

```yaml
receivers:
  filelog:
    include: [/var/log/app.log]

exporters:
  loki:
    endpoint: http://loki:3100/loki/api/v1/push
    tenant_id: demo
    labels:
      attributes:
        log.file.name: "filename"

processors:
  batch:

service:
  pipelines:
    logs:
      receivers: [filelog]
      processors: [batch]
      exporters: [loki]

```

以上就是有关 demo app 可观测性与 Grafana LGTM 技术栈集成的核心代码与配置，全部配置请参考 https://github.com/grafanafans/prometheus-exemplar 。

总结

本文我们通过一个简单的 Go 程序，导出了可观测性相关的遥测数据，其中包括 metrics、traces、logs, 然后统一由 OTel Collector 进行抓取，分别将三种遥测数据推送到 Grafana 的 Mimir、Tempo、Loki 进行存储，最后再通过 Grafana 统一看板进行 metrics、traces、logs 关联查询。

其关联为 Prometheus 的 exemplar 数据中的 traceID，通过该 traceID 可以查询相关日志和 trace。

更多文章，请关注我们公众号 【Grafana 爱好者】。

发布于: 2022-11-01 | 阅读数: 927

可观测性 Observability

Grafana 爱好者
学习是从了解到使用再到输出的过程。 · 2018-04-27 加入
GrafanaFans 是由南京多位 Grafana 爱好者一起发起的 Grafana 开源产品学习小组，致力于 LGTM（Loki、Grafana、Tempo、Mimir）技术栈在国内的普及和应用，欢迎关注开源项目...

评论