什么是可观测模块

可观测模块（Insight）是以应用为中心、开箱即用的新一代云原生可观测性平台。 能够实时监控应用及资源，采集各项指标、日志及事件等数据用来分析应用健康状态，不仅提供告警能力以及全面、清晰、多维度数据可视化能力，兼容主流开源组件，而且提供快捷故障定位及一键监控诊断的能力。

可观测模块实现了指标、日志、链路的统一采集，支持对指标、日志进行多维度的告警并提供简洁明了的可视化管理界面。

模块指引

- : material-server:{ .lg .middle } **安装与升级**

可观测性模块包含 Insight 和 Insight Agent，后者需要部署在被观测的 Kubernetes 上。

  - 部署资源规划
  - Insight Agent 安装 与升级
  - Insihgt Agent 安装在 DCE 4 或 OpenShift
  - 大规模日志部署调整
  - 升级注意事项

- : :material-auto-fix:{ .lg .middle } **开始观测**

使用 OpenTelemetry 技术对您的应用程序进行观测。

  - 了解 OpenTelemetry，向 Insight 发送观测数据
  - 使用无侵入方式增强应用

- 针对 Java 应用观测
- 针对 Golang 应用观测
- 其他观测技术集成

基本概念

可观测性 (Insight) 有关的基本概念如下。

| 术语 | 英文 | 定义 |

: — | :——— | ———————— | :————————————————————— |

1 | 监控目标 | Target | 被监控的对象；系统会定时向监控点发起抓取任务，从中获取指标 |

2 | 指标 | Metric | 使用 open-metric 格式描述，衡量软件或硬件系统中某种属性的程度的标准 |

3 | 自定义指标 | Recording Rule | 一个被命名的 PromQL 表达式，这是将多个指标通过计算而得到的新指标，用来描述更加完整和复杂的系统状态 |

4 | 仪表盘 | Dashboard | 仪表盘是可视化管理的一种表现形式，即对数据、情报等状况一目了然的表现，它通过形象直观而又色彩适宜的各种视觉感知来展示信息。通过可视化图形展示平台的实时情况和 DCE 中所有的性能指标。 |

6 | 服务发现 | Service Discovery | 一个用于 Kubernetes 环境的服务发现配置，用于批量且自动地接入 Kubernetes 上的监控点 |

7 | Exporter | Exporter | 一个能够提供指标的服务，往往被理解为监控对象 |

8 | 告警规则 | Rule | 一个返回值是布尔值的 PromQL 表达式，它描述了指标或自定义指标是否处于阈值范围中，如果不满足将产生一条告警事件 |

9 | 告警消息 | Alert | 告警规则被触发时的记录信息，记录了告警规则、触发时间、当前

系统状态；同时将触发相应的动作，例如发送邮件 |

10 | 通知 | Notification | 由系统通过邮件等渠道发送给用户的告警事件信息 |

11 | PromQL | PromQL | Prometheus 系统所支持的查询语句 |

下载 DCE 5.0 安装 DCE 5.0 申请社区免费体验

功能列表

本页列出了可观测性 Insight 的功能特性，欢迎使用。

类别	子类	描述	社区版	标准版
仪表盘	平台组件监控	通过原生 Grafana 提供开源精选仪表盘，提供内置仪表盘支持对 etcd、APIServer 等组件进行监控	✓	✓
	集群资源监控	对集群、节点、命名空间等多维度提供监控。Grafana 使用的数据源支持查看多集群的数据。	✓	✓

类别	子类	描述	社区版	标准版
	核心组件监控	提供监控平台核心组件的仪表盘，实时检测组件运行状态	✓	✓
基础设施	多集群监控	提供多集群业务集中可观测管理员统一管理多集群告警，且满足集群、租户管理员数据隔离支持持久化集群的指标、日志数据。	✓	✓
	集群监控	提供对单个集群的监控概览，可查看该集群的运行状态、了解集群的资源使用情况，以及当前集群正在发生的告警	✓	✓
	节点监控	支持查看节点运	✓	✓

类别	子类	描述	社区版	标准版
		行状态等，并了解该节点的 CPU、内存、网络等资源变化情况		
	命名空间监控	支持查看命名空间中运行的资源数量统计，以及命名空间中容器组使用的 CPU、内存量的总和。	✓	✓
	容器监控	支持对无状态负载、守护进程、容器组等资源进行监控，可以监控该工作负载的运行状态，可查看正在告警的数量以及 CPU、内存等资源消耗的变化趋势图	✓	✓

类别	子类	描述	社区版	标准版
	事件	支持查看集群中产生的 Kubernetes 事件记录集合，并支持按照事件类型、对象、原因等进行查询。	✓	✓
	拨测	基于黑盒监控定期通过 HTTP、TCP 等方式对目标进行连通性测试，快速发现正在发生的故障。	✓	✓
指标	普通查询	普通查询预订了基础指标，选择集群、类型、节点、指标名称等查询条件后可查询资源的变化趋势	✓	✓
	高级查询	支持通过原生	✓	✓

类别	子类	描述	社区版	标准版
		PromQL 语句，查询指标图表及数据详情		
日志	普通查询	可查询 Node、Pod、Depoyment、Statefulset 等日志，可查询单条日志的上下文内容支持按照关键字进行搜索默认按照时间排序，通过直方图可查询日志数量的变化趋势	✓	✓
	高级查询	支持原生 lucene 语法，快速查询目标日志	✓	✓
	日志上下文	点击单行日志右侧的图标可查看该行日志的上下	✓	✓

类别	子类	描述	社区版	标准版
		文信息。		
	日志下载	支持下载一段时间内的日志支持导出单条日志上下文的内容支持自定义日志下载的字段	✓	✓
链路追踪	服务拓扑	管理员可查看接入观测平台和链路采集的服务间的调用关系、健康状态，快速的故障定位可查看服务间请求的流量方向和关键指标可快速查看单个服务的实时吞吐量、请求数、请求延时和错误率		✓
	服务	可查看当前接入		✓

类别	子类	描述	社区版	标准版
		链路数据的服务列表，以及服务最近 15 分钟的吞吐率、错误率、请求延时 点击服务可查看所选服务最近 15 分钟的流量趋势以及该服务操作的聚合指标		
	调用链	默认查询所选服务最近 15 分钟中的所有请求以及请求状态、延时、Span 数等 点击列表后侧的图标，可查询该链路的相关容器日志和链路日志。		✓
告警中心	活动告警	提供直方图查看	✓	✓

类别	子类	描述	社区版	标准版
		告警时间的变化趋势支持查看所有正在告警的规则及详情		
	历史告警	可查询自动恢复或手动被解决后的所有告警	✓	✓
	告警规则	内置 100+ 告警规则，对集群组件、容器资源等提供预定义的告警规则管理员可创建全局告警规则，对已安装 insight-agent 的集群进行统一告警支持通过预定义指标创建告警规则支持通过编写 PromQL 语句创建告警规则	✓	✓

类别	子类	描述	社区版	标准版
		支持自定义阈值、持续时间及通知方式可自定义告警的级别，支持紧急、警告、提示三个等级		
	日志/事件告警	针对日志关键字、事件状态频次设置告警规则		✓ |
	通知配置	在通知配置页面，可以配置通过邮件组、企业微信、钉钉、Webhook 等方式向用户发送消息支持同时通知到多个告警对象	✓	✓
	消息模板	消息模板功能支持自定义消息模板的内容，并可邮件、企业微信、	✓	✓

类别	子类	描述	社区版	标准版
		钉钉、Webhook、短信的形式通知指定的对象		
	告警静默	通过配置静默规则，可以在指定时间段内不再接收告警通知。	✓	✓
	告警抑制	通过配置抑制规则，可以抑制或阻止与某些特定告警相关的其他告警通知。	✓	✓
	告警模板	支持告警模板，平台管理员创建告警模板及规则；业务侧可以直接使用告警模板创建告警策略。	✓	✓
日志采集和查询	统一日志采集	统一采集节点、容器、容器内、	✓	✓

类别	子类	描述	社区版	标准版
		k8s 事件的日志数据采集全局管理平台的审计操作，默认不开启采集 k8s 审计日志		
	日志持久化存储	日志可标注输出到 Elasticsearch 等中间件进行持久化	✓	✓
指标采集	指标数据采集	支持通过使用 ServiceMonitor 自行定义 Pod 发现的 Namespace 范围以及通过 matchLabel 来选择监听的 Service	✓	✓
系统配置	系统配置	系统配置展示指标、日志、链路默认的保存时长	✓	✓

类别	子类	描述	社区版	标准版
		以及默认的 Apdex 阈值支持自定义修改指标、日志、链路数据的存储时间支持管理员配置服务拓扑渲染的阈值		
	系统组件	提供对可观测组件的统一监控，实时检测系统组件的健康状态	✓	✓

可观测性的优势

- 多集群管理

  通过全局服务集群统一存储指标、日志、链路数据，实现对多集群的统一监控，用户可以查询多集群数据。

- 开箱即用

  - 开源精选仪表盘：Insight 提供了多种开箱即用的预制监控仪表盘，您可以通过内置仪表盘实现对集群、节点、工作负载等组件的全面监控。
  - 内置告警规则：Insight 提供了内置的告警规则，您可以在免配置的情况下实

现对集群资源、系统组件等基础指标实现开箱即用的监控。

- 高可用性
  - 轻量级 Agent，支持通过 Helm 一键式安装。
  - 数据存储组件支持多副本，保证数据的高可用。
- 开源兼容
  - 兼容标准开源 Prometheus，支持原生的 PromQL 查询指标的数据。
  - 兼容标准开源 **Prometheus.yaml** 采集规则配置文件、适合自定义 Kubernetes 内监控采集规则 ServiceMonitor。
  - 遵循开源 OpenTelemetry 规范，支持 Jaeger 的链路数据接入。

应用场景

多集群数据统一采集及观测

痛点

- 云上云下及多云环境系统无法统一管理，出现故障要逐个排查，导致运维效率低且成本高。
- 不同数据分开采集，导致数据无法关联分析，排障难度高。

解决方案

- 支持 Helm 一键安装，支持图形化的配置管理，不限 Kubernetes 发行版，实现了 Insight on Any Kubernetes。
- 可观测性具备日志、指标、分布式链路追踪、事件日志等一站式采集和存储。全栈数据观测实现了数据采集、存储、分析、可视化、告警一体化。

- 实时监控各种维度资源，包括集群监控、节点监控、工作负载监控、服务监控等。

安装采集器

安装采集器

快速故障定位及排查

痛点

- 在复杂微服务架构中，监控配置的维护成本增加，服务故障的定位成本增加，监控的生效速度面临巨大挑战。
- 容器环境叠加微服务架构使得排障复杂，需要避免以单次异常为依据判断根因进行排障工作。

解决方案

- 根据链路数据绘制服务拓扑图，快速定位异常服务。
- 拓扑图中发现异常应用后，通过调用链一键下钻，故障根因清晰可见。
- 通过关联工作负载查询错误日志，快速解决故障。

安装采集器

安装采集器

可观测性权限说明

可观测性模块使用以下角色：

- Admin / Kpanda Owner
- Cluster Admin
- NS Admin / NS Editor
- NS Viewer

各角色所具备的权限如下：

workload[Workload cluster otel collector] - -> otel[Global cluster otel collector] - -> jaeger[Global cluster jaeger collector] - -> es[Elasticsearch cluster]
classDef plain fill: #ddd,stroke:#fff,stroke-width:1px,color:#000; classDef k8s fill: #326ce5,stroke:#fff,stroke-width:1px,color:#fff; classDef cluster fill: #fff,stroke:#bbb,stroke-width:1px,color:#326ce5;
class sdk,workload,otel,jaeger,es cluster

如上图所示，在任一步骤传输失败都会导致无法查询出链路数据。如果您在完成应用链路增强后发现没有链路数据，请执行以下步骤：

1. 使用 DCE 5.0 平台，进入 __可观测性__ ，选择左侧导航栏的 __仪表盘__ 。

   ![insight 入口](https://docs.daocloud.io/daocloud-docs-images/docs/insight/images/insight01.png)

2. 点击仪表盘标题 __概览__ 。

   ![概览](https://docs.daocloud.io/daocloud-docs-images/docs/insight/images/insight02.png)

3. 切换到 __insight-system__ -> __insight tracing debug__ 仪表盘。

   ![tracing debug](https://docs.daocloud.io/daocloud-docs-images/docs/insight/images/insighttrace01.png)

4. 可以看到该仪表盘由三个区块组成，分别负责监控不同集群、不同组件传输链路的数据情况。通过生成的时序图表，检查链路数据传输是否存在问题。

   - workload opentelemetry collector
   - global opentelemetry collector
   - global jaeger collector

   ![tracing debug](https://docs.daocloud.io/daocloud-docs-images/docs/insight/images/insighttrace02.png)

区块介绍

1. **workload opentelemetry collector**

   展示不同工作集群的 __opentelemetry collector__ 在接受 language probe/SDK 链路数据，发送聚合链路数据情况。可以通过左上角的 __Cluster__ 选择框选择所在的集群。

   ![tracing debug](https://docs.daocloud.io/daocloud-docs-images/docs/insight/images/insighttrace03.png)

   !!! note

根据这四张时序图，可以判断出该集群的 __opentelemetry collector__ 是否正常运行。

2. **global opentelemetry collector**

展示 __全局服务集群__ 的 __opentelemetry collector__ 在接收 __工作集群__ 中 __otel collector__ 链路数据以及发送聚合链路数据的情况。

![tracing debug](https://docs.daocloud.io/daocloud-docs-images/docs/insight/images/insighttrace04.png)

!!! note

__全局服务集群__ 的 __opentelemetry collector__ 还负责发送所有工作集群的[全局管理模块](../../ghippo/intro/index.md)的[审计日志](../../ghippo/user-guide/audit/audit-log.md)以及 Kubernetes 审计日志（默认不采集）到全局管理模块的 __audit server__ 组件。

3. **global jaeger collector**

展示 __全局服务集群__ 的 __jaeger collector__ 在接收 __全局服务集群__ 中 __otel collector__ 的数据，并发送链路数据到 [ElasticSearch 集群](../../middleware/elasticsearch/intro/index.md)的情况。

![tracing debug](https://docs.daocloud.io/daocloud-docs-images/docs/insight/images/insighttrace05.png)

使用 Insight 定位应用异常

本文将以 DCE 5.0 中举例，讲解如何通过 Insight 发现 DCE 5.0 中异常的组件并分析出组件异常的根因。

本文假设你已经了解 Insight 的产品功能或愿景。

拓扑图 — 从宏观察觉异常

随着企业对微服务架构的实践，企业中的服务数量可能会面临着数量多、调用复杂的情况，开发或运维人员很难理清服务之间的关系，
因此，我们提供了拓扑图监控的功能，我们可以通过拓扑图对当前系统中运行的微服务状况进行初步诊断。

如下图所示，我们通过拓扑图发现其中 __Insight-Server__ 这个节点的颜色为 __红色__ ，并将鼠标移到该节点上，
发现该节点的错误率为 __2.11%__ 。因此，我们希望查看更多细节去找到造成该服务错误率不

为 __0__ 的原因:

![01](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/01.png)

当然，我们也可以点击最顶部的服务名，进入到该服务的总览界面：

![02](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/02.png)

服务总览 — 具体分析的开始

当你需要根据服务的入口和出口流量分别分析的时候，你可以在右上角进行筛选切换，筛选数据之后，我们发现该服务有很多 __操作__
对应的错误率都不为 0. 此时，我们可以通过点击 __查看链路__ 对该 __操作__ 在这段时间产生的并记录下来的链路进行分析：

![03](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/03.png)

![04](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/04.png)

链路详情 — 找到错误根因，消灭它们

在链路列表中，我们可以通过界面直观地发现链路列表中存在着 __错误__ 的链路（上图中红框圈起来的），我们可以点击错误的链路查看链路详情，如下图所示：

![05](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/05.png)

在链路图中我们也可以一眼就发现链路的最后一条数据是处于 __错误__ 状态，将其右边 __Logs__ 展开，我们定位到了造成这次请求错误的原因：

![06](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/06.png)

根据上面的分析方法，我们也可以定位到其他 __操作__ 错误的链路：

![07](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/07.png)

![08](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/08.png)

![09](https: //docs.daocloud.io/daocloud-docs-images/docs/zh/docs/insight/images/find_root_cause/09.png)

接下来 — 你来分析！

ElasticSearch 数据塞满如何操作？

当 ElasticSearch 内存占满时，可以选择[扩容](#_1)或者[删除数据](#_2)来解决：

你可以运行如下命令查看 ES 节点的资源占比。

```bash
kubectl get pod -n mcamel-system | grep common-es-cluster-masters-es | awk '{print $1}' | xargs -I {} kubectl exec {} -n mcamel-system -c elasticsearch -- df -h | grep /usr/share/elasticsearch/data
```

扩容

在主机节点还有资源的情况下， **扩容** 是一种常见的方案，也就是提高 PVC 的容量。

1. 先运行以下命令获取 es-data-0 节点的 PVC 配置，请以实际的环境的 PVC 为准。

   ```
   kubectl edit -n mcamel-system pvc elasticsearch-data-mcamel-common-es-cluster-masters-es-data-0
   ```

2. 然后修改以下 storage 字段（需要使用的存储类 SC 可以扩容）

   ```
   spec:
     accessModes:
       - ReadWriteOnce
     resources:
       requests:
         storage: 35Gi # (1)!
   ```

   1. 这个数值需调整

删除数据

当 ElasticSearch 内存占满时，你还可以删除 index 数据释放资源。

你可以参考以下步骤进入 Kibana 页面，手动执行删除操作。

1. 首先明确 Kibana Pod 是否存在并且正常运行：

```
kubectl get po -n mcamel-system |grep mcamel-common-es-cluster-masters-kb
```

2. 若不存在，则手动设置 replica 为 1，并且等待服务正常运行；若存在，则跳过该步骤

```
kubectl scale -n mcamel-system deployment mcamel-common-es-cluster-masters-kb --replicas 1
```

3. 修改 Kibana 的 Service 为 NodePort 暴露访问方式

```
kubectl patch svc -n mcamel-system mcamel-common-es-cluster-masters-kb-http -p '{"spec":{"type":"NodePort"}}'

# 修改完成后查看 NodePort。此例的端口为 30128，则访问方式为 https://{集群中的节点 IP}:30128
[root@insight-master1 ~]# kubectl get svc -n mcamel-system |grep mcamel-common-es-cluster-masters-kb-http
mcamel-common-es-cluster-masters-kb-http NodePort 10.233.51.174 <none> 5601:30128/TCP 108m
```

4. 获取 ElasticSearch 的 Secret，用于登录 Kibana（用户名为 elastic）

```
kubectl get secrets -n mcamel-system mcamel-common-es-cluster-masters-es-elastic-user -o jsonpath="{.data.elastic}" |base64 -d
```

5. 进入 **Kibana** -> **Stack Management** -> **Index Management** ，打开 **Include hidden indices** 选项，即可见所有的 index。 根据 index 的序号大小，保留序号大的 index，删除序号小的 index。

容器日志黑名单

具体配置方式如下：

1. 对于任意一个不需要采集容器日志的 Pod，在 Pod 的 annotation 中添加 insight.opentelemetry.io/log-ignore: "true" 来指定不需要采集的容器日志，例如：

```
apiVersion: apps/v1
kind: Pod
metadata:
  name: log-generator
spec:
```

```
    selector:
      matchLabels:
        app.kubernetes.io/name: log-generator
    replicas: 1
    template:
      metadata:
        labels:
          app.kubernetes.io/name: log-generator
        annotations:
          insight.opentelemetry.io/log-ignore: "true"
      spec:
        containers:
          - name: nginx
            image: banzaicloud/log-generator:0.3.2
```

2. 重启 Pod，等待 Pod 恢复运行状态之后，Fluenbit 将不再采集这个 Pod 内的容器的日志。

Prometheus 集群标签配置

本文说明如何修改 Prometheus(CR) 为监控指标添加集群标识标签（cluster_name），以提高指标、告警消息的可读性。

方式一：通过 Prometheus 修改

1. 在命令行执行以下语句找到 Prometheus(CR)

   ```
   kubectl get prometheus -n insight-system
   ```

   预期输出示例：

   ```
   NAME                                    VERSION    DESIRED   READY   RECONCILED   AVAILABLE   AGE
   insight-agent-kube-prometh-prometheus   v2.44.0    1         1       True         True        73d
   ```

2. 编辑 Prometheus CR，在 spec.externalLabels 参数中增加 cluster_name，作为在容器管理中注册的名字。

```
apiVersion: monitoring.coreos.com/v1
    kind: Prometheus
    metadata:
      name: insight-agent-kube-prometh-prometheus
      namespace: insight-system
    spec:
      externalLabels:
        cluster: deb65d24-1090-40cf-b5e6-489fac2f2c1b
+       cluster_name: kpanda-global-cluster
```

方式二：通过 Helm 修改（推荐）

1. 建议通过 Helm 来更新 Prometheus CR，以避免在升级过程中丢失这些配置。编辑 values.yaml 里对应的参数：

```
kube-prometheus-stack:
    prometheus:
      prometheusSpec:
        externalLabels:
          cluster: '{{ (lookup "v1" "Namespace" "" "kube-system").metadata.uid }}'
+         cluster_name: 'kpanda-global-cluster'
```

2. 可以通过 --set 参数来设置

```
--set kube-prometheus-stack.prometheus.prometheusSpec.externalLabels.cluster_name='kpanda-global-cluster'
```

补充说明

在容器管理 v0.27 版本之后，集群的名字也会标记在 kpanda-system 的 namespace 中，记录在 kpanda.io/cluster-name 的标签中。

```
kubectl get ns kpanda-system -o yaml
apiVersion: v1
kind: Namespace
metadata:
  finalizers:
  - kpanda.io/kpanda-system
  labels:
    kpanda.io/cluster-name: kpanda-global-cluster
    kpanda.io/kube-system-id: deb65d24-1090-40cf-b5e6-489fac2f2c1b
    kubernetes.io/metadata.name: kpanda-system
```

```
    name: kpanda-system
  name: kpanda-system
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

Insight 告警通知开启 v1alpha2

Insight 引入了新的模板体系，新的模板体系在渲染的数据结构上做了调整，因此和旧版本（v1alpha1）存在兼容性的问题。 为了避免产生兼容性的问题，默认不启用。用户需要主动开启。 需要特别注意的是，开启时，需要立即迁移模板到 v1alpha2 语法，否则将产生“使用已保存的原消息模板无法正常发送告警消息”错误的问题。（运维实践：记住备份旧模板数据）

本文接下来将介绍：

- 开启新模板
- 迁移至新模板

启用 v1alpha2

方法 1：(推荐) 通过 helm 命令 upgrade

1. 在 Helm upgrade 的执行命令中增加如下参数：

   --set server.alerting.notifyTemplate.version="v1alpha2"

2. 除 #1 之外亦可编辑 helm 的 values 文件，如下：

```
server:
      alerting:
        notifyTemplate:
-         version: v1alpha1
+         version: v1alpha2
```

方法 2：临时调整配置文件（configmap）

1. 编辑 insight-server 的配置文件（configmap）insight-server-config，调整配置文件如下：

```
alerting:
      notifyTemplate:
  -     version: v1alpha1
  +     version: v1alpha2
```

2. 编辑保存之后，重启 insight-server 即可。

模板迁移

模板的迁移主要是因为数据结构从 AmAlert 调整为 AMHookRequest，新的模板语句可以渲染和展示更多的有用信息。

数据架构

这是 v1alpha2 的模板里使用的结构：

```
type AMHookRequest struct {
    Version           string            `protobuf:"bytes,1,opt,name=version,proto3" json:"version,omitempty"`
    GroupKey          string            `protobuf:"bytes,2,opt,name=groupKey,proto3" json:"groupKey,omitempty"`
    Status            string            `protobuf:"bytes,3,opt,name=status,proto3" json:"status,omitempty"`
    Receiver          string            `protobuf:"bytes,4,opt,name=receiver,proto3" json:"receiver,omitempty"`
    GroupLabels       map[string]string `protobuf:"bytes,5,rep,name=groupLabels,proto3" json:"groupLabels,omitempty" protobuf_key:"bytes,1,opt,name=key,proto3" protobuf_val:"bytes,2,opt,name=value,proto3"`
    CommonLabels      map[string]string `protobuf:"bytes,6,rep,name=commonLabels,proto3" json:"commonLabels,omitempty" protobuf_key:"bytes,1,opt,name=key,proto3" protobuf_val:"bytes,2,opt,name=value,proto3"`
    CommonAnnotations map[string]string `protobuf:"bytes,7,rep,name=commonAnnotations,proto3" json:"commonAnnotations,omitempty" protobuf_key:"bytes,1,opt,name=key,proto3" protobuf_val:"bytes,2,opt,name=value,proto3"`
    ExternalURL       string            `protobuf:"bytes,8,opt,name=externalURL,proto3" json:"externalURL,omitempty"`
```

```
    Alerts          []*AmAlert      `protobuf:"bytes,9,rep,name=alerts,proto3" json:"alerts,omitempty"`
    TruncatedAlerts int64           `protobuf:"varint,10,opt,name=truncatedAlerts,proto3" json:"truncatedAlerts,omitempty"`
}
```

这是 v1alpha1 的模板里使用的结构：

```
type AmAlert struct {
    Status          string          `protobuf:"bytes,1,opt,name=status,proto3" json:"status,omitempty"`
    Labels          map[string]string `protobuf:"bytes,2,rep,name=labels,proto3" json:"labels,omitempty" protobuf_key:"bytes,1,opt,name=key,proto3" protobuf_val:"bytes,2,opt,name=value,proto3"`
    Annotations     map[string]string `protobuf:"bytes,3,rep,name=annotations,proto3" json:"annotations,omitempty" protobuf_key:"bytes,1,opt,name=key,proto3" protobuf_val:"bytes,2,opt,name=value,proto3"`
    StartsAt        string          `protobuf:"bytes,4,opt,name=startsAt,proto3" json:"startsAt,omitempty"`
    EndsAt          string          `protobuf:"bytes,5,opt,name=endsAt,proto3" json:"endsAt,omitempty"`
    GeneratorURL    string          `protobuf:"bytes,6,opt,name=generatorURL,proto3" json:"generatorURL,omitempty"`
    Fingerprint     string          `protobuf:"bytes,7,opt,name=fingerprint,proto3" json:"fingerprint,omitempty"`
}
```

可以注意到，最大的差别是，旧结构仅仅是新结构的 .Alerts 字段，新结构提供更丰富的 CommonLabels，CommonAnnotations 等信息。

示例

以邮件通知模板为例，下面是新旧模板的 diff。可以看到，第二行新增了 {{range .Alerts}} 语法，只需将原有模板包裹在 range 关键字中即可。 需要特别注意的是，邮件的标题和正文 **共享相同的数据结构**，不再进行特殊处理，从而降低心智负担。

```
+<b style="font-weight: bold">[{{ .Alerts | len -}}] {{.Status}}</b><br />
+{{range .Alerts}}
ruleName: {{ .Labels.alertname }} <br />
groupName: {{ .Labels.alertgroup }} <br />
severity: {{ .Labels.severity }} <br />
cluster: {{ .Labels.cluster | toClusterName }} <br />
{{if .Labels.namespace }} namespace: {{ .Labels.namespace }} <br /> {{ end }}
```

{{if .Labels.node }} node: {{ .Labels.node }}
 {{ end }}
targetType: {{ .Labels.target_type }}
{{if .Labels.target }} target: {{ .Labels.target }}
 {{ end }}
value: {{ .Annotations.value }}
startsAt: {{ .StartsAt }}
{{if ne "0001-01-01T00:00:00Z" .EndsAt }} EndsAt: {{ .EndsAt }}
 {{ end }}
description: {{ .Annotations.description }}
+
+{{end}}

旧模板的写法：

[{{ .status }}] [{{ .severity }}] alert: {{ .alertname }}

新模板里，可以注意到 severity 调整为 CommonLabels.severity，这才是原始的无额外处理的数据结构。

[{{ .Status }}] [{{ .CommonLabels.severity }}] alert: {{ .CommonLabels.alertname }}

更多示例

飞书、钉钉和企业微信的通知模板。

+[{{ .Alerts | len -}}] {{.Status}}
+{{range .Alerts}}
Rule Name: {{ .Labels.alertname }}
Group Name: {{ .Labels.alertgroup }}
Severity: {{ .Labels.severity }}
Cluster: {{ .Labels.cluster | toClusterName }}
{{if .Labels.namespace }}Namespace: {{ .Labels.namespace }}
{{ end }}{{if .Labels.node }}Node: {{ .Labels.node }}
{{ end }}Target Type: {{ .Labels.target_type }}
{{if .Labels.target }}Target: {{ .Labels.target }}
{{ end }}Value: {{ .Annotations.value }}
Starts At: {{ .StartsAt }}
{{if ne "0001-01-01T00:00:00Z" .EndsAt }}Ends At: {{ .EndsAt }}
{{ end }}Description: {{ .Annotations.description }}
+{{end}}