

## 31 | 日志：如何搭建轻量云原生业务日志系统？

2023-02-17 王伟 来自北京

《云原生架构与GitOps实战》

[课程介绍 >](#)



讲述：王伟

时长 16:31 大小 15.08M



你好，我是王伟。

上一节课，我们学习了如何迅速判断业务可用性以及当故障出现时应该如何排查问题。其中，在可观测性的日志方面，我提到了使用 `kubecttl` 来查看 `Pod` 日志的方法。不过，在生产环境下，我们面临的情况会更加复杂。

比如，对于同一个工作负载，它可能有几十上百个 `Pod` 实例，当我们需要定位具体某一个请求的日志时，它可能出现在任何一个 `Pod` 里。很显然，查看每一个 `Pod` 的日志是不切实际的。其次，即便我们能找到对应的 `Pod`，在查询日志时也很难进行条件搜索。最后，`Kubernetes` 并不能持久化存储 `Pod` 日志，这意味着一段时间后我们将无法查询到以前的日志信息。

此外，有时候我们还希望通过日志进一步了解系统的性能表现。例如：出现频率最高的错误是什么？错误日志的增长情况是怎样的？

日志系统作为应用可观测性的三大支柱之一，能够为我们提供软件全生命周期的事件和错误记录。因此，建立中心化的日志存储和查询系统能够很好地帮助我们解决这些问题。

所以，这节课，我将带你学习如何构建可观测性的日志体系，并通过 **Grafana Loki** 来搭建轻量的日志系统。在使用方面，我还会简单介绍 **Loki** 查询日志的 **LogQL** 语法，带你从零上手 **Loki**。

## 安装 Loki

完整的 **Loki** 日志系统包含下面三个组件。

1. **Loki**: 核心组件，负责日志存储和处理查询。
2. **Promtail**: 日志收集工具，负责收集 **Pod** 的日志并发送给 **Loki**。
3. **Grafana**: **UI Dashboard**，用于查询日志。

为了一次性安装所有的组件，**Grafana** 封装了 **Loki-Stack Helm Chart**，它包含这三个组件，并且已经做了初始化的配置，我推荐你用 **Helm** 的方式来安装。

首先，执行下面的命令来添加 **Helm Repo**。

```
1 $ helm repo add grafana https://grafana.github.io/helm-charts
```

 复制代码

然后，使用 **Helm** 来安装 **Loki**。

```
1 $ helm upgrade --install loki --namespace=loki-stack grafana/loki-stack --creat
```

 复制代码

在上面的安装命令中，我们指定安装的命名空间为 **loki-stack**，并且使用了 **--set** 参数开启了 **Grafana** 组件，镜像版本设置为了 **9.3.2**。

接下来，等待 **Loki** 所有组件处于 **Ready** 状态。

```

1 $ kubectl wait --for=condition=Ready pods --all -n loki-stack --timeout=300s
2 pod/loki-0 condition met
3 pod/loki-grafana-6f54cd8746-z5pmh condition met
4 pod/loki-promtail-qzcwj condition met

```

到这里，Loki 就安装好了。我们来继续看一下这三个组件具体的部署方式，你可以通过 `kubectl get all` 来查看 loki-stack 命名空间下的工作负载。

```

1 $ kubectl get all -n loki-stack
2 .....
3 NAME                                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
4 daemonset.apps/loki-promtail        1         1         1       1             1
5
6 NAME                                READY     UP-TO-DATE   AVAILABLE   AGE
7 deployment.apps/loki-grafana        1/1       1             1           22h
8
9 NAME                                DESIRED   CURRENT   READY   AGE
10 replicaset.apps/loki-grafana-6f54cd8746 1         1         1       22h
11 replicaset.apps/loki-grafana-7cbbfb8f9b 0         0         0       22h
12
13 NAME                                READY     AGE
14 statefulset.apps/loki                1/1       22h

```

从返回结果可以看出，日志采集代理 **Promtail** 组件是以 **Daemonset** 的方式来部署的，UI 界面 **Grafana** 是以 **Deployment** 的方式部署的，核心的 **Loki** 组件因为需要持久化存储，所以是通过有状态的 **Statefulset** 工作负载来部署的。这三个组件的工作原理我会在后面的原理解析部分详细介绍。

## 访问 Grafana

**Grafana** 是我们日常查询日志的入口，在访问前我们需要先从 **Secret** 对象中获取登录密码。

```

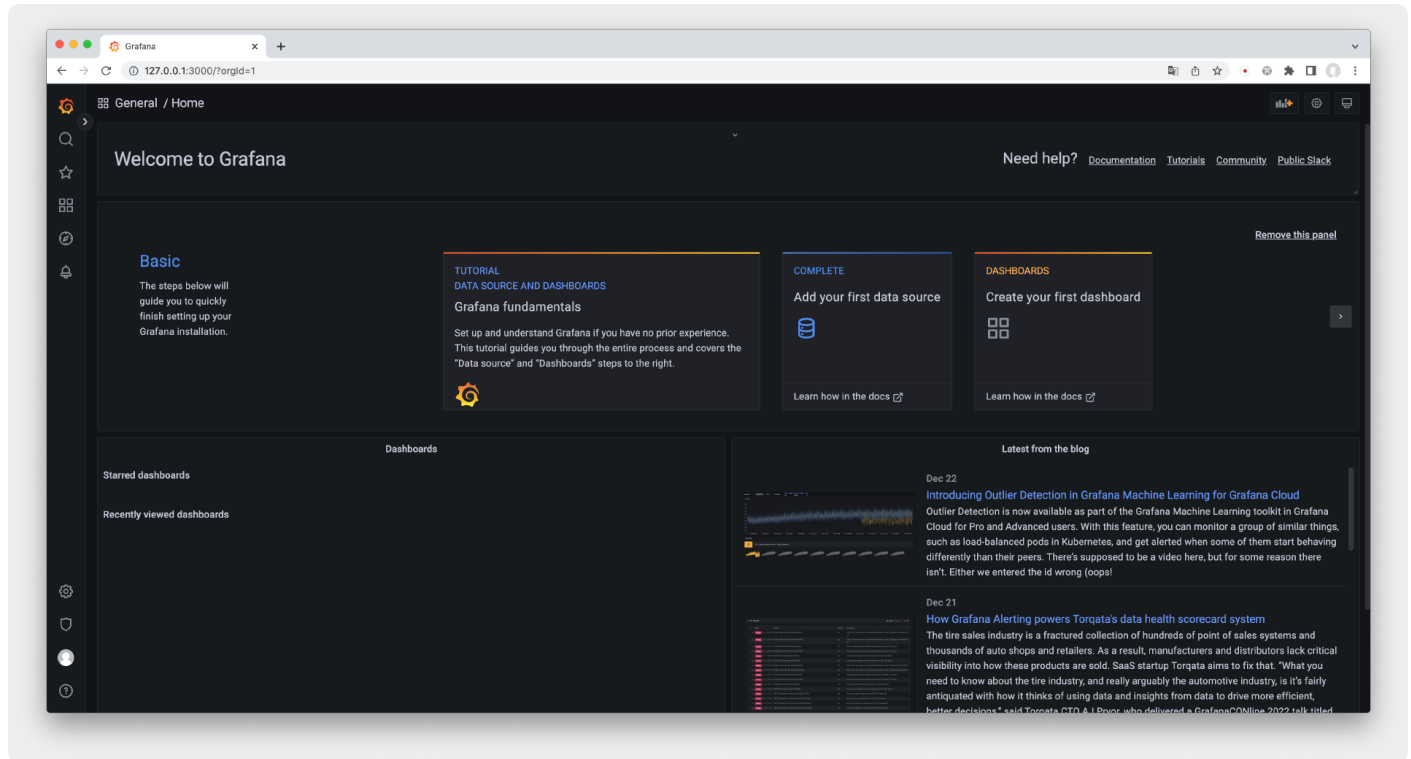
1 $ kubectl get secret --namespace loki-stack loki-grafana -o jsonpath="{.data.ad
2 R1JTMgYLrmFBq_lKCs9PWH6rk0LmhhHlobaZLdHU7

```

在生产环境下，我推荐你配置 **Ingress** 策略，并通过 **Ingress-Nginx** 网关来访问 **Grafana**。实验阶段则可以通过端口转发的方式进行访问。

```
1 $ kubectl port-forward --namespace loki-stack service/loki-grafana 3000:80
```

下一步使用浏览器访问 <http://127.0.0.1:3000>，输入用户名 **admin** 和上面获取的密码，登录后，你应该能看到 **Grafana** 的界面。



## 部署示例应用

接下来，为了介绍 **Loki** 的使用方法，我们首先需要让集群内产生一些日志，你需要先部署我提前写好的示例应用。

```
1 $ kubectl apply -f https://ghproxy.com/https://raw.githubusercontent.com/lyzhan
2 deployment.apps/log-example created
3 service/log-example created
4 ingress.networking.k8s.io/log-example created
```

在这个示例应用中，由于我配置了存活探针，所以 **Pod** 每隔 **10** 秒钟会在标准输出中打印日志信息。接下来我们尝试通过 **kubectl logs** 命令来查看 **Pod** 的日志信息。

```
1 $ kubectl logs -l app=log-example
```

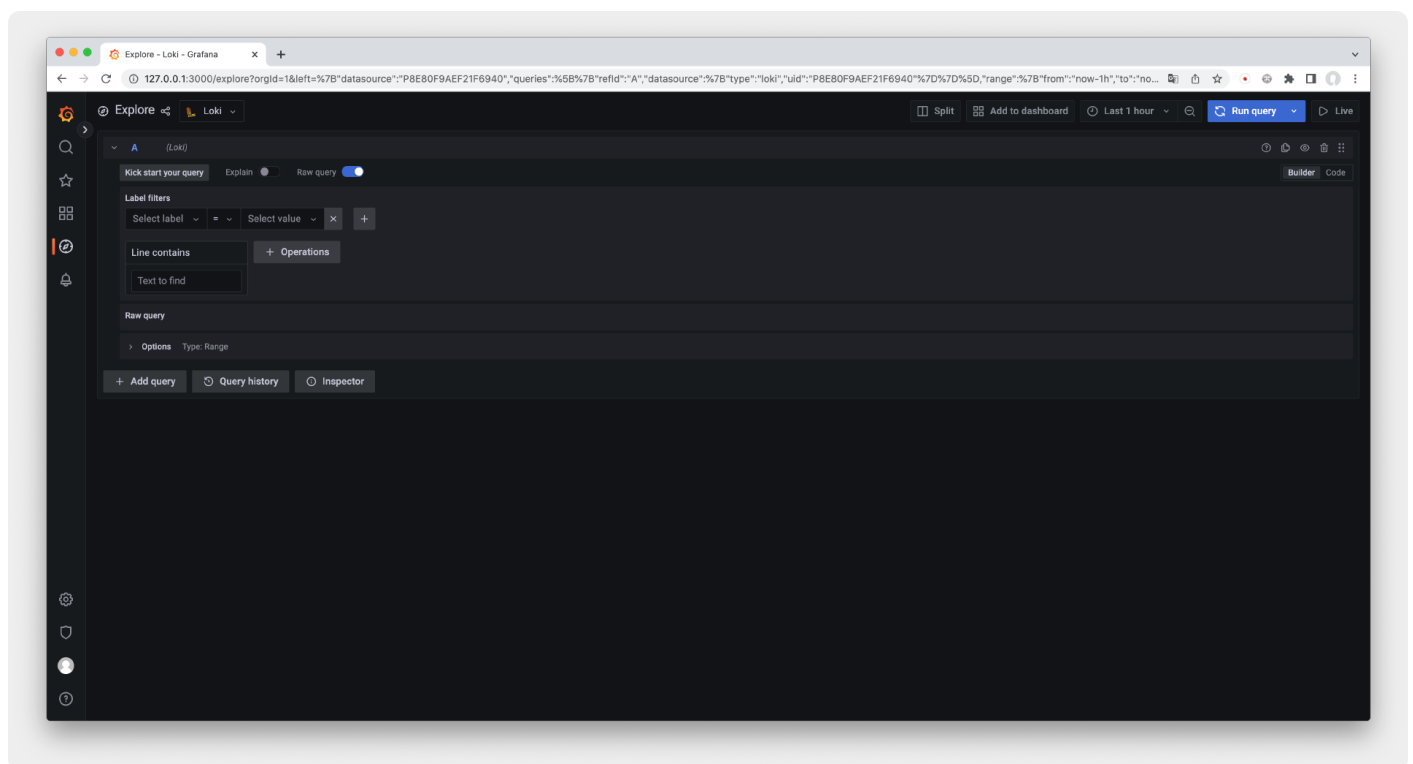
```
2 time="2022-12-23T02:55:52Z" level=info msg="request completed" duration="63.233
3 {"duration":63233,"level":"info","method":"GET","msg":"request completed","size
```

从返回结果我们会发现，示例应用输出了两种格式的日志信息，一种是**标准的 logfmt 格式**，另一种是 **JSON 格式**，这两种日志格式在生产环境下是最常见的。在日志内容中，`duration` 表示请求处理的时间，`method` 表示请求方式，`size` 表示返回内容的大小，`status` 表示返回的状态码，`uri` 表示请求的路径。

## 查询日志

在生产环境下，通过命令行来查询日志并不是一个好的实践。接下来我们尝试使用 **Loki** 来查询 **Pod** 的日志信息。

在进入 **Grafana** 之后，点击左侧的“**Explore**”进入日志查询界面，如下图所示。



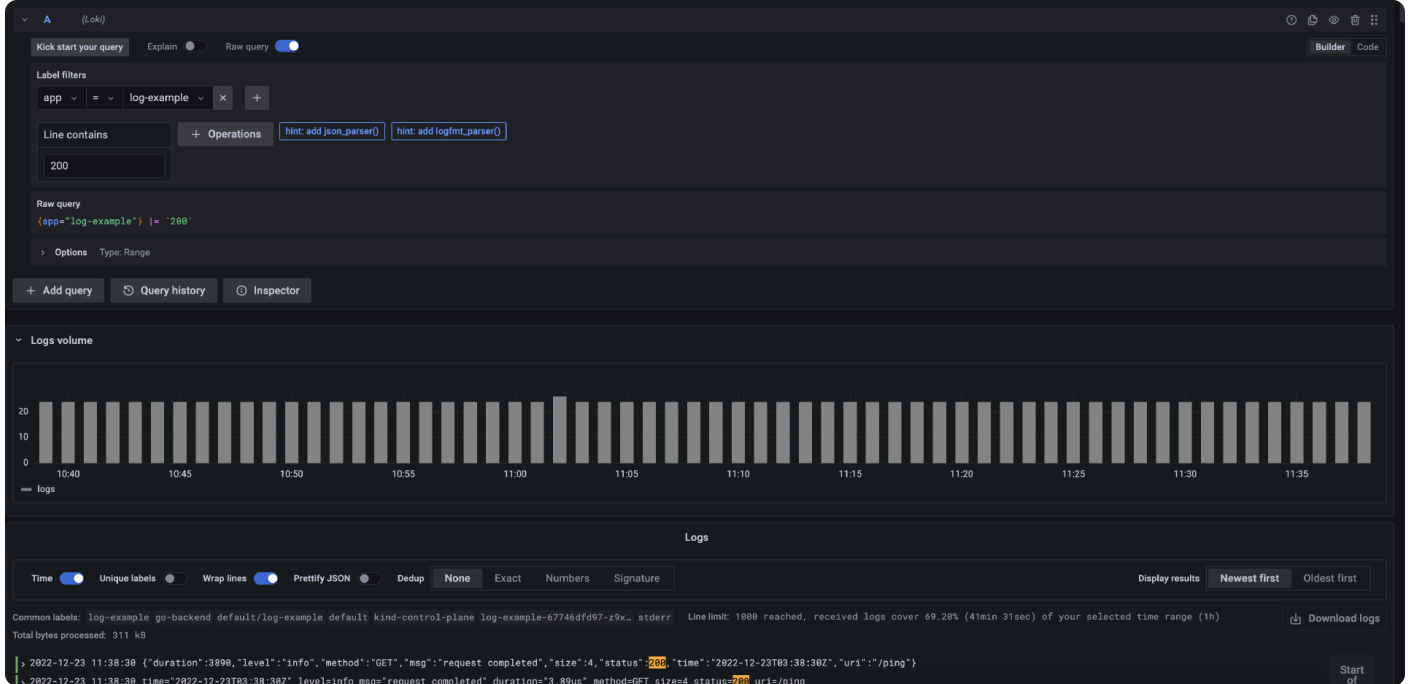
在一般情况下，查询 **Loki** 的日志信息需要编写 **LogQL**，它是 **Loki** 用来查询日志的一种特殊语法，学习的门槛也比较高。

为了解决这个问题，我们之前部署 **Grafana** 的时候，我就特意部署了 **9.0** 的版本，它带有图形化的查询语句构造器，我们可以通过表单的方式来构造查询语句。

[illegible]

在生产环境下，我们一般还会通过查找日志中的关键信息来定位错误的位置。在 **Loki** 中，要通过关键字来查找日志也非常简单，你可以在“**Line contains**”下方的输入框中输入关键字来查找日志。比如要查找包含字符串“**200**”的日志，如下图所示。

在生产环境下，我们一般还会通过查找日志中的关键信息来定位错误的位置。在 **Loki** 中，要通过关键字来查找日志也非常简单，你可以在“**Line contains**”下方的输入框中输入关键字来查找日志。比如要查找包含字符串“**200**”的日志，如下图所示。



这样，我们就能够通过关键字来定位日志信息了。

接下来，我们进一步分析刚才的查询过程。之前通过表单构造的 LogQL 是下面这样的。

 复制代码

```
1 {app="log-example"} |~ '200'
```

这个典型的 LogQL 实际上是由两部分组成的，前半部分大括号里的 `{app="log-example"}` 是日志流选择器，后半部分是日志管道操作，如下图所示。

**`{app="log-example"} |~ '200'`**

日志流选择器

日志管道操作

日志流选择器决定日志的来源，它可以通过 Label filters 标签过滤器来选择日志，它是一段包在大括号内的表达式。



日志管道操作是对日志的进一步处理，它以“|”符号作为管道操作符。例如我们可以通过关键字搜索日志，或者通过正则表达式来过滤日志。

如果你熟悉 Linux 系统，那么你完全可以把它和 Linux 中的管道操作进行类比来加深理解。比如在 Linux 下，要从一个 log 文件中查找一段包含某个关键字的日志，你可以像下面这么做。

```
1 $ cat log.log | grep "key_word"
```

 复制代码

在这段 Linux 命令中，cat 其实就是 Loki 中的日志流选择器，它决定了日志来源。而后面的管道操作符和 grep 则是对日志的进一步检索，这样看是不是更好理解了呢？

接下来，我先简单介绍一下日志流选择器中的 Label filters 标签过滤器。

## Label filters

Label filters 是标签过滤器，Loki 已经为我们内置了一些过滤器，我们可以用它来选择日志的来源，例如通过 Kubernetes Label 匹配一组 Pod 的日志，还可以匹配某个命名空间的日志，甚至是某个容器的日志。通常，你可以使用下面几种类型的标签过滤器。

1. 工作负载的 Label：通过 Label 匹配一组 Pod 作为日志来源，我们在上面的例子中就用到了它。
2. 容器：选择特定容器的日志作为来源。
3. 命名空间：选择特定的命名空间作为日志来源。
4. 节点：选择特定的节点作为日志来源。
5. Pod：选择特定的 Pod 作为日志来源。

在实际的业务场景中，我们查看日志的对象往往是明确的，所以一般会通过 **Label 或者容器名** 的方式来选择日志来源，这会减少检索日志的数量并加快查询速度。

## 日志管道操作

日志管道操作是以“|”符号为标识的，和 Linux 的管道操作符一样，在一次日志查询中，你可以使用多个日志管道组成链式操作。



日志管道操作内容非常多，所以我并不打算做完整的介绍。这里我从实际的场景出发，向你介绍一些常用的日志管道编写方法。

日志管道操作最简单的例子是通过关键字查找日志，也就是上面用到的匹配包含字符串“200”的日志，它的写法是：`|= 200`

这里的“=”号实际上是一个表达式，代表包含字符串。除了包含，你还可以使用下面几个表达式。

1. `!=`，日志不包含字符串。
2. `|~`，日志匹配正则表达式。
3. `!~`，日志不匹配正则表达式。

此外，对于结构化的日志信息，日志管道操作里还可以使用内置的日志解析器，例如 `logfmt` 和 `JSON` 解析器，它们可以进一步分析日志中的键值对。

## 日志字段值查询

在实际的项目中，我们一般会使用 `logfmt` 和 `JSON` 格式输出日志。`logfmt` 和 `JSON` 都是包含 `Key` 和 `Value` 的日志格式，例如下面就是一个典型的 `logfmt` 的例子。

 复制代码

```
1 time="2022-12-23T02:55:52Z" level=info msg="request completed" duration="63.233"
```

有时候，通过关键字搜索找到的日志可能并不是我们想要的。比如在上面的例子中，包含字符串“200”的日志并不代表请求返回状态码为 200 的日志，我们需要精确到日志 `status` 字段的值。其次，如果我们想找到所有请求返回状态码为非 200 的日志，关键字搜索就不能满足我们的要求了。这时候，我们需要一种更精确的查找方案：日志解析器。

例如，对于 `logfmt` 格式的日志，**要找到 `status` 字段值为 200 的日志**，我们可以像下面这么写 `LogQL`。

 复制代码

```
1 {app="log-example"} | logfmt | status = `200`
```

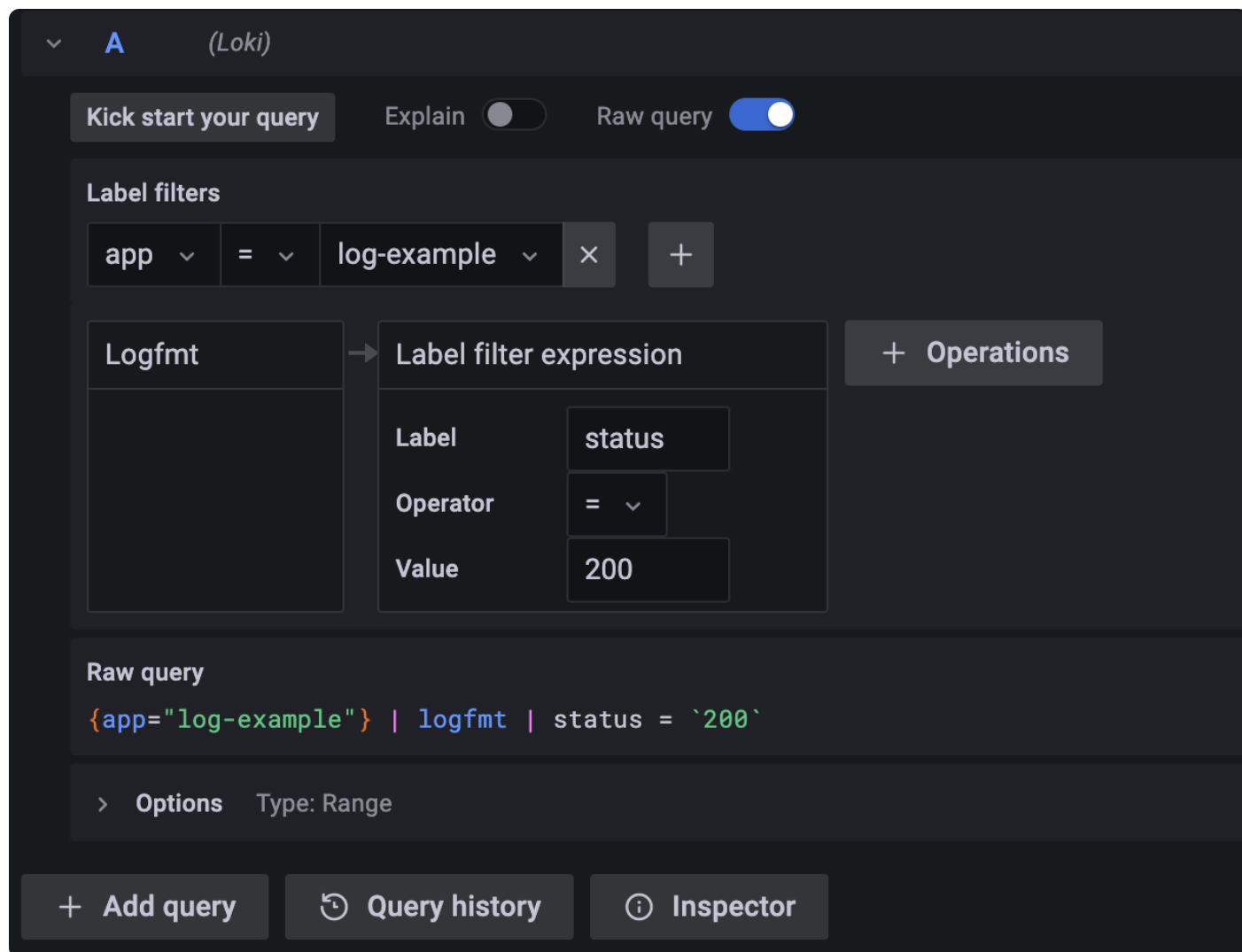
通过 `logfmt` 管道操作符，Loki 便能够帮我们解析日志内的所有字段。以此类推，你可以把 `status` 字段替换为 `uri` 字段来查找特定的请求路径。

对于 JSON 格式的日志，你可以使用 JSON 解析器，在上面的语句中，你只需要将 `logfmt` 替换为 `json` 即可。

 复制代码

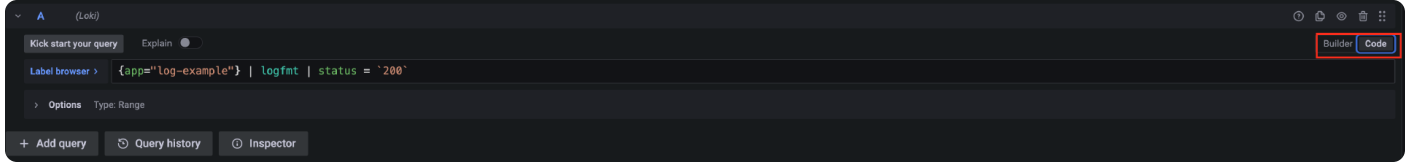
```
1 {app="log-example"} | json | status = `200`
```

你也可以通过 Grafana 表单来构造这个查询语句，如下图所示。



The screenshot shows the Grafana Loki query builder interface. At the top, there's a header with a dropdown menu showing 'A' and '(Loki)'. Below this, there are two tabs: 'Kick start your query' and 'Raw query', with 'Raw query' being the active tab. The 'Label filters' section shows a filter for 'app' with the value 'log-example'. Below this, there's a 'Logfmt' section with a 'Label filter expression' table. The table has three rows: 'Label' with the value 'status', 'Operator' with the value '=', and 'Value' with the value '200'. To the right of the table is a '+ Operations' button. Below the 'Logfmt' section, there's a 'Raw query' section showing the resulting query: `{app="log-example"} | logfmt | status = `200``. At the bottom, there are three buttons: '+ Add query', 'Query history', and 'Inspector'.

此外，你还可以将 Grafana 切换到输入 Code 模式（参考截图中红框部分），并将上面的 LogQL 粘贴到输入框内来查询，如下图所示。



## 常用 LogQL 例子

为了更好地帮助你使用 Loki，这里我再总结几个常用的 LogQL 例子，你可以根据下面这些例子举一反三。

- 查询状态码不等于 200 的日志。
- 查询状态码在 400 和 500 区间的日志。
- 查询接口返回时间超过 3us 的日志。
- 重新格式化输出日志。
- 分组统计 10s 内 Pod 的日志数量。
- 分组统计 1 分钟内的请求状态码数量。
- 统计 TP99 的请求耗时并分组。

由于示例应用输出了 logfmt 和 JSON 两种格式的日志，所以在所有的例子中，我都会通过管道操作 “status=” 来匹配 logfmt 的日志，以便把它和 JSON 格式的日志区分开。此外，你可以先把 Grafana 的查询界面切换到 **Code 模式**，这样就可以直接复制下面的语句来进行试验了。

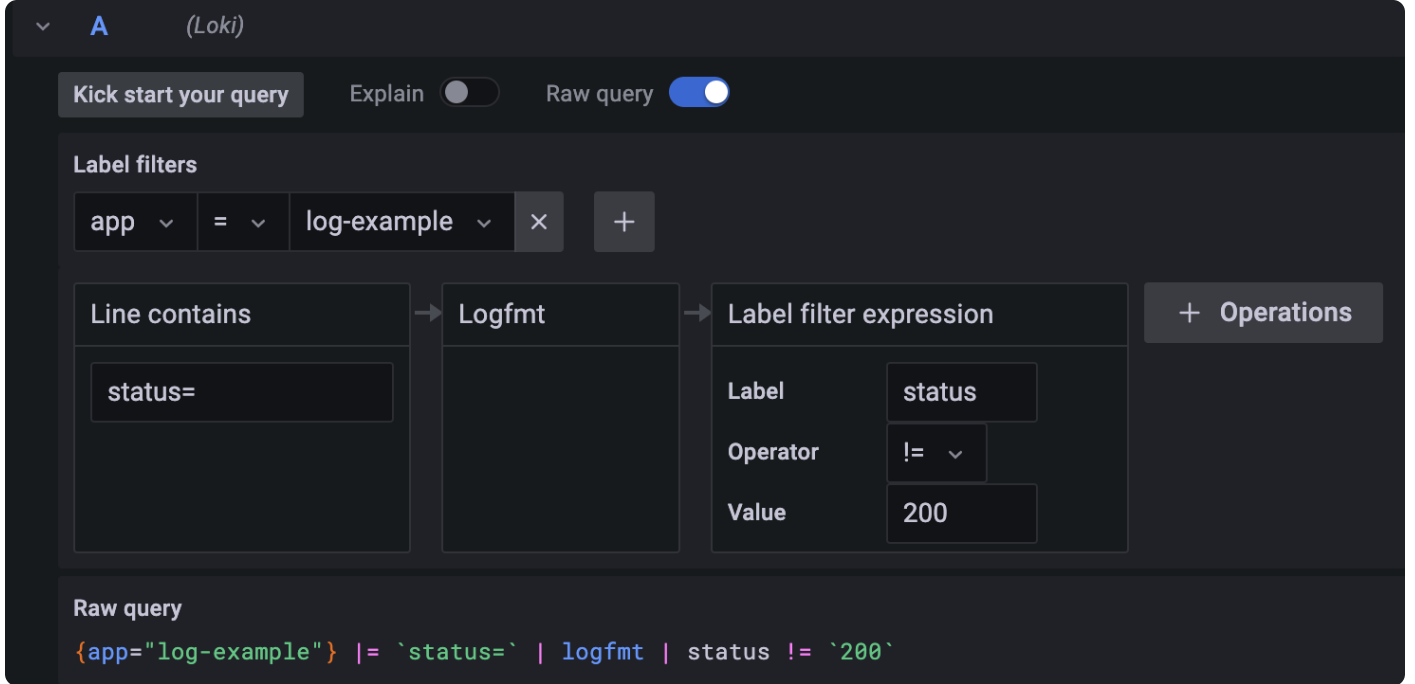
### 查询状态码不等于 200 的日志

在生产环境下，查询返回状态码不等于 200 的场景是比较常见的，以示例应用为例，你可以通过下面的 LogQL 来查询。

```
1 {app="log-example"} | `status=` | logfmt | status != `200`
```

复制代码

你也可以通过表单构建查询语句，如下图所示。



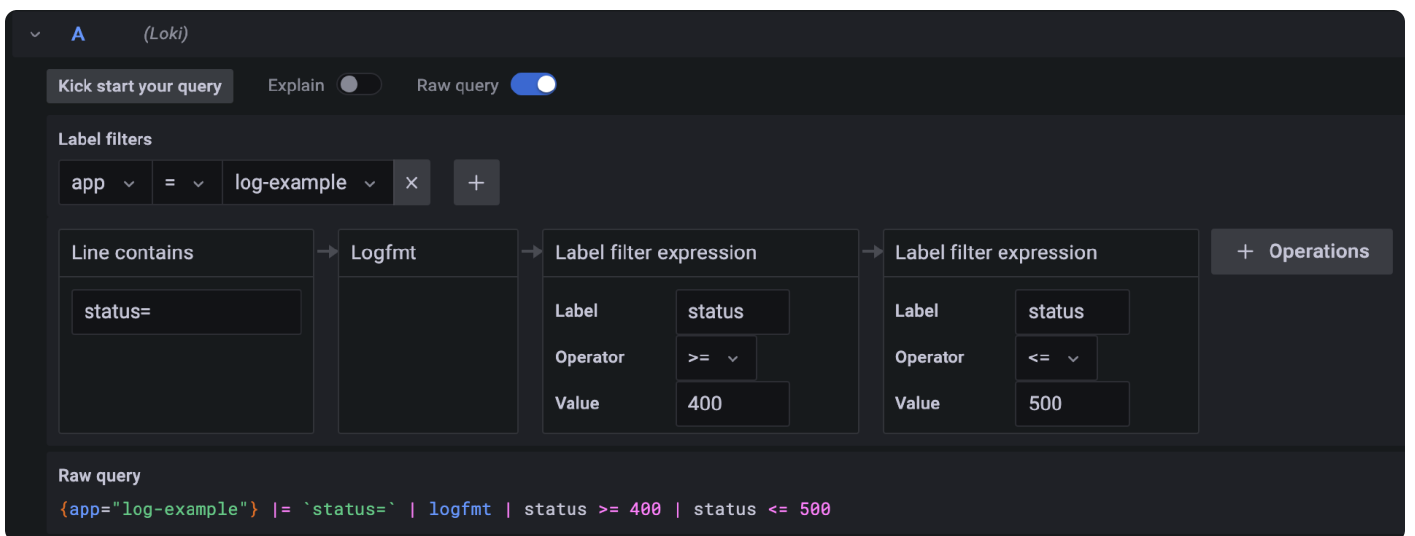
## 查询状态码在 400 和 500 区间的日志

在生产环境下，查询某个区间的日志有助于针对性地定位错误日志，你可以通过下面的 LogQL 来查询。

```
1 {app="log-example"} |= `status=` | logfmt | status >= 400 | status <= 500
```

复制代码

你还可以通过表单构建查询语句，如下图所示。



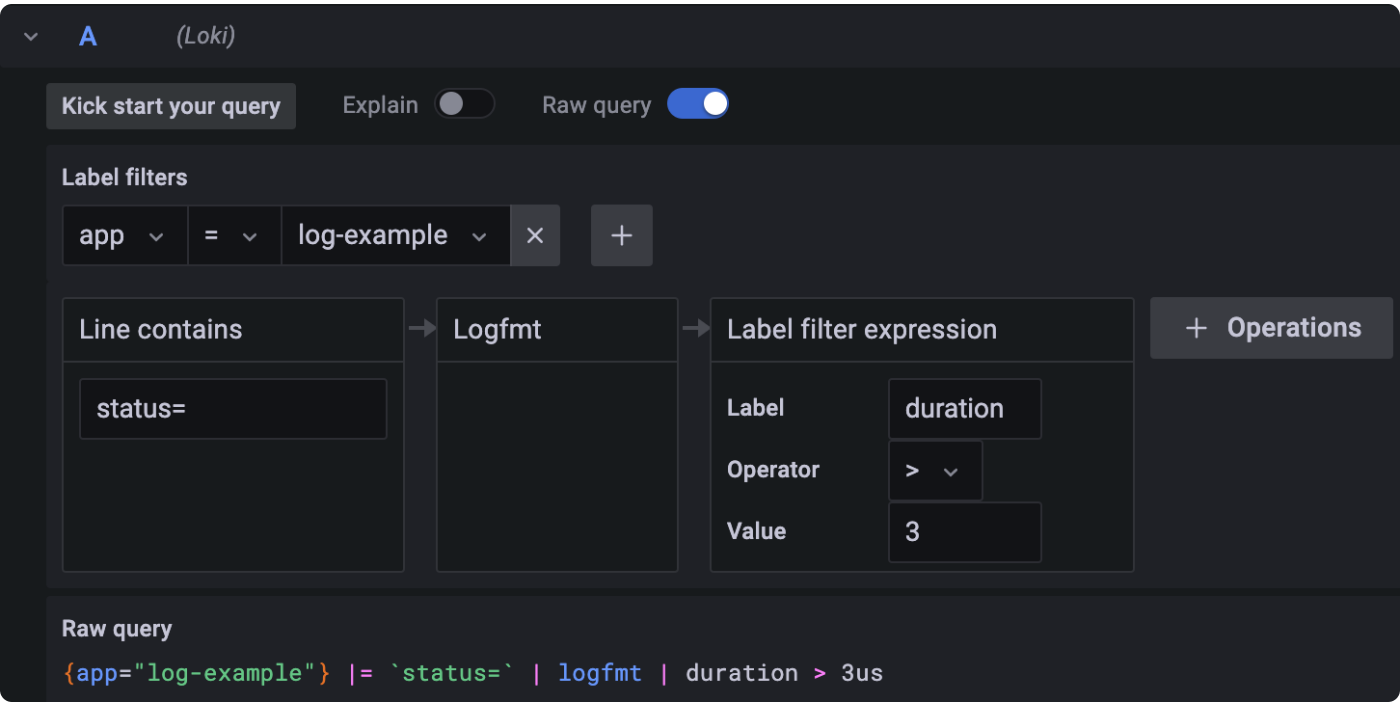
## 查询接口返回时间超过 3us 的日志

在生产环境下，我们经常有查找响应比较慢的接口的需求，你可以通过下面的 LogQL 来查询。

 复制代码

```
1 {app="log-example"} |= `status=` | logfmt | duration > 3us
```

你还可以通过表单构建查询语句，如下图所示。



通过这个例子我们会发现，Grafana 会自动推断日志值的类型，例如常见的字符串类型、数字类型和时间类型，并且支持比较操作符，使用非常方便。

## 重新格式化输出日志

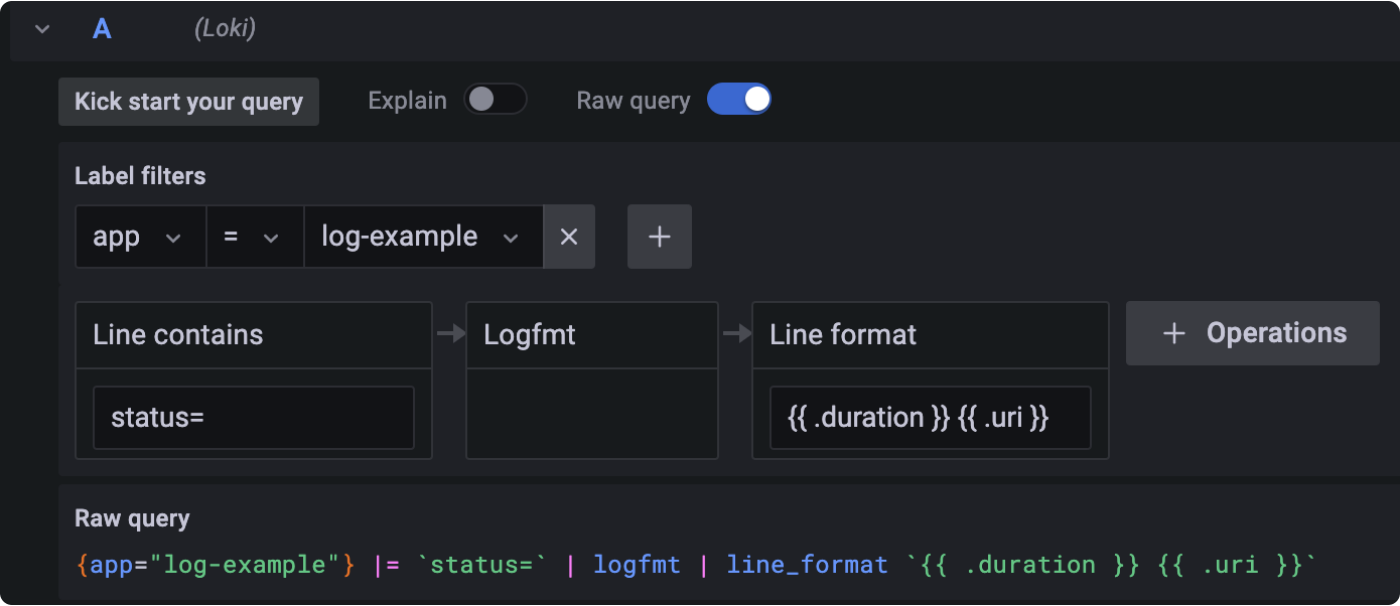
有时候，日志的输出内容比较多，当我们希望重点查看部分内容时，重新对日志输出进行格式化就显得比较重要了，你可以通过下面的 LogQL 来重新格式化输出日志。

 复制代码

```
1 {app="log-example"} |= `status=` | logfmt | line_format `{{ .duration }} {{ .ur
```

在这条语句中，line\_format 关键字后面指定了日志输出的内容，在这个例子中，查询日志将重新格式化为响应时间和请求路径输出，而不包含其他内容。

你还可以通过表单构建查询语句，如下图所示。



## 分组统计 10s 内 Pod 的日志数量

除了日志查询以外，Loki 还能够帮我们对日志进行度量统计。例如，要统计 10s 内接口返回时间超过 4us 的日志数量，并按 Pod 进行分组，你可以通过下面的 LogQL 来进行统计。

复制代码

```
1 sum by (pod) (count_over_time({app="log-example"} |= `status=` | logfmt | durat
```

在这条查询语句中，count\_over\_time 指的是统计给定范围内每个日志的条目数量，sum by 指的是对 Pod 进行分组。

运行查询后，你将得到下面的图表。



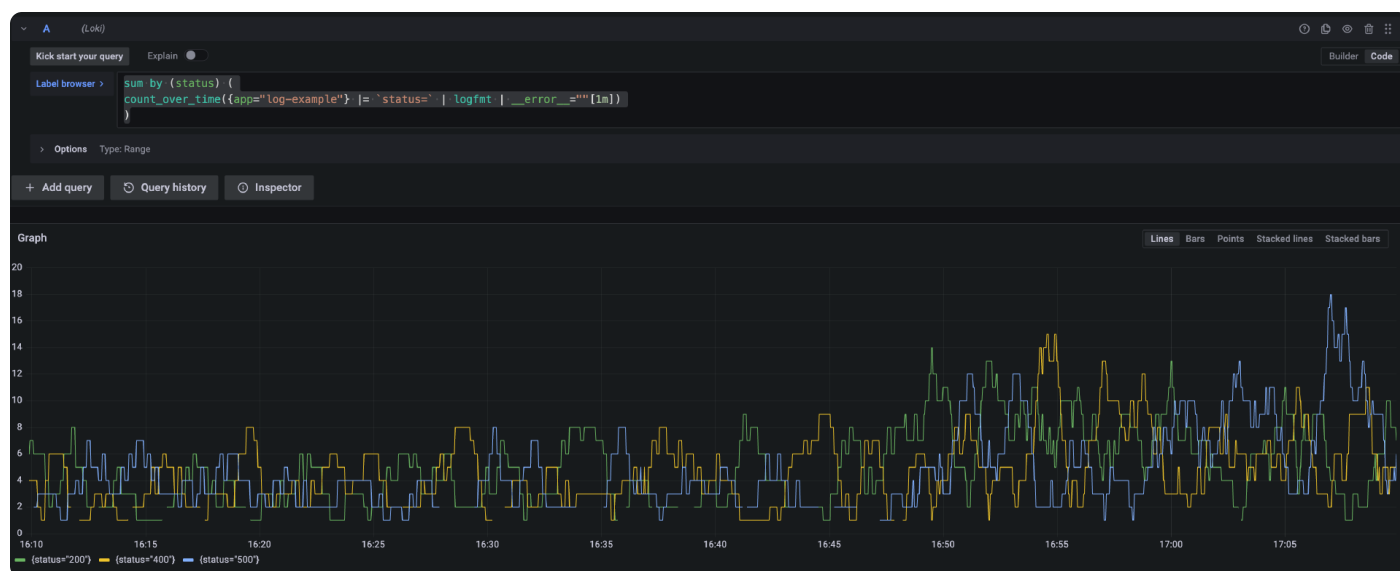
## 分组统计 1 分钟内的请求状态码数量

要对 1 分钟内的请求状态码进行分组的数量统计，你可以通过下面的 LogQL 来查询。

 复制代码

```
1 sum by (status) (  
2   count_over_time({app="log-example"} |= `status=` | logfmt | __error__=""[1m])  
3 )
```

运行查询语句后，你将得到下面的图表。



## 统计 TP99 的请求耗时并分组

要统计 1 分钟内 99% 的请求耗时并按照条件分组，你可以使用下面的 LogQL 来查询。

 复制代码

```
1 quantile_over_time(0.99,  
2   {app="log-example"}  
3   |= `status=` | logfmt | unwrap duration(duration) | __error__=""[1m]  
4 ) by (method,status)
```

quantile\_over\_time 是 Loki 的内置函数，用来指定间隔内值的分位数。

运行查询语句后，你将得到下面的图表。





从返回结果我们可以看到 99% 的请求在这段时间内的响应情况，这是一个非常好的系统评估指标。

不过，在生产环境下，我们一般不会直接从日志中观察 HTTP 请求的响应情况，而是会通过监测网关的性能数据来进行判断，这部分内容将会在下一节课进行详细介绍。

## 工作原理解析

相比较传统的 EFK 和 ELK 日志技术栈，Loki 在 Kubernetes 场景下更加轻量，同时，它还使用了 Grafana 作为查询界面，在构建了监控系统之后，可以在同一套界面查询日志和指标信息。

接下来我简单介绍一下 Loki 的工作原理，它的核心组件包含 Promtail 和 Loki。

## Promtail

在深入了解 Promtail 之前，我们首先需要知道一些 Kubernetes 日志的背景知识。

首先，当 Pod 打印输出日志时，会被 Kubernetes 存储在对应 Node 节点的 /var/log 目录的文件中，对于 Kind 集群，你可以通过下面的实验来验证这个过程。

首先，通过 docker ps 命令来查看本地正在运行中的容器。

 复制代码

```
1 $ docker ps
2 CONTAINER ID   IMAGE                                COMMAND                                CREATED    S
```

由于 Kind 是使用容器来模拟 Node 节点的，所以我们可以通过 `docker exec` 来进入到容器中，这就相当于我们进入了 Kubernetes 的 Node 节点。

 复制代码

```
1 $ docker exec -it a68175b2b097 bash
2 root@kind-control-plane:/#
```

接下来，进入 `/var/log/pods` 目录。

 复制代码

```
1 root@kind-control-plane:/# cd /var/log/pods
2 root@kind-control-plane:/var/log/pods# ls
3 default_log-example-67746dfd97-cq7lb_201ecbd1-d6e4-4efe-8bd8-c7620ef21f15
4 default_log-example-67746dfd97-f2h74_76fce842-09fb-456d-bfcd-dd6fa3d67029
```

这个目录存储了节点中所有 Pod 的日志文件，目录名包含 Pod 名称。我们进入到第一个 Pod 的目录进一步查看日志。

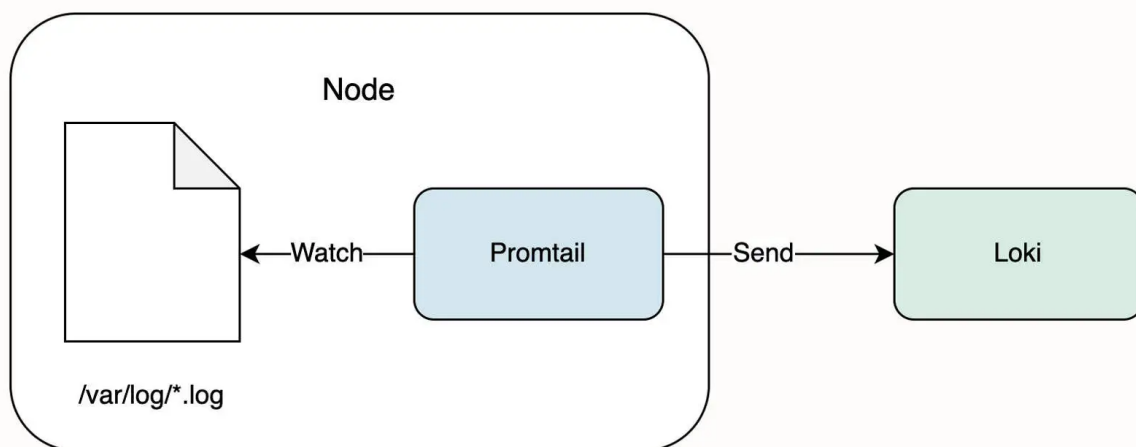
 复制代码

```
1 root@kind-control-plane:/var/log/pods# cd default_log-example-67746dfd97-cq7lb_
2 root@kind-control-plane:/var/log/pods/default_log-example-67746dfd97-cq7lb_201e
3 go-backend# ls
4 0.log
```

在返回的结果中，`0.log` 文件实际上就是 Pod 的日志文件，你可以尝试使用 `tail -f` 命令来查看它，你会发现这个文件会实时产生新的日志。

也就是说，我们只要能获取到这个日志文件的内容，就可以实现对 Pod 日志的采集。

回到 Loki 的 Promtail 组件中，我们知道它是以 DaemonSet 的方式部署在 Kubernetes 集群中的，这意味着，Kubernetes 每一个 Node 节点都将运行 Promtail，它通过实现类似于 `tail -f` 的能力来获取新的日志，并将日志信息发送到 Loki 进行存储，如下图所示。



## Loki

Loki 主要负责日志的存储和读取，它的写入 QPS 可能非常高，所以整体架构设计会更加灵活，它核心的设计思想是**读写分离**。

当一条日志发送到 Loki 之后，它会经历下面的流程。

1. 日志到达的第一站是 Distributor 组件，Distributor 接收到日志后会分发到多个 Ingestor 接收器上。
2. Ingestor 接收器是一个有状态的组件（这也是 Loki 在 Kubernetes 采用 StatefulSet 部署的原因），它会对日志进行 gzip 压缩和缓存，当缓存达到一定规模时，就将日志数据写入到数据库中。

在数据库存储方面，Loki 将存储索引的数据库和存储日志的数据库做了分离，并且支持丰富的云厂商数据库。

当 Grafana 发出日志查询请求时，Loki Querier 查询器组件将首先从索引数据库中获得日志索引，并获取日志。此外，Querier 组件还会从 Ingestor 读取还没有写入数据库的日志信息。

## 生产建议

由于 Loki 的配置比较复杂，而生产环境下日志量一般比较大，所以我也给你提供几点生产建议供你参考。

## 配置持久化

在使用 Helm 安装 Loki 时，默认并不会对日志持久化存储，你需要进行额外的配置。在生产环境下，我建议开启日志的持久化存储。

需要注意的是，持久化存储需要 Kubernetes StorageClass 的支持。在安装前，你需要将下面的内容保存为 pvc-values.yaml。

 复制代码

```
1 loki:
2   persistence:
3     enabled: true
4     size: 500Gi
```

然后，在安装时指定这个配置文件。

 复制代码

```
1 helm upgrade --install loki --namespace=loki-stack grafana/loki-stack --create-
```

需要注意的是，在生产环境下，日志存储量的增长可能会很快，所以你可以提前分配一个较大存储容量的 PVC 持久卷，并留意集群是否支持对持久卷的动态扩容，以便后续随时扩大日志存储容量。

## 使用合适的部署方式

**Loki** 一共有三种部署方式，它们分别是单体模式、扩展模式和微服务模式。

其中，单体模式是我们这节课采用的部署方式，它的安装和配置都非常简单，但性能一般，这种部署方式适用于每天日志量在 **100G** 左右的业务系统。

扩展模式在部署上的主要的变化是读写分离，如果每天的日志量是几百 G 甚至是上 T 级别，建议使用这种部署方式。

你仍然可以通过 Helm Chart 的方式进行部署，在部署的时候你需要选择名称为 grafana/loki-simple-scalable 的 Helm Chart。

微服务部署方式适用于更大的日志存储规模，在部署时你可以选择名称为 grafana/loki-distributed 的 Helm Chart。

## 使用云厂商托管服务存储日志

通常，Kubernetes 持久卷存在最大容量限制，这意味着日志存储早晚会达到上限。此外，PVC 的价格相对较高，并且日志查询频率不高，我们可以选择更加适合的日志存储的方式，例如 AWS S3 和 DynamoDB。

其中，DynamoDB 用于存储日志索引，S3 对象存储用来保存日志，这种日志存储方式没有容量限制，也不需要额外的维护，在日志规模比较大的场景下，我推荐你用这种存储方式。

要使用 S3 和 DynamoDB，你可以在安装 Loki 时指定存储方式。将下面的内容保存为 s3-values.yaml。

 复制代码

```
1 loki:
2   config:
3     schema_config:
4       configs:
5         - from: 2020-05-15
6           store: aws
7           object_store: s3
8           schema: v11
9           index:
10            prefix: loki_
11   storage_config:
12     aws:
13       s3: s3://access_key:secret_access_key@region/bucket_name
14       dynamodb:
15         dynamodb_url: dynamodb://access_key:secret_access_key@region
```

然后在使用 Helm 安装 Loki 时指定这个配置文件即可。更多关于存储方式的配置你可以参考 [这个链接](#)。

## 总结

这节课涉及的知识点比较多，我希望你能多花一点时间放在实战和查询例子上，这会帮助你更好地从零入门 Loki。

简单总结一下，这节课我带你学习了如何通过 **Loki Stack** 来搭建日志系统。在实战环节，我们部署的核心组件包括 **Loki**、**Promtail** 和 **Grafana**。其中，**Loki** 负责日志的存储和查询，**Promtail** 负责采集 **Pod** 日志，**Grafana** 是日志的 **UI** 查询界面。

要查询日志，我们需要编写 **LogQL** 语句，它包括日志流选择器和日志管道操作。其中，日志流选择器决定日志的来源，比如通过 **Kubernetes Label** 来匹配 **Pod** 对象获取日志。管道操作是对日志的进一步处理，比如查找关键字、查找日志的字段值以及一些聚合操作。

**LogQL** 编写起来比较复杂，所以我从真实的例子出发，让你进一步理解 **LogQL** 的能力。其中，对于日志字段值的查询是一项非常重要并且常用的功能，它主要通过解析器来实现，你可以像查询**关系型数据库一样来查询日志**。不过，你需要特别注意业务系统输出日志的格式，通常 **Logfmt** 和 **JSON** 是首选的日志格式。

在介绍常用的 **LogQL** 的时候，我不仅介绍了查询日志的语句，还进一步介绍了如何通过日志查询 **HTTP** 请求相关指标。不过，在生产环境下，**我们一般不这么做**，而是会通过网关来获取更多专业的 **HTTP** 请求相关指标，这部分的内容我将在下节课继续深入介绍。

此外，我还向你介绍了 **Loki** 的工作原理。在这部分内容中，我顺便帮你复习了 **Kubernetes Pod** 的日志存储机制以及 **Promtail** 采集日志的基本原理，实际上，不管是 **EFK** 还是 **ELK** 的日志系统，它们采集日志的基本原理都是类似的。

最后，我还为你提供了几个生产建议，你需要特别注意存储配置和 **Loki** 的部署方式，提前了解日志系统可能存在的瓶颈。

## 思考题

最后，给你留两道思考题吧。

1. 以示例应用为例，请你尝试查询 **1** 小时内请求状态码不为 **200** 的增长速率（**rate**），并按状态码分组（提示：结合常用查询例子中的第一个和第五个例子）；
2. 以示例应用为例，请你尝试查询 **10** 分钟内日志吞吐量最大的前 **3** 个应用程序（提示：前十可以用 **topk** 函数）。

欢迎你给我留言交流讨论，你也可以把这节课分享给更多的朋友一起阅读。我们下节课见。

分享给需要的人，Ta购买本课程，你将得 18 元

生成海报并分享

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 30 | 应用健康：如何迅速判断业务状态和可用性？

下一篇 32 | 监控：如何快速搭建业务 HTTP 健康状态监控？

## 更多课程推荐

### 李三红·搞定 Java 开发基础

极客时间 × 阿里云开发者社区联合出品

李三红  
阿里云程序语言  
与编译器技术总监  
Java Champion

免费订阅



## 精选留言 (8)

写留言



Geek\_28f561

2023-02-21 来自北京

每天上T级别的日志，如果使用阿里云，Loki的存储用什么服务。开源的JuiceFS+阿里的OSS怎么样？

作者回复: 是一个不错的方案，OSS 可以实现接近无限存储的方案，JuiceFS 提供文件系统。不过需要注意两个问题，一个是存储卷的挂载问题，第二个是读取和写入速度的问题。





邵涵

2023-04-01 来自北京

文中收集的pod的日志是pod中进程打印到标准输出的，如果pod中部署的应用是打印日志到日志文件中，要如何收集日志？或者，在将应用部署到k8s中时，是建议都将日志打印到标准输出？

作者回复: 一般建议把日志打印到标准输出中，如果是输出文件的话可能 EFK 技术栈更适合。



黑鹰

2023-03-18 来自北京

思考题1:

```
sum by(status) (  
  rate({app="log-example"} |= `status=` | logfmt | status != 200 [1h])  
)
```

思考题2:

```
topk(3,  
  sum(  
    rate({ app != "" } [10m])  
  ) by(app)  
)
```

作者回复:



黑鹰

2023-03-15 来自北京

配置持久化 和 日志持久化配置 这两个有什么区别呢？

loki:

```
persistence:  
  enabled: true  
  size: 500Gi
```

与

loki:

config:

schema\_config:

configs:

- from: 2020-05-15

store: aws

object\_store: s3

schema: v11

index:

prefix: loki\_

storage\_config:

aws:

s3: s3://access\_key:secret\_access\_key@region/bucket\_name

dynamodb:

dynamodb\_url: dynamodb://access\_key:secret\_access\_key@region

作者回复: 一种是通过 K8s 的 PVC 来提供持久化存储, 一种是使用外部 S3 来存储镜像。生产环境推荐用外部存储系统。



👍 1



争光 Alan

2023-03-14 来自上海

针对elk, splunk, loki能谈一下优缺点吗? 都适合什么场景

作者回复: 小规模场景用 Loki, 大规模场景下 ELK 查询的性能比较好, 但索引文件占用的空间会比较大。Splunk 采集数据比较快, 但查询比较慢。Loki 相对来说性能比较均衡。



👍 1



黑鹰

2023-03-06 来自北京

`{app="log-example"} | logfmt | status = `200``

code模式使用上面的查询语句, 一条日志也没有查询出来。

用 `{app="log-example"} != `status=`` 这条语句, 可以查询到很多 `status=200` 的日志。

请问下是我哪里操作不对吗?

作者回复: 用 `{app="log-example"} |= `status=`` 这条语句能查到 logfmt 格式输出的日志吗? 示例应用输出了 logfmt 和 json 两种格式的日志, 也可以尝试用 `{app="log-example"} | json | status = `200`` 语句来查询。

共 2 条评论 >



旋风

2023-02-18 来自北京

log-example镜像只支持arm?

作者回复: 感谢指正, 已经同时上传了 amd64 版本了。



林龍

2023-02-18 来自广东

想问一下, 项目中已经有了链路的功能, 还需要日志吗? 链路记录了请求值, 响应值, 只能执行的sql, es、redis的语句等。

作者回复: 链路追踪主要用来查看服务的请求链路情况, 日志主要用来排查服务输出的错误日志。

共 2 条评论 >

