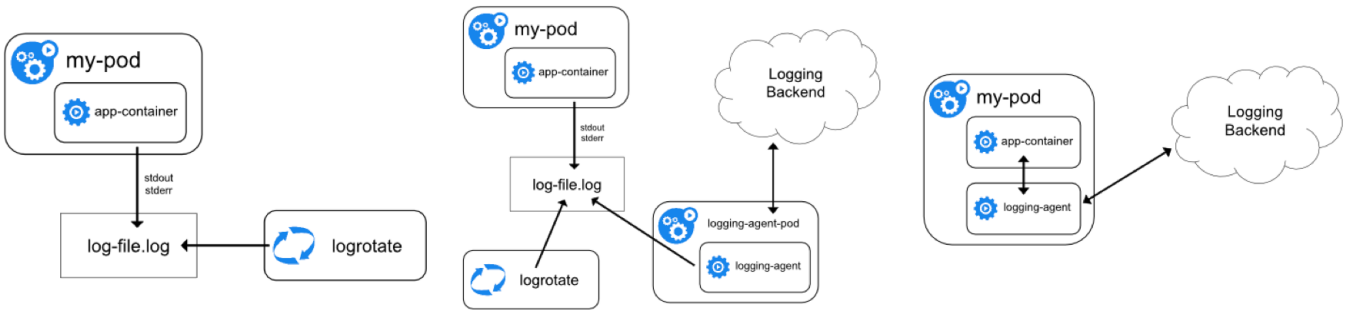
k8s日志采集架构选型

> 1，Node方式：每台节点采用DaemonSet部署agent：
> 原理：每台节点采用DaemonSet部署一个采集日志的agent，从/var/log/containers/目录采集所有容器的日志，而容器中的日志需要遵循docker的日志规范，把日志打入pid为1的主程序的stdout/stderr，这样k8s会自动在/var/log/containers/目录生成对应容器的日志。
>
> 优点：部署维护简单，且能收集所有容器的日志
> 缺点：需要应用程序日志支持stdout/stderr输出，如果每个节点的日志规模过多，单个采集日志的agent可能成为瓶颈，不太灵活。
>
> 2，pod的SideCar方式：
> 原理：每个pod通过SideCar方式部署一个采集日志的agent
>
> 优点：每个pod可单独配置agent，灵活性高
> 缺点：每个pod都需要一个SideCar比较麻烦
>
> 3，应用程序直接推送日志到日志存储：
> 原理：部署在pod的应用程序支持把日志直接推送到日志存储程序
>
> 优点：不用维护日志采集程序，运维简单
> 缺点：开发成本较高。

三种日志采集模型比较



原生方式　　DaemonSet方式　　Sidecar方式

	原生方式	DaemonSet方式	Sidecar方式
采集日志类型	标准输出	标准输出+部分文件	文件
部署运维	低，原生支持	一般，需维护DaemonSet	较高，每个需要采集日志的POD都需要部署sidecar容器
日志分类存储	无法实现	一般，可通过容器/路径等映射	每个POD可单独配置，灵活性高

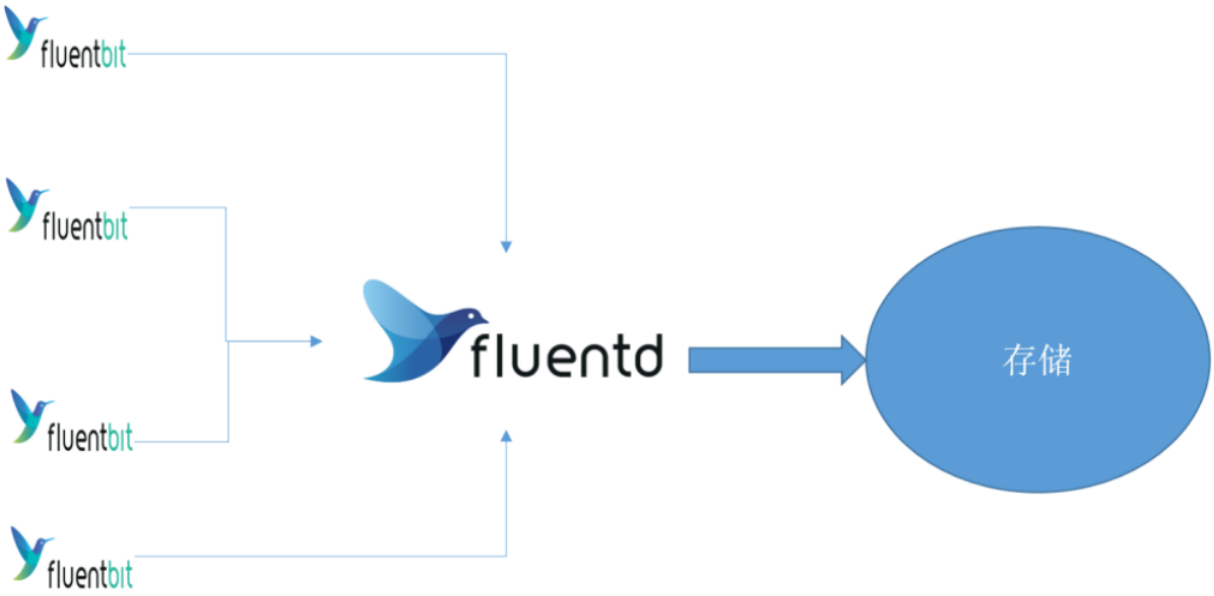
多租户隔离	弱	一般，只能通过配置间隔离	强，通过容器进行隔离，可单独分配资源
支持集群规模	本地存储无限制，若使用syslog、fluentd会有单点限制	中小型规模，业务数最多支持百级别	无限制
资源占用	低，docker engine提供	较低，每个节点运行一个容器	较高，每个POD运行一个容器
查询便捷性	低	较高，可进行自定义的查询、统计	高，可根据业务特点进行定制
可定制性	低	低	高，每个POD单独配置
适用场景	测试、POC等非生产场景	功能单一型的集群	大型、混合型、PAAS型集群

容器内日志

> 和主机采集（daemonset）方式相比，容器内日志的采集方案一般使用 sidecar，好处是：

- 和容器本身的生命周期一致，容器销毁 sidecar 也不再采集，不需要对 sidecar 做回收或管理
- 更方便的做资源规划和计费，日志采集一般会消耗很多的 cpu 和带宽资源，将 sidecar 容器和业务容器绑定在一个 pod 中，可以更方便地做 quota 管理、资源计费。
- sidecar 和业务容器使用 emptydir 共享日志目录，多租户环境下更加安全。
- sidecar 可以动态感知容器的环境信息，如 pod ip，节点名称等，比 daemonset 或者主机采集方便很多。
- 如果容器内也需要暴露 metric，即日志转 metric，使用sidecar 方式也更加方便，sidecar 暴露 prometheus 的 metric 即可。

采用如下架构模式(待定)



fluentd与fluent-bit通信

fluentd与fluent-bit之间的比较（现在fluent-bit支持超过80个插件）

	Fluentd	FluentBit
范围	服务器	嵌入设备和 IoT 设备
内存	约 20 MB	约 150 KB
语言	C 和 Ruby	C
性能	高	高
依赖	以 Ruby Gem 构建，依赖一系列的 Gem	零依赖，可能有些插件会有依赖。
插件	超过三百个	目前15个左右
授权	Apache License v2.0	Apache License v2.0

fluentd的source数据来源类型Forward

> Forward是Fluentd用来在对等体之间路由消息的协议。前向输出插件允许在Fluent Bit和Fluentd之间提供互操作性。除了指定Fluentd的位置外，不需要其他配置步骤，它可以在本地主机中，也可以在远程机器中。
>
> 这个插件提供了两种不同的传输和模式:
>
> - Forward (TCP):使用普通的TCP连接。
> - 安全转发(TLS):当启用TLS时，插件将切换到安全转发模式。
> - fluentd官网

配置参数(fluentd基于forward配置参数详细讲解)

> “安全转发”模式下的“转发”必须配置以下参数:

Key	Description	Default
Host	fluentbit或Fluentd正在侦听转发消息的目标主机。	
Port	TCP目标服务的端口。	24224
Time_as_Integer	设置整数格式的时间戳，启用Fluentd v0.12系列的兼容模式。	False
Upstream	如果Forward将连接到Upstream而不是一个简单的主机，则此属性将定义Upstream配置文件的绝对路径，有关此属性的详细信息请参见Upstream Servers文档部分。	
Tag	当我们传输时覆盖标签。这允许接收管道重新开始，或者属性源。	
Send_options	总是发送选项(使用"size"=消息计数)。	False
Require_ack_response	发送“chunk”选项并等待服务器的“ack”响应。至少启用一次，接收服务器可以控制流量速率。(需要Fluentd v0.14.0+服务器)	False
Compress	设置为“gzip”启用gzip压缩。与Time_as_Integer=True和使用重写标签过滤器动态设置的标签不兼容。(需要Fluentd v0.14.7+服务器)	On

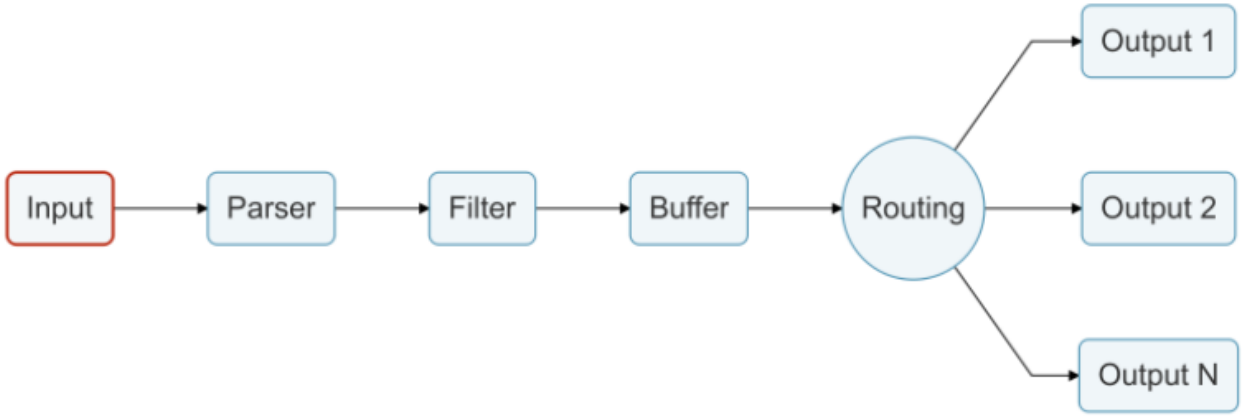
> 安全转发模式配置参数:

Key	Description	Default
Shared_Key	用于授权的远程Fluentd知道的密钥字符串。	
Empty_Shared_Key	使用此选项以零长度秘密连接到Fluentd。	
Username	指定要显示给启用user_auth的Fluentd服务器的用户名。	
Password	指定要显示给启用user_auth的Fluentd服务器的密码。	
Self_Hostname	自动生成证书CN (common name)的默认值。	
tls	启用或禁用TLS支持	Off
tls.verify	证书验证，我这里没有证书，设置为Off	On
tls.debug	设置TLS调试详细级别。它接受以下值:0(没有调试)，1(错误)，2(状态改变)，3(信息)和4 Verbose	1
tls.ca_file	CA证书文件的绝对路径.	
tls.crt_file	证书文件的绝对路径。	

tls.key_file	私钥文件的绝对路径。	
tls.key_passwd	可选tls密码。key_file文件。	

fluent-bit组件基础知识了解

> 采用fluent-bit采集日志采用的时侵入式的方式进行采集
> 详细API介绍参考官方文档 fluent-bit介绍 fluent-bit基本组件了解如下图所示：



1.Input(这里介绍tail)

> 尾部输入插件允许监控一个或几个文本文件。它具有与tail -f shell命令类似的行为。 插件读取Path模式中每个匹配的文件，对于每发现一行(用\n分隔)，它生成一个新记录。可以选择使用数据库文件，这样插件就可以有跟踪文件的历史和偏移量的状态，如果服务重新启动，这对于恢复状态非常有用。

Key	Description	Default
Buffer_Chunk_Size	设置读取文件数据的初始缓冲区大小。此值用于增加缓冲区大小。该值必须符合Unit Size规格。	32k
Buffer_Max_Size	设置每个监视文件的缓冲区大小的限制。当一个缓冲区需要增加(例如:非常长的行)，这个值用于限制内存缓冲区可以增长多少。如果读取文件超过此限制，则从监控文件列表中删除该文件。该值必须符合Unit Size规格。	32k
Path	通过使用通用通配符指定特定日志文件或多个日志文件的模式。也允许使用逗号分隔的多个模式。	
Path_Key	如果启用，它将附加被监视文件的名称作为记录的一部分。分配的值成为映射中的键。	
Exclude_Path	设置一个或多个shell模式，以逗号分隔，以排除匹配某些条件的文件，例如:Exclude_Path *.gz，*.zip	
Offset_Key	如果启用，Fluent Bit将附加当前监控文件的偏移量作为记录的一部	

	分。分配的值成为映射中的键	
Read_from_Head	对于开始时发现的新文件(没有数据库偏移/位置)，从文件的头部读取内容，而不是尾部。	False
Refresh_Interval	刷新观看文件列表的时间间隔，以秒为单位。	60
tls.ca_file	CA证书文件的绝对路径.	
Rotate_Wait	指定在文件被旋转一次时监视文件的额外时间(以秒为单位)，以防某些挂起的数据被刷新。	5
Ignore_Older	忽略比这个时间(以秒为单位)更老的记录。支持m,h,d(分钟，小时，天)语法。默认行为是从指定文件中读取所有记录。仅当指定了Parser并且它可以解析记录的时间时可用。	
Skip_Long_Lines	当一个被监视的文件由于非常长的行(Buffer_Max_Size)而达到它的缓冲区容量时，默认行为是停止监视该文件。Skip_Long_Lines会改变这种行为，并指示Fluent Bit跳过长行，继续处理适合缓冲区大小的其他行。	Off
DB	指定跟踪监视文件和偏移量的数据库文件。	
DB.sync	设置默认同步(I/O)方法。取值范围:Extra, Full, Normal, Off。这个标志影响内部SQLite引擎如何进行磁盘同步，关于每个选项的详细信息请参阅本节。大多数工作负载场景都可以使用普通模式，但是如果在每次写操作之后都需要完全同步，那么应该设置完全模式。请注意，full具有很高的I/O性能成本。	normal
DB.locking	指定仅由Fluent Bit访问数据库。启用此特性有助于提高访问数据库时的性能，但它限制了任何外部工具查询内容。	false
DB.journal_mode	设置WAL (journal mode for databases)日志模式。开启WAL可以提供更高的性能。注意，WAL与共享网络文件系统不兼容。	WAL
Mem_Buf_Limit	设置Tail插件在添加数据到引擎时可以使用的内存限制。如果达到了极限，就会暂停;当刷新数据时，它将恢复。	
exit_on_eof	当读取一个文件时，当它到达文件的末尾时就会退出。用于批量加载和测试	false
Parser	指定解析器的名称，以将条目解释为结构化消息。	
Key	当消息是非结构化的(没有应用解析器)时，它将作为字符串附加到密钥名日志下。此选项允许为该键定义另一个名称。	log
Tag	设置一个标记(带有regex-extract字段)，它将被放置在读取的行上。例如kube。< namespace_name >。< pod_name >。< container_name >。注意，支持“标签扩展”:如果标签包含星号(*)，该星号将被替换为被监视文件的绝对路径(也参见Tail + Kubernetes Filter的工作流)。	log

Tag_Regex	设置正则表达式从文件名中提取字段。如(?[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*)_(?[^_]+)_(?.+)-	log

注意：
　　如果没有指定数据库参数DB，默认情况下，插件将从头开始读取每个目标文件。
这也可能导致一些不需要的行为，例如，当一行大于Buffer_Chunk_Size和
Skip_Long_Lines时，将从每个Refresh_Interval的开始读取文件，直到文件被旋转。

> 支持多行 新的多行核心通过以下配置暴露

Key	Description	Default
multiline.parser	指定应用于内容的一个或多行解析器定义。	32k

> 正如Multiline Parser文档中所述，现在我们提供了内置的配置模式。注意，当使用一个新的多行。解析器定义，你必须从你的尾部禁用旧配置，如:
>
> - parser
> - parser_firstline
> - parser_N
> - multiline
> - multiline_flush
> - docker_mode
>
> 多行和容器
> 如果您正在运行Fluent Bit来处理来自Docker或CRI等容器的日志，则可以为此使用新的内置模式。这将有助于重新组装原来由Docker或CRI分割的多行消息:

```
[INPUT]
    name              tail
    path              /var/log/containers/*.log
    multiline.parser  docker, cri
```

> 具体多行详情参考官方文档

2.Buffering & Storage

> Fluent Bit的最终目标是收集、解析、过滤日志并将其传送到中心位置。在这个工作流中有许多阶段，其中一个关键部分是缓冲能力:一种将处理过的数据放置到临时位置的机制，直到准备好发送为止。 在默认情况下，当Fluent Bit处理数据时，它使用内存作为存储记录的主要和临时位置，但在某些场景中，理想的做法是基于文件系统中的持久缓冲机制，以提供聚合和数据安全功能。 选择正确的配置

2.1 Chunks, Memory, Filesystem and Backpressure的理解

2.1.1 Chunks

> 当输入插件(源)发出记录时，引擎将记录分组在Chunk中。块大小通常在2MB左右。通过配置，引擎决定在哪里放置这个Chunk，默认情况下，所有的Chunk只在内存中创建。

2.1.2 Buffering and Memory

> 如上所述，由引擎生成的块被放置在内存中，但这是可配置的。 如果内存是输入插件设置的唯一机制，它就会尽可能多地存储数据(内存)。这是速度最快、系统开销更少的机制，但如果服务由于网络慢或远程服务响应不及时而不能足够快地交付记录，则Fluent Bit内存使用量将增加，因为它将积累比它能交付的更多的数据。 在具有背压的高负载环境中，高内存使用率的风险是有机会得到的

2.1.3 Filesystem buffering to the rescue

> 启用文件系统缓冲有助于反压和整体内存控制。 在幕后，内存和文件系统缓冲机制并不是互斥的，事实上，当为你的输入插件(源)启用文件系统缓冲时，你得到了两个世界中最好的:性能和数据安全。 当启用文件系统缓冲时,引擎的行为是不同的,块创建后,它将内容存储在内存中,也映射磁盘上的一个副本(通过mmap(2)),这一块是活跃在内存和磁盘备份叫做“块反对.

2.2 Configuration

> 存储层配置发生在三个区域:

- Service Section
- Input Section
- Output Section

> 已知的Service部分为存储层配置全局环境，Input部分定义使用哪种缓冲机制，输出定义逻辑队列的限制。

2.2.1 Service Section Configuration

> Service部分引用在主配置文件中定义的部分

Key	Description	Default
storage.path	在文件系统中设置一个可选的位置来存储流和数据块。如果不设置此参数，则输入插件只能使用内存缓冲。	
storage.sync	配置文件系统的同步方式。它可以取正常值或满值。	normal
storage.checksum	当从文件系统写入和读取数据时，启用数据完整性检查。存储层采用CRC32算法。	Off
storage.max_chunks_up	如果输入插件启用了文件系统存储类型，该属性将设置内存中可up的chunk的最大数量。这有助于控制内存使用。	128
storage.backlog.mem_limit	如果存储。路径设置后，Fluent Bit将查找尚未交付且仍在存储层中的数据块，这些数据块称为待定数据。此选项配置处理这些记录时使用的内存最大值提示。	5M
storage.metrics	如果在main [SERVICE]部分启用了http_server选项，该选项将注册一个新的端点，在那里可以使用存储层的内部指标。有关详细信息，请参阅监视一节。	off

> 如下配置：

```
    [SERVICE]
       flush                     1
       log_Level                 info
       storage.path              /var/log/flb-storage/
       storage.sync              normal
       storage.checksum          off
       storage.backlog.mem_limit 5M
```

> 该配置配置了一个可选的缓冲机制，其中它的根数据是/var/log/flb-storage/，它将使用普通的同步模式，没有校验和，在处理待定数据时最多有5MB的内存。

2.2.2 Input Section Configuration

> 任意Input插件都可以配置它们的存储首选项，下表描述了可用的选项

Key	Description	Default
storage.type	指定要使用的缓冲机制。它可以是内存或文件系统。	memory

> 下面的示例配置了一个提供文件系统缓冲功能的服务和两个Input插件，第一个基于文件系统，第二个仅使用内存。

```
     [SERVICE]
       flush                     1
       log_Level                 info
       storage.path              /var/log/flb-storage/
       storage.sync              normal
       storage.checksum          off
       storage.backlog.mem_limit 5M

     [INPUT]
       name          cpu
       storage.type  filesystem

     [INPUT]
       name          mem
       storage.type  memory
```

2.2.3 Output Section Configuration

> 如果某些块是文件系统存储。基于类型，可以控制输出插件的逻辑队列的大小。下表描述了可供选择的方案:

Key	Description	Default
storage.total_limit_size	限制文件系统中当前输出逻辑目标的最大chunk数量。	

> 下面的示例在文件系统中使用CPU使用示例创建记录，然后将这些记录发送到谷歌Stackdriver服务，将逻辑队列(缓冲)限制为5M:

```
    [SERVICE]
      flush                     1
      log_Level                 info
      storage.path              /var/log/flb-storage/
      storage.sync              normal
      storage.checksum          off
      storage.backlog.mem_limit 5M

    [INPUT]
      name                      cpu
      storage.type              filesystem

    [OUTPUT]
      name                      stackdriver
      match                     *
      storage.total_limit_size  5M
```

> 如果由于某种原因Fluent Bit由于网络问题而脱机，它将继续缓冲CPU样本，但只保留最大5M的最新数据。

采集部署

1.部署es和kibana

> 调用API接口部署es

2.采用Node方式采集容器内日志(详细参数参考sidecar部署案例2)

2.1 采用DaemonSet部署agent

> 原理：
> 从/var/log/containers/目录采集所有容器的日志，而容器中的日志需要遵循docker的日志规范，把日志打入pid为1的主程序的stdout/stderr，这样k8s会自动在/var/log/containers/目录生成对应容器的日志。

2.1.1 配置configMap

1. input配置

```
   input-kubernetes.conf: |
       [INPUT]
           Name              tail
           Tag               kube.*
           Path              /var/log/containers/*.log
```

```
        Parser            docker
        DB                /var/log/flb_kube.db
        Mem_Buf_Limit     5MB
        Skip_Long_Lines   On
        Refresh_Interval  10
```

2. filter过滤

```
    filter-kubernetes.conf: |
        [FILTER]
            Name                kubernetes
            Match               kube.*
            Kube_URL            https://kubernetes.default.svc:443
            Kube_CA_File        /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
            Kube_Token_File     /var/run/secrets/kubernetes.io/serviceaccount/token
            Kube_Tag_Prefix     kube.var.log.containers.
            Merge_Log           On
            Merge_Log_Key       log_processed
            K8S-Logging.Parser  On
            K8S-Logging.Exclude Off
```

3. 配置解析Parser

```
    parsers.conf: |
        [PARSER]
            Name   apache
            Format regex
            Regex  ^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?: +
            Time_Key time
            Time_Format %d/%b/%Y:%H:%M:%S %z

        [PARSER]
            Name   apache2
            Format regex
            Regex  ^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?: +
            Time_Key time
            Time_Format %d/%b/%Y:%H:%M:%S %z

        [PARSER]
            Name   apache_error
            Format regex
            Regex  ^\[[^ ]* (?<time>[^\]]*)\] \[(?<level>[^\]]*)\](?: \[pid (?<pid>[^\]]*)\])?( \

        [PARSER]
            Name   nginx
            Format regex
            Regex ^(?<remote>[^ ]*) (?<host>[^ ]*) (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>
            Time_Key time
            Time_Format %d/%b/%Y:%H:%M:%S %z


        [PARSER]
```

```
        Name   json
        Format json
        Time_Key time
        Time_Format %d/%b/%Y:%H:%M:%S %z

    [PARSER]
        Name        docker
        Format      json
        Time_Key    time
        Time_Format %Y-%m-%dT%H:%M:%S.%L
        Time_Keep   On

    [PARSER]
         #http://rubular.com/r/tjUt3Awgg4
        Name cri
        Format regex
        Regex ^(?<time>[^ ]+) (?<stream>stdout|stderr) (?<logtag>[^ ]*) (?<message>.*)$
        Time_Key    time
        Time_Format %Y-%m-%dT%H:%M:%S.%L%z

    [PARSER]
        Name        syslog
        Format      regex
        Regex       ^\<(?<pri>[0-9]+)\>(?<time>[^ ]* {1,2}[^ ]* [^ ]*) (?<host>[^ ]*) (?<ider
        Time_Key    time
        Time_Format %b %d %H:%M:%S
    [PARSER]
        Name        java_multiline
        Format      regex
        Time_Key    time
        Time_Key    Time_Format %Y-%m-%d %H:%M:%S
        Regex       /^\[(?<time>\d{4}-\d{1,2}-\d{1,2} \d{1,2}:\d{1,2}:\d{1,2})\] \[(?<level>
```

4. output配置

```
output-elasticsearch.conf: |
    [OUTPUT]
        Name            es
        Match           kube.*
        Host            ${FLUENT_ELASTICSEARCH_HOST}
        Port            ${FLUENT_ELASTICSEARCH_PORT}
        Logstash_Format On
        Logstash_Prefix fluent-bit
        Replace_Dots    On
        Retry_Limit     2
        HTTP_User       ${FLUENT_ELASTICSEARCH_USER}
        HTTP_Passwd     ${FLUENT_ELASTICSEARCH_PWD}
        tls.verify      Off
        tls             On
```

```
            #Trace_Output    On
            #Trace_Error     On
```

> 基于上面的配置，configMap完成

2.1.2 以daemonset创建fluent-bit

> fluent-bit支持多种存储点，根据不同的存储点有不容的配置具体参考官方Outputs
>
> 下面以es作为存储点作为示例

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
    name: fluent-bit
    namespace: fluent
    labels:
        k8s-app: fluent-bit-logging
        version: v1
        kubernetes.io/cluster-service: "true"
spec:
    selector:
        matchLabels:
            k8s-app: fluent-bit-logging
    template:
        metadata:
            labels:
                k8s-app: fluent-bit-logging
                version: v1
                kubernetes.io/cluster-service: "true"
        annotations:
            prometheus.io/scrape: "true"
            prometheus.io/port: "2020"
            prometheus.io/path: /api/v1/metrics/prometheus
        spec:
            containers:
            - name: fluent-bit
              image: fluent/fluent-bit:1.8.1
              imagePullPolicy: Always
              ports:
              - containerPort: 2020
              env:
            - name: FLUENT_ELASTICSEARCH_HOST
              value: "fluent-es3-es-http"
            - name: FLUENT_ELASTICSEARCH_PORT
              value: "9200"
            - name: FLUENT_ELASTICSEARCH_USER
              value: "elastic"
            - name: FLUENT_ELASTICSEARCH_PWD
              value: "k31c1Ym48nh609EVl77QKaSd"
            volumeMounts:
            - name: varlog
```

```
        mountPath: /var/log
      - name: varlibdockercontainers
        mountPath: /var/lib/docker/containers
        readOnly: true
      - name: fluent-bit-config
        mountPath: /fluent-bit/etc/
  terminationGracePeriodSeconds: 10
  volumes:
  - name: varlog
    hostPath:
      path: /var/log
  - name: varlibdockercontainers
    hostPath:
      path: /var/lib/docker/containers
  - name: fluent-bit-config
    configMap:
      name: fluent-bit-config
  serviceAccountName: fluent-bit
  tolerations:
  - key: node-role.kubernetes.io/master
    operator: Exists
    effect: NoSchedule
  - operator: "Exists"
    effect: "NoExecute"
  - operator: "Exists"
    effect: "NoSchedule"
```

> 配置文件的过程中只需要注意日志文件容器内的挂载点为

```
/var/lib/docker/containers
```

2.1.3 启动以及查看日志

```
kubectl apply -f fluent-bit-configmap.yaml
kubectl apply -f fluent-bit-ds.yaml
```

> 启动成功后通过kibana查看日志（详细配置步骤参考下面sidecar案例1）

2.2 部署其他应用，采用上面的agent采集日志

> 只需要部署自己的应用即可，都能采集到日志

3.采用Sidecar采集容器内日志

> 原理：
> 每个pod通过SideCar方式部署一个采集日志的agent

示例1:采集nginx日志(详细参数介绍请看案例2)

> 采用Sidecar部署时可以实现一个应用一个Index 配置过程中需要注意：
> 1. configMap文件中需要注意数据来源 Input、Output 2. 部署应用时必须保证 fluent-bit的挂载点volumeMounts且指定configMap配置文件
> 3. 应用日志的挂载点volumeMounts。

1. 创建fluent-bit的configMap配置文件

> fluentbit-config.yaml配置文件(具体配置参数在下面的第二个示例中讲解)：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluentbit-config-sidecar
  namespace: fluent
  labels:
     app.kubernetes.io/name: fluentbit
data:
  fluent-bit.conf: |-
     [SERVICE]
         Parsers_File     parsers.conf
         Daemon           Off
         Log_Level        info
         HTTP_Server      off
         HTTP_Listen      0.0.0.0
         HTTP_Port        24224
     [INPUT]
         Name             tail
         Tag              nginx-access.*
         Path             /var/log/nginx/*.log
         Mem_Buf_Limit    5MB
         DB               /var/log/flt_logs.db
         Refresh_Interval 5
         Ignore_Older     10s
         Rotate_Wait      5

     [FILTER]
         Name             parser
         Match            *
         Parser           nginx
         Key_name         aux
     [FILTER]
         Name             record_modifier

         Match            *
         Key_name         message
         Record           podname ${HOSTNAME}
         Record           namespace  ${POD_NAMESPACE}
         Record           nodename  ${NODE_NAME}
         Record           environment prod
     #具体参数的详细介绍下面的示例2中做了详细备注
     [OUTPUT]
         Name             es
```

```
        Match            *
        Host             ${FLUENT_ELASTICSEARCH_HOST}
        Port             ${FLUENT_ELASTICSEARCH_PORT}
        Logstash_Format On
        Logstash_Prefix ${INDEX_PREFIX}-fluentbit-sidecar
        Replace_Dots    On
        Retry_Limit     2
        HTTP_User       ${FLUENT_ELASTICSEARCH_USER}
        HTTP_Passwd     ${FLUENT_ELASTICSEARCH_PWD}
        tls.verify      Off
        tls             On

    [OUTPUT]
        Name            forward
        Match           *
        Host            172.17.0.24
        Port            31672

  parsers.conf: |
    [PARSER]
        Name   nginx
        Format regex
        Regex ^(?<remote>[^ ]*) (?<host>[^ ]*) (?<user>[^ ]*)   \[(?<time>[^\]]*)\] "(?<method>\S
                (?: +(?<path>[^\"]  *?)(?: +\S*)?)?" (?<code>[^ ]*) (?<size>[^ ]*)(?:
                "(?  <referer>[^\"]*)" "(?<agent>[^\"]*)")? \"-\"$
        Time_Key time
        Time_Format %d/%b/%Y:%H:%M:%S %z
```

注意： （上面map结束） PARSER的配置需要的正则表达式一般官方文档中已经给出创建的类型的写法如下；

```
 parsers.conf: |
   [PARSER]
       Name   apache
       Format regex
       Regex  ^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?:
                +(?<path>[^\"]*?)(?: +\S*)?)?" (?<code>[^ ]*) (?<size>[^ ]*)(?: "(?<referer>[^\"]*)"
                "(?<agent>[^\"]*)")?$
       Time_Key time
       Time_Format %d/%b/%Y:%H:%M:%S %z

   [PARSER]
       Name   apache2
       Format regex
       Regex  ^(?<host>[^ ]*) [^ ]* (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?:
               +(?<path>[^ ]*) +\S*)?" (?<code>[^ ]*) (?<size>[^ ]*)(?: "(?<referer>[^\"]*)"
               "(?<agent>[^\"]*)")?$
       Time_Key time
       Time_Format %d/%b/%Y:%H:%M:%S %z

   [PARSER]
```

```
    Name   apache_error
    Format regex
    # 为了方便打印pdf这里换行了
    Regex  ^\[[^ ]* (?<time>[^\]]*)\] \[(?<level>[^\]]*)\](?: \[pid (?<pid>[^\]]*)\])?
           ( \[client (?<client>[^\]]*)\])? (?<message>.*)$

[PARSER]
    Name   nginx
    Format regex
    Regex ^(?<remote>[^ ]*) (?<host>[^ ]*) (?<user>[^ ]*) \[(?<time>[^\]]*)\] "(?<method>\S+)(?:
           +(?<path>[^\"]*?)(?: +\S*)?)?" (?<code>[^ ]*) (?<size>[^ ]*)(?: "(?<referer>[^\"]*)"
            "(?<agent>[^\"]*)")?$
    Time_Key time
    Time_Format %d/%b/%Y:%H:%M:%S %z

[PARSER]
    Name   json
    Format json
    Time_Key time
    Time_Format %d/%b/%Y:%H:%M:%S %z

[PARSER]
    Name        docker
    Format      json
    Time_Key    time
    Time_Format %Y-%m-%dT%H:%M:%S.%L
    Time_Keep   On

[PARSER]
    # http://rubular.com/r/tjUt3Awgg4
    Name cri
    Format regex
    Regex ^(?<time>[^ ]+) (?<stream>stdout|stderr) (?<logtag>[^ ]*) (?<message>.*)$
    Time_Key    time
    Time_Format %Y-%m-%dT%H:%M:%S.%L%z

[PARSER]
    Name        syslog
    Format      regex
    Regex       ^\<(?<pri>[0-9]+)\>(?<time>[^ ]* {1,2}[^ ]* [^ ]*) (?<host>[^ ]*)
                (?<ident>[a-zA-Z0-9_\/\.\-]*)(?:\[(?<pid>[0-9]+)\])?(?:[^\:]*\:)? *(?<message>.*
    Time_Key    time
    Time_Format %b %d %H:%M:%S
```

2. 创建pod

```
 apiVersion: v1
 kind: Service
 metadata:
    name: nginx-log-sidecar
```

```
    namespace: fluent
    labels:
        app: nginx
 spec:
    ports:
    - name: http
      port: 80
      targetPort: 80
    selector:
        app: nginx
    type: ClusterIP

---

apiVersion: apps/v1
kind: Deployment
metadata:
    name: nginx-log-sidecar
    namespace: fluent
spec:
    replicas: 1
    selector:
        matchLabels:
            app: nginx
    template:
        metadata:
            labels:
                app: nginx
    spec:
        containers:
        - name: nginx
          image: nginx:latest
          imagePullPolicy: Always
          volumeMounts:
          - mountPath: /var/log/nginx
            name: log-volume
          resources:
            requests:
                cpu: 10m
                memory: 50Mi
            limits:
                cpu: 50m
                memory: 100Mi
          - name: fluentbit-logger
            image: fluent/fluent-bit:1.8.1
            imagePullPolicy: IfNotPresent
            resources:
                requests:
                    cpu: 20m
                    memory: 100Mi
                limits:
                    cpu: 100m
```

```
            memory: 200Mi
        env:
        - name: FLUENT_ELASTICSEARCH_HOST
          value: "fluent-es3-es-http"
        - name: FLUENT_ELASTICSEARCH_PORT
          value: "9200"
        - name: FLUENT_ELASTICSEARCH_USER
          value: "elastic"
        - name: FLUENT_ELASTICSEARCH_PWD
          value: "DD45Ge2K42F728Qw9A8NiKPR"
        - name: INDEX_PREFIX
          valueFrom:
            fieldRef:
                fieldPath: metadata.name
        - name: NODE_NAME
          valueFrom:
            fieldRef:
                fieldPath: spec.nodeName
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
                fieldPath: metadata.namespace
        - name: POD_IP
          valueFrom:
            fieldRef:
                fieldPath: status.podIP
        volumeMounts:
        - mountPath: /fluent-bit/etc
          name: config
        - mountPath: /var/log/nginx
            name: log-volume
        volumes:
        - name: config
          configMap:
            name: fluentbit-config-sidecar
        - name: log-volume
          emptyDir: {}
```

> 配置文件中通过env动态配置参数，比如设置的索引前缀、节点名、命名空间以及es的相关参数
>
> 通过这种在创建应用pod的同时创建sidecar的agent方式收集应用日志

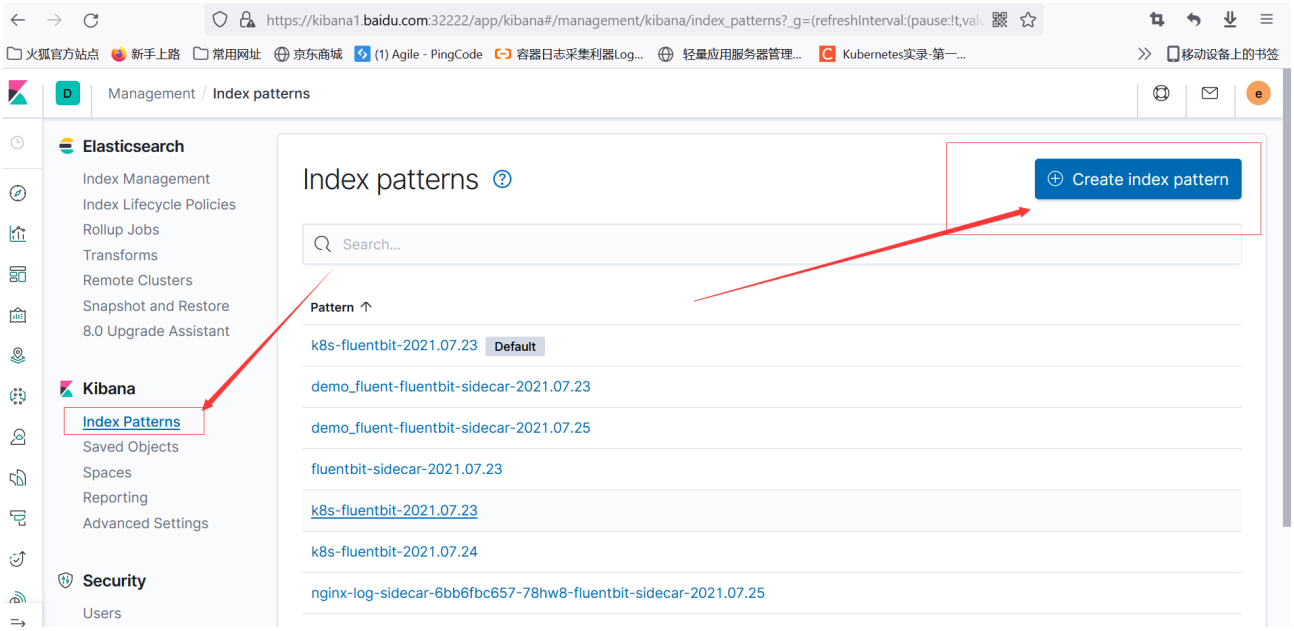
3.查看service

> 通过kubectl get svc -n fluent查看svc
>
> 找到上面创建的service
>
> 通过curl 进行访问，以便产生日志

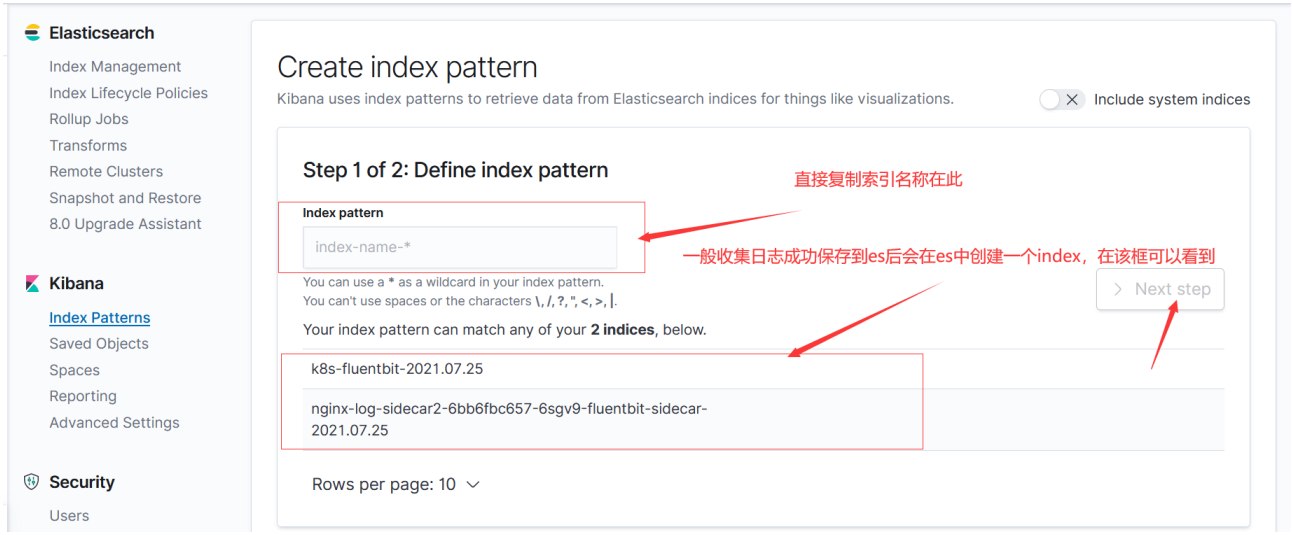
4.登录kibana查看

1. 创建索引模板

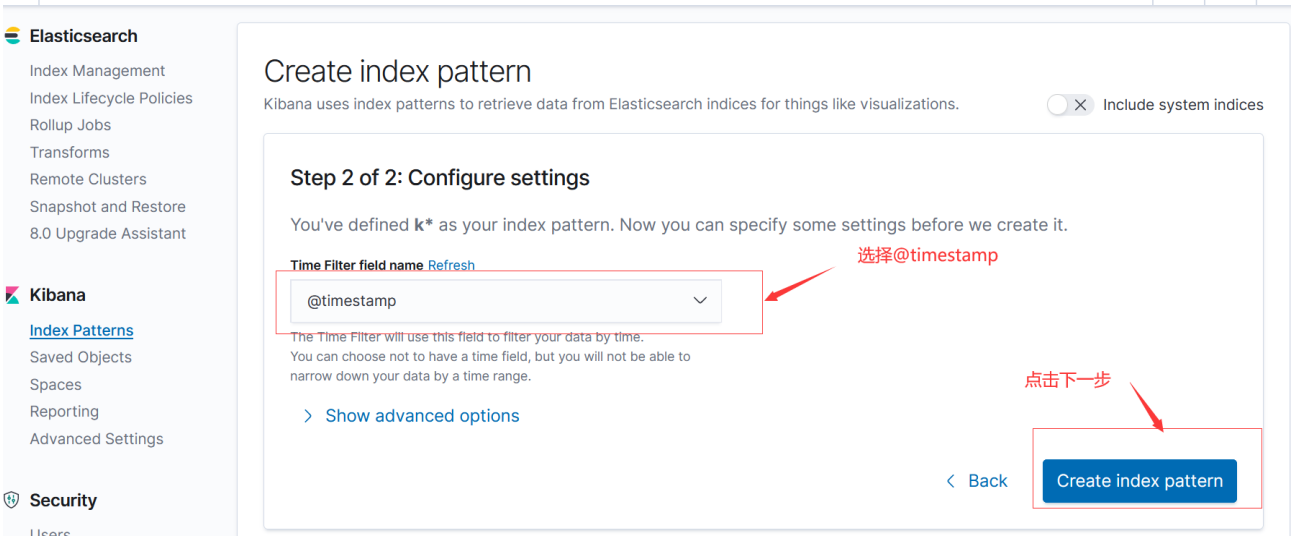
从导航 Mangement -> Kibana -> Index Patterns创建索引模板 点击Index Patterns



点击Create index pattern



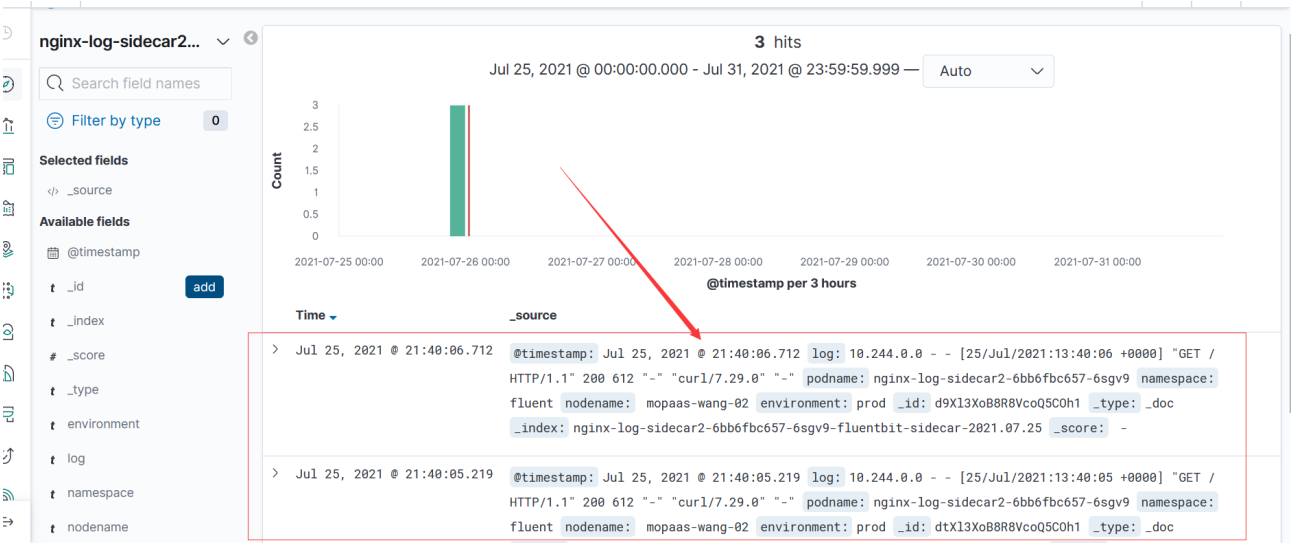
点击Next step



1. 查看日志信息

点击导航菜单：Discover

选择创建的索引模板即可查看到日志信息



示例2:采集SpringBoot项目日志

1. 通过Dockerfile将jar打包成镜像

```
FROM azul/zulu-openjdk:8
VOLUME /apps
ADD a.jar a.jar
RUN bash -c 'touch /a.jar'
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","a.jar"]
```

前提条件时将jar上传到服务器

执行 docker build -t a_mirror .

2. 创建configMap文件及参数的详细介绍

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: webapp-fb-config
    namespace: fluent
data:
    fluent-bit.conf: |-
        [SERVICE]
            Flush         1  #buffer里的数据每隔1秒写到output插件里，
                             这里写到ES里。
            Log_Level     info
            Daemon        off
            Parsers_File  parsers.conf #指向了另外一个配置文件，里面配置所有的parser。
            HTTP_Server   Off
            HTTP_Listen   0.0.0.0
```

```
        HTTP_Port     2020
@INCLUDE input-java.conf
@INCLUDE filter-stdout.conf
@INCLUDE output-stdout.conf

input-java.conf: |-
    [INPUT]
        Name              tail #指定input的采集方式
        Tag               javaLog
        Path              /log/* #数据来源
        Mem_Buf_Limit     5MB #设置Tail插件在添加数据到引擎时可以使用的内存限制。如果达到了极限
                          就会暂停；当刷新数据时，它将恢复。
        Buffer_Chunk_Size 256KB #设置读取文件数据的初始缓冲区大小。此值用于增加缓冲区大小。
                                该值必须符合Unit Size规格。

        Buffer_Max_Size   512KB #设置每个监视文件的缓冲区大小的限制。当一个缓冲区需要增加
                                (例如:非常长的行)，这个值用于限制内存缓冲区可以增长多少。
                                如果读取文件超过此限制，则从监控文件列表中删除该文件。
                                该值必须符合Unit Size规格。
        Multiline         On #开启多行支持
        Refresh_Interval  10
        Skip_Long_Lines   On #开启这个时，当一个被监视的文件由于非常长的行(Buffer_Max_Size)
                             而达到它的缓冲区容量时，默认行为是停止监视该文件。
                             Skip_Long_Lines会改变这种行为，并指示Fluent Bit跳过长行，
                             继续处理适合缓冲区大小的其他行。
        Parser_Firstline  java_multiline
        DB                /var/log/flb_kube.db 文件指针偏移量存储点

filter-stdout.conf: |-
    [FILTER]
        Name              record_modifier
        Match             javaLog #匹配input数据源中Tag为javaLog的数据，进行过滤
        Key_name          log
        #为采集到的数据源加上下面的标签记录，数据均为deployment动态传参
        Record            hostname ${HOSTNAME}
        Record            environment dev
        Record            podname ${HOSTNAME}
        Record            namespace  ${POD_NAMESPACE}
        Record            nodename  ${NODE_NAME}

output-stdout.conf: |-
    [OUTPUT]
        Name              es
        #匹配到tag为任意的数据源，这里只有一个input，等价于Match javaLog*
        Match             *
        #es的host，这里是通过deployment动态传参
        Host              ${FLUENT_ELASTICSEARCH_HOST}
        #es的端口号
        Port              ${FLUENT_ELASTICSEARCH_PORT}
        #是否采用类似logstash的index，可以根据时间设置index名字
        Logstash_Format   On
        ##索引名称的前缀
```

```
            Logstash_Prefix  ${INDEX_PREFIX}
            #
            Replace_Dots     On
            #传输失败后重试次数，默认为2，设置为False时，无限次重试
            Retry_Limit      2
            #Generate_ID  On #对记录去重，可能有性能消耗。
            #Trace_Output Off #打印elasticsearch API calls 调试的时候用。
            #es的登录名和密码
            HTTP_User        ${FLUENT_ELASTICSEARCH_USER}
            HTTP_Passwd      ${FLUENT_ELASTICSEARCH_PWD}
            #证书验证，我这里没有证书，设置为Off，如果没有证书，但是设置为On，不会报错，
            #但是input的数据源不会采集到es中
            tls.verify       Off
            #启用或禁用TLS支持
            tls              On


    parsers.conf: |-
        [PARSER]
            Name        java_multiline
            Format      regex
            Time_Key    time
            Time_Key    Time_Format %Y-%m-%d %H:%M:%S
            Regex       /^\[(?<time>\d{4}-\d{1,2}-\d{1,2} \d{1,2}:\d{1,2}:\d{1,2})\]
                        \[(?<level>(INFO|ERROR|WARN))\] \[(?<thread>.*)\] (?<message>[\s\S].*
```

2.1 在配置文件解析 Input-tail以及Output基于es的上面已经详细介绍

> 在一些重要参数添加了详细注解，在SERVICE配置的参数Flush表示每隔多少秒执行一次push到存储点，这样操作如果缓冲区中存在数据，那么存储点首先pull缓冲区内的数据，在根据上次读取文件的指针偏移量（BD中存储了该偏移量）继续读取日志，存入到缓存区，再从缓冲区push到存储点（es）。
>
> 基本上这样配置数据理论上不会丢失；

2.2 Filters配置

> fluent-bit 的filter根据不同类型有不同类型的参数 record_modifier类型支持自定义正则解析。指定自定义配置的Parser。
>
> 详细配置参考官方API

2.3 Parser配置

> 目前基本上大部分业务日志格式在案例1已经给出，他具体支持那些类型参考

官方文档

3. 创建应用pod

```
        kind: Deployment
        apiVersion: apps/v1
```

```
metadata:
  name: jave-deploy
  namespace: fluent
  labels:
    app: javaApp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: javeApp
      env: dev
  template:
    metadata:
      labels:
        app: javeApp
        env: dev
    spec:
      terminationGracePeriodSeconds: 20
      imagePullSecrets:
      - name: haid-registry-secret
      #nodeSelector:
      #  app: javeApp
      containers:
      - name: webapp
        image: a_mirror:latest
        imagePullPolicy: IfNotPresent
        #设置容器类日志的挂载点
        volumeMounts:
        - name: log-storage
          mountPath: /usr/local/tomcat/webapps/ROOT/WEB-INF/classes/dbconfig.properties
          subPath: dbconfig.properties
        - name: log-storage
          mountPath: /log
        ports:
        - name: http
          containerPort: 8080
          protocol: TCP
        resources:
          requests:
            cpu: 800m
            memory: "2Gi"
          limits:
            cpu: 2
            memory: "6Gi"
        livenessProbe:
          initialDelaySeconds: 120
          periodSeconds: 10
          timeoutSeconds: 5
          httpGet:
            path: /a
            port: 8080
            scheme: HTTP
```

```
    readinessProbe:
      initialDelaySeconds: 120
      periodSeconds: 10
      timeoutSeconds: 5
      httpGet:
        path: /a
        port: 8080
        scheme: HTTP

  - name: javeApp-fb-sidecar
    image: fluent/fluent-bit:1.8.1
    imagePullPolicy: IfNotPresent
    name: fluentbit-sidecar
    #configMap的参数来源此
    env:
    - name: FLUENT_ELASTICSEARCH_HOST
      value: "fluent-es3-es-http"
    - name: FLUENT_ELASTICSEARCH_PORT
      value: "9200"
    - name: FLUENT_ELASTICSEARCH_USER
      value: "elastic"
    - name: FLUENT_ELASTICSEARCH_PWD
      value: "DD45Ge2K42F728Qw9A8NiKPR"
    - name: INDEX_PREFIX
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP

    resources:
      limits:
        cpu: 200m
        memory: 800Mi
      requests:
        cpu: 20m
        memory: 100Mi
    #设置容器类日志的收集点
    volumeMounts:
    - name: fluentbit-config
      mountPath: /fluent-bit/etc
    - name: log-storage
```

```
            mountPath: /log
        volumes:
        - name: fluentbit-config
          configMap:
            name: webapp-fb-config
        - name: log-storage
          emptyDir: {}


---
#仅仅只是为了方便测试，没有采用ingress
apiVersion: v1
kind: Service
metadata:
  name: testjavasvc
  namespace: fluent
spec:
  type: NodePort
  ports:
  - port: 8080
    targetPort: 8080
    -- 节点暴露给外部的端口（范围必须为30000-32767）测试接口没有使用ingress
    nodePort: 30004
  selector:
    app: javeApp
```

> 执行以下命令启动

```
kubectl apply -f webapp-fb-config.yaml
kubectl apply -f javeApp-deploy.yaml
```

注意：应用部署文件中保证有两个镜像的配置 ①应用镜像以及相应的配置 ②fluent-bit的镜像以及配置 着重注意日志容器内的挂载点以及configMap中Input文件的path保持一致

3.1. 应用状态检察

1. 通过一下命令查看pod状态

```
kubectl get pods -n (命名空间)
```

2. 当pod启动之后查看日志是否正常

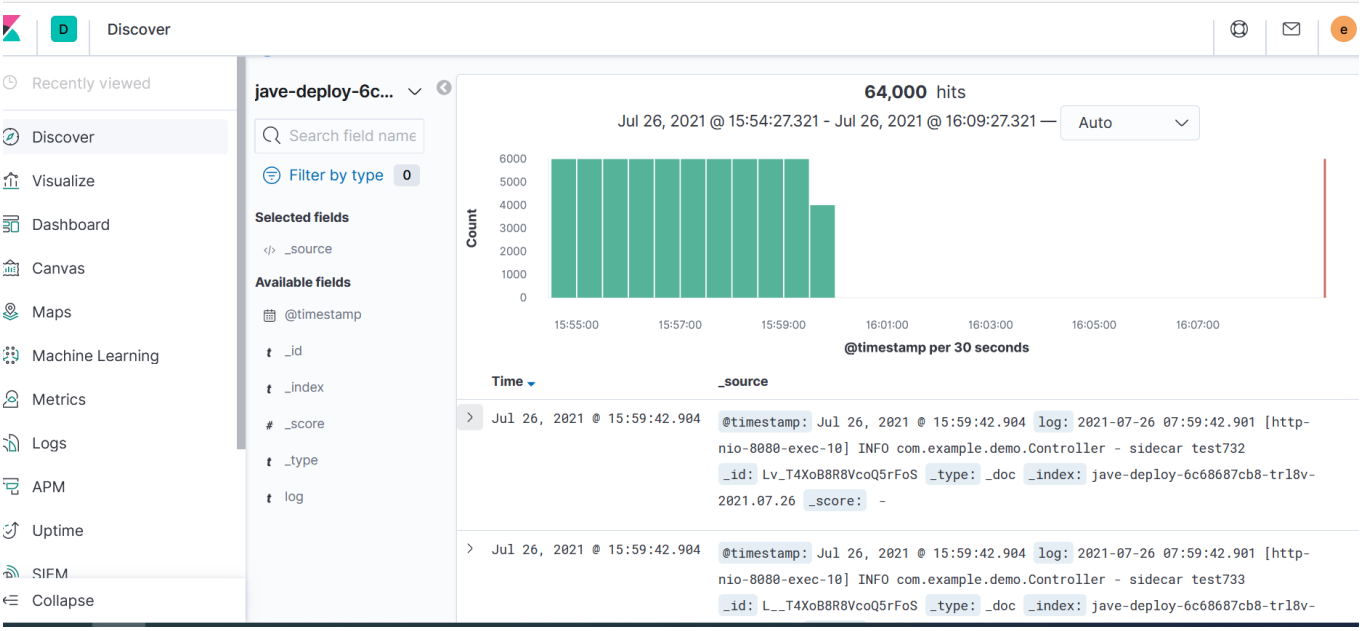
```
首先查看应用的日志是否正常启动
kubectl logs -f pod名称 -n 命名空间 -c 应用容器名称    如下:
kubectl logs -f jave-deploy-58bfc8bd78-swrqq -n fluent -c webapp(应用容器名)

当应用正常启动时查看fluent-bit是否正常
kubectl logs -f jave-deploy-58bfc8bd78-swrqq -n fluent -c fluentbit-sidecar
上面命令执行日志启动正常时，则证明日志从采集到输出均为正常，可以通过kibana查看日志。
```

这里可能会因为fluent-bit配置文件出错或者是deployment中env配置相关参数时出错，
导致应用正常启动，但是日志并不没有被采集到存储点

4. 查看收集的数据

> 通过kubectl get pods -n fluent查看pod全部启动完毕后安照案例1创建索引模板查看收集到的数据



5. 同一个集群调用不同namespace存储

> 当es的namespace与待采集日志的应用之间的namespace不同时，在配置es的host的名称时使用如下：

```
{SERVICE_NAME}.{NAMESPACE_NAME}.svc.cluster.local
```

> 如下配置：

```
env:
- name: FLUENT_ELASTICSEARCH_HOST
  value: "fluent-es3-es-http.fluent.svc.cluster.local"
- name: FLUENT_ELASTICSEARCH_PORT
  value: "9200"
- name: FLUENT_ELASTICSEARCH_USER
  value: "elastic"
- name: FLUENT_ELASTICSEARCH_PWD
  value: "k31c1Ym48nh609EVl77QKaSd"
```

Sidecar采集容器内日志总结

> 通过上面两个简单示例，可以发现这种采集方式都有如下共同点：

1. fluent-bit的configMap配置文件可以共存；也就是说一个config文件可以配置多种类型数据源；
2. 侵入用户应用程序中的fluent-bit配置基本相同；

根据上面的分析，实现一个configMap完成整个项目的配置要求

> 通过上面的分析，可以想到在开发过程中通过一个configMap文件配置fluent-bit，如下面这中配置方式：

```
    input-kubernetes.conf: |
      [INPUT]
          Name        tail
          Path        /data/logs/*.log
          Db          /tmp/biz_log.db
          Db.sync     Full
          Tag         biz-${NODE_NAME}

      [INPUT]
          Name              tail
          Tag               applog.*
          Path              /var/applog/*.log
          Exclude_Path      /var/applog/4.log
          Path_Key          filename
          DB                /var/applog/flb_app.db
          Mem_Buf_Limit     5MB
          Skip_Long_Lines   On
          Refresh_Interval  10

      [INPUT]
          Name              tail
          Tag               kube.*
          Path              /var/log/containers/*.log
          Parser            docker
          DB                /var/log/flb_kube.db
          Mem_Buf_Limit     5MB
          Skip_Long_Lines   On
          Refresh_Interval  10
```

> 在多个input的情况下可以匹配多个output，只需要将output的Match * 改成Match 对应input的Tag名称 + * 即可如：
>
> 匹配上面多个Input

```
      [OUTPUT]
          Name            es
          Match           biz-${NODE_NAME}*
          ....

      [OUTPUT]
          Name            es
          Match           applog.*
          ....
```

> FILTER也可以根据input的Tag过滤相应的数据源，从而指定对应名称的PARSER
>
> fluent-bit的应用侵入是否可以用过通过动态的采取echo将其配置重定向到用户创建的pod配置文件中。

日志采集过程中是否使用fluentd的问题分析

使用fluent-bit的原因：

> fluent-bit更加轻量级相对于fluentd来说，但是他支持的插件相对于fluentd来说较少
> 因此在一些业务需求比较高，对日志处理要求比较高的场景时，需要采用fluent-bit将日志收集转发到fluentd进行处理。

参考：

1. Kubernetes日志采集Sidecar模式介绍
2. fluent-bit官方文档
3. fluentd官方文档