Ingestão de Dados e Backend de Armazenamento

O Priax suporta diferentes backends de armazenamento para Traces, Logs e Métricas coletados e podendo trabalhar com Opensearch, ou Elasticsearch. Para que os dados sejam injetados nessas bases de dados, utilizamos o OpenTelemetry Collector, e pode ser combinado ou não com outras tecnologias de transporte e transformação de dados como o Data Prepper e o Kafka.

Apresentaremos aqui os conceitos básicos do Collector e cenários práticos de implantação para utilização destas tecnologias com o Priax.

Data Collector

O Collector oferece uma implementação independente de fornecedor para receber, processar e exportar dados de telemetria. Ele elimina a necessidade de operar e manter múltiplos agentes/coletores. O Collector funciona com escalabilidade aprimorada e suporta formatos de dados de observabilidade de código aberto (por exemplo, Jaeger, Prometheus, Fluent Bit, etc.), enviando para um ou mais backends de código aberto ou comerciais. O agente Collector local é o destino padrão para o qual as bibliotecas de instrumentação exportam seus dados de telemetria.

Objetivos

- **Usabilidade**: Configuração padrão prática, suporta protocolos populares, funciona e coleta dados imediatamente.
- **Desempenho**: Altamente estável e eficiente sob diferentes cargas e configurações.
- **Observabilidade**: Um exemplo de serviço observável.
- **Extensibilidade**: Personalizável sem necessidade de alterar o código principal.
- **Unificação**: Base de código única, implantável como agente ou coletor, com suporte para rastros, métricas e logs.

Em abbientes de desenvolvimento ou testes e para ter resultados rápido com a observabilidade de maneira pontual é possível enviar seus dados diretamente de sua aplicação para um backend. Em ambientes de produção, no entanto, recomendamos usar um coletor junto com seus serviços, pois ele permite que o serviço descarregue dados rapidamente, realize cache quando necessário e, além disso, o coletor pode lidar com tarefas adicionais, como tentativas de reenvio, agrupamento, criptografia ou até mesmo filtragem de dados sensíveis.

O Collector facilita a unificação de dados, visto que suporta diversas formas de ingestão (Receivers) e diversas formas de exportação de dados (Exporters). Abaixo podemos ver como o Collector trabalha e como ele pode ser utilizado para enviar dados para os Backends de Armazenamento compatíveis com o Priax.

image.png

Como se pode observar o Collector pode receber dados diretamente das aplicações instrumentadas, de outros sistemas de observabilidade como o Prometheus ou o Jaeger mas também pode receber dados de outros Collectors ou buscar dados em sistemas Kafka, que podem ter sido alimentados por outros Collectors. Dessa forma pode-se criar um layout de implantação flexível onde os Collectors podem atuar como gateways concentradores ou como agentes de simples transmissão de dados para os BackEnds de Armazenamento. Para enviar dados para o Backend de armazenamento compatível com o Priax (Opensearch) o Collector pode ou não utilizar o Data Prepper (ver abaixo as informações sobre o DataPrepper).

Para compreender mais sobre os layouts de implantação do Data Collector, consulte a página do OpenTelemetry que trata sobre o assunto.

Instalação e Configuração dos Collectors

A instalação dos Collectors é relativamente simples e pode ser realizada diretamente sobre os sistemas operacionais Windows ou Linux ou ainda utilizando Docker ou Kubernetes. As instruções para instalação podem ser consultadas na página de instalação do Collector do OpenTelemetry.

A Configuração completa do Collector pode ser consultada na extensa documentação fornecida pelo OpenTelemetry porém aqui apresentamos alguns cenários tipicamente utilizados.

Configuração do Collector para uso com e sem Data Prepper

```
receivers:
  otlp:
    protocols:
      http:
        #include_metadata: true
        cors:
          allowed_origins: ["http://*"]
          allowed_headers: ["*"]
          #max_age: 7200
        #endpoint: 127.0.0.1:4318
  kafka:
    protocol_version: 3.3.0
    group_id: priaxapm1
    encoding: otlp_proto
    brokers:
      - priax-teste-01.servicebus.windows.net:9093
    topic: priax
```

```
    auth:
      sasl:
        username: $$ConnectionString
        password: Endpoint=sb://kafkaAzure.servicebus.windows.net/;SharedAccessKeyName=teste-
priax;SharedAccessKey=z5kREHICeAw0EEMySt7cP2kgh4XGb/zh4+AEhMe7r0g=;EntityPath=priax
        mechanism: PLAIN
      tls:
        insecure: true
processors:
  batch: {}
  tail_sampling/priax:
    decision_wait: 40s
    policies:
      - name: sample-ottl
        type: ottl_condition
        ottl_condition:
          span:
            - "status.code == 2 and attributes[\"lime.envelope.type\"] == nil"
            - "status.code == 2 and attributes[\"lime.envelope.type\"] != nil and
attributes[\"lime.envelope.type\"] != \"Notification\""
            - "end_time - start_time > Duration(\"29s\")"
            - "attributes[\"sample.force\"] == true"
      - name: probabilistic-policy
        type: probabilistic
        probabilistic:
          hash_salt: "custom-salt"
          sampling_percentage: 0.1
  k8sattributes:
    extract:
      metadata:
      - k8s.namespace.name
      - k8s.deployment.name
      - k8s.statefulset.name
      - k8s.daemonset.name
      - k8s.cronjob.name
      - k8s.job.name
      - k8s.node.name
      - k8s.pod.name
      - k8s.pod.uid
      - k8s.pod.start_time
```

```
    passthrough: false
    pod_association:
    - sources:
      - from: resource_attribute
        name: k8s.pod.ip
    - sources:
      - from: resource_attribute
        name: k8s.pod.uid
    - sources:
      - from: connection
  memory_limiter:
    check_interval: 5s
    limit_percentage: 80
    spike_limit_percentage: 25
  resource:
    attributes:
    - action: insert
      from_attribute: k8s.pod.uid
      key: service.instance.id
exporters:
  debug:
    verbosity: detailed
  otlp/data-prepper:
    endpoint: data-prepper:21890
    tls:
      insecure: true
  opensearch:
    http:
    endpoint: https://opensearchserver:9200
    auth:
      authenticator: basicauth/client
extensions:
  health_check:
  pprof:
  zpages:
    endpoint: ":55679"
  basicauth/client:
    client_auth:
      username: ingester
      password: ingesterpass
```

```
service:
  extensions: [health_check, pprof, zpages]
  pipelines:
    traces:
      receivers: [otlp]
      processors: [k8sattributes,memory_limiter,resource,batch,tail_sampling/priax]
      exporters: [debug,opensearch]
    metrics:
      receivers: [otlp]
      processors: [k8sattributes,memory_limiter,resource,batch]
      exporters: [debug,opensearch]
    logs:
      receivers: [otlp]
      processors: [k8sattributes,memory_limiter,resource,batch]
      exporters: [debug,opensearch]
    traceswithKafkaandDataprepper:
      receivers: [kafka]
      processors: [k8sattributes,memory_limiter,resource,batch]
      exporters: [debug,otlp/dataprepper]
    metricswithKafkaandDataprepper:
      receivers: [kafka]
      processors: [k8sattributes,memory_limiter,resource,batch]
      exporters: [debug,otlp/dataprepper]
    logswithKafkaandDataprepper:
      receivers: [kafka]
      processors: [k8sattributes,memory_limiter,resource,batch]
      exporters: [debug,otlp/dataprepper]
```

Amostragem de Traces e Spans

É importante perceber que a configuração acima define que serão enviadas ao Backend de Armazenamento o 0,1% das spans coletadas pela instrumentação. Isso se dá nos trechos destacados abaixo:

```
[...]
processors:
[...]
  tail_sampling/priax:
    decision_wait: 40s
    policies:
```

```
      - name: sample-ottl
        type: ottl_condition
        ottl_condition:
          span:
            - "end_time - start_time > Duration(\"29s\")"
            - "attributes[\"sample.force\"] == true"
      - name: probabilistic-policy
        type: probabilistic
        probabilistic:
          hash_salt: "custom-salt"
          sampling_percentage: 0.1
[...]
pipelines:
  traces:
      receivers: [otlp]
      processors: [k8sattributes,memory_limiter,resource,batch,tail_sampling/priax]
      # Para não utilizar o sampling neste pipeline basta rever o processador
tail_sampling/priax desta seção.
      exporters: [debug,opensearch]
    traces:
      receivers:
        - kafka
      processors:
        - batch
        - tail_sampling/priax
      exporters:
        - otlp/data-prepper
[...]
```

Para que sejam enviados 100% dos traces para o Backend de Armazenamento, remova a configuração de sampling do pipeline desejado conforme comentado no exemplo acima.

Data Prepper

Data Prepper
O Data Prepper é um componente da infraestrutura do Opensearch. É um coletor de dados do lado do servidor, capaz de filtrar, enriquecer, transformar, normalizar e agregar dados para análise e visualização posteriores. É a ferramenta de ingestão de dados preferida para o OpenSearch, recomendada para a maioria dos casos de uso de ingestão de dados no OpenSearch, especialmente para o processamento de conjuntos de dados grandes e complexos.

Com o Data Prepper, você pode criar pipelines personalizados para melhorar a visão operacional de suas aplicações. Dois casos de uso comuns do Data Prepper são a análise de rastros (trace analytics) e a análise de logs (log analytics). A análise de rastros ajuda a visualizar fluxos de eventos e identificar problemas de desempenho. Já a análise de logs oferece ferramentas para aprimorar as capacidades de busca, realizar análises detalhadas e obter insights sobre o desempenho e o comportamento das suas aplicações.

Conceitos-chave e fundamentos
O Data Prepper processa dados por meio de pipelines personalizáveis. Esses pipelines são compostos por componentes modulares que podem ser ajustados para atender às suas necessidades, inclusive permitindo a integração de implementações próprias. Um pipeline do Data Prepper consiste nos seguintes componentes:

- **Uma fonte (source)**
- **Um ou mais destinos (sinks)**
- **(Opcional) Um buffer**
- **(Opcional) Um ou mais processadores (processors)**

Cada pipeline contém dois componentes obrigatórios: a **fonte** e o **destino**. Se um buffer, um processador, ou ambos estiverem ausentes do pipeline, o Data Prepper usará o buffer padrão `bounded_blocking` e um processador `no-op`. É importante observar que uma única instância do Data Prepper pode conter um ou mais pipelines.

O uso do Data Prepper com o Priax, facilita a adoção de padrões do Opensearch e é altamente recomendável.

Instalação do Data Prepper

A instalação do Data Prepper é relativamente simples e sua documentação pode ser consultada diretamente na página do OpenSearch.

Configurações básicas de pipeline

A configuração básica do Data Prepper para uso com o Priax pode ser vista abaixo:

```
entry-pipeline:
  delay: "100"
  source:
    otel_trace_source:
      ssl: false
    buffer:
      buffer_size: 10240
      batch_size: 160
  sink:
    - pipeline:
```

```
        name: "raw-trace-pipeline"
    - pipeline:
        name: "service-map-pipeline"
raw-trace-pipeline:
  source:
    pipeline:
      name: "entry-pipeline"
  buffer:
    bounded_blocking:
      buffer_size: 10240
      batch_size: 160
  processor:
    - otel_trace_raw:
  sink:
    - stdout: null
    - opensearch:
        hosts: ["http://opensearchserver:9200"]
        username: ingester
        password: passwordingester
        max_retries: 20
        bulk_size: 4
        insecure: true
        index_type: trace-analytics-raw
service-map-pipeline:
  delay: "100"
  source:
    pipeline:
      name: "entry-pipeline"
  buffer:
    bounded_blocking:
      buffer_size: 10240
      batch_size: 160
  processor:
    - service_map_stateful:
  sink:
    - stdout: null
    - opensearch:
        hosts: ["http://opensearchserver:9200"]
        username: ingester
        password: passwordingester
```

```
        max_retries: 20
        bulk_size: 4
        insecure: true
        index_type: trace-analytics-service-map
```

Revision #12
Created 2024-08-16 14:08:58 UTC by Wagner B. Simonato
Updated 2024-09-10 23:57:57 UTC by Wagner B. Simonato