Sinais e Métricas das Aplicações

Tipos de Sinais

Conheça as categorias de telemetria suportadas pelo Priax e bibliotecas de instrumentação compatíveis:

O objetivo de uma biblioteca de instrumentação é **coletar, processar e exportar sinais**. Sinais são **saídas do sistema** que descrevem a atividade subjacente do sistema operacional e das aplicações em execução em uma plataforma. Um **sinal** pode representar algo que você deseja medir em um ponto específico no tempo, como a **temperatura** ou o **uso de memória**, ou um evento que percorre os componentes de um sistema distribuído e que você gostaria de rastrear.

Com o Priax você pode **agrupar diferentes sinais** para observar o funcionamento interno de uma mesma tecnologia sob diferentes perspectivas.

Atualmente, o **Priax** é capaz de exibir, detalhar, quantificar e detalhar as seguintes categorias de sinais:

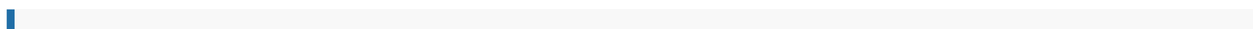
- **Rastreamentos** (*traces*)
- **Métricas**
- **Logs**
- **Baggage**

Rastreamentos (Traces)

O caminho de uma solicitação através da sua aplicação.

Os **rastreamentos** nos dão uma **visão geral** do que acontece quando uma solicitação é feita a uma aplicação. Seja sua aplicação um monólito com um único banco de dados ou uma complexa malha de serviços, **os rastreamentos são essenciais para entender o “caminho” completo que uma solicitação percorre** na sua aplicação.

Vamos explorar isso com três unidades de trabalho representadas como **Spans** (faixas de execução):



> **Observação**
> Os exemplos de JSON a seguir não representam um formato específico, especialmente o **OTLP/JSON**, que é mais detalhado.

1. Span Hello:

```
{
  "name": "hello",
  "context": {
    "trace_id": "5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "051581bf3cb55c13"
  },
  "parent_id": null,
  "start_time": "2022-04-29T18:52:58.114201Z",
  "end_time": "2022-04-29T18:52:58.114687Z",
  "attributes": {
    "http.route": "some_route1"
  },
  "events": [
    {
      "name": "Guten Tag!",
      "timestamp": "2022-04-29T18:52:58.114561Z",
      "attributes": { "event_attributes": 1 }
    }
  ]
}
```

Este é o **span raiz**, indicando o início e o fim da operação inteira. Observe que ele tem um campo **trace_id**, mas **não tem parent_id**, o que o caracteriza como a raiz do rastreamento.

2. Span *hello-greetings:*

```
{
  "name": "hello-greetings",
  "context": {
    "trace_id": "5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "5fb397be34d26b51"
  },
  "parent_id": "051581bf3cb55c13",
```

```
  "start_time": "2022-04-29T18:52:58.114304Z",
  "end_time": "2022-04-29T22:52:58.114561Z",
  "attributes": {
    "http.route": "some_route2"
  },
  "events": [
    {
      "name": "hey there!",
      "timestamp": "2022-04-29T18:52:58.114561Z",
      "attributes": {
        "event_attributes": 1
      }
    },
    {
      "name": "bye now!",
      "timestamp": "2022-04-29T18:52:58.114585Z",
      "attributes": {
        "event_attributes": 1
      }
    }
  ]
}
```

Este span encapsula tarefas específicas, como exibir saudações, e seu **parent_id** é o ID do span raiz *hello*. Ele compartilha o mesmo **trace_id**, indicando que faz parte do mesmo rastreamento.

3. Span *hello-salutations*:

```
{
  "name": "hello-salutations",
  "context": {
    "trace_id": "5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "93564f51e1abe1c2"
  },
  "parent_id": "051581bf3cb55c13",
  "start_time": "2022-04-29T18:52:58.114492Z",
  "end_time": "2022-04-29T18:52:58.114631Z",
  "attributes": {
    "http.route": "some_route3"
  },
  "events": [
```

```
    {
      "name": "hey there!",
      "timestamp": "2022-04-29T18:52:58.114561Z"
    }
  ]
}
```

Este span representa a **terceira operação** neste rastreamento e, assim como o anterior, é um **filho do span raiz (*hello*)**. Isso também o torna um **irmão** do span *hello-greetings*

Esses três blocos de JSON compartilham o mesmo **trace_id** e utilizam o **parent_id** para representar a hierarquia. Isso cria o **rastreamento completo**!

Uma observação importante é que cada **Span** se assemelha a um **log estruturado**, com contexto, correlação e hierarquia embutidos. Contudo, esses “logs estruturados” podem vir de diferentes **processos, serviços, VMs ou datacenters**, permitindo que o rastreamento represente uma **visão ponta a ponta** de qualquer sistema.

Detalhamento de uma span

Uma **span** representa uma **unidade de trabalho** ou operação e inclui:

- **Nome**
- **ID do Span pai** (vazio para spans raiz)
- **Timestamps de início e fim**
- **Contexto do Span**
- **Atributos**
- **Eventos**
- **Links**
- **Status do Span**

Exemplo de span:

```
{
  "name": "/v1/sys/health",
  "context": {
    "trace_id": "7bba9f33312b3dbb8b2c2c62bb7abe2d",
    "span_id": "086e83747d0e381e"
  },
  "parent_id": "",
  "start_time": "2021-10-22 16:04:01.209458162 +0000 UTC",
  "end_time": "2021-10-22 16:04:01.209514132 +0000 UTC",
  "status_code": "STATUS_CODE_OK",
  "status_message": "",
```

```
  "attributes": {
    "net.transport": "IP.TCP",
    "net.peer.ip": "172.17.0.1",
    "net.peer.port": "51820",
    "net.host.ip": "10.177.2.152",
    "net.host.port": "26040",
    "http.method": "GET",
    "http.target": "/v1/sys/health",
    "http.server_name": "mortar-gateway",
    "http.route": "/v1/sys/health",
    "http.user_agent": "Consul Health Check",
    "http.scheme": "http",
    "http.host": "10.177.2.152:26040",
    "http.flavor": "1.1"
  },
  "events": [
    {
      "name": "",
      "message": "OK",
      "timestamp": "2021-10-22 16:04:01.209512872 +0000 UTC"
    }
  ]
}
```

Contexto do Span

Os spans podem ser aninhados, como é indicado pela presença de um **ID de span pai**: os spans filhos representam suboperações. Isso permite que os spans capturem com mais precisão o trabalho realizado em uma aplicação. O span pai terá um trace_Id idêntico ao seu filho indicando portanto que atende a uma mesma requisição ao sistema.

O contexto do span é um objeto imutável presente em todos os spans que contém as seguintes informações:

- **Trace ID**: identifica o trace ao qual o span pertence.
- **Span ID**: identifica o próprio span.
- **Trace Flags**: uma codificação binária com informações sobre o trace.
- **Trace State**: uma lista de pares chave-valor que podem conter informações específicas de fornecedores.

Atributos

Os atributos são pares chave-valor usados para adicionar metadados a um span, permitindo capturar informações sobre a operação monitorada.

Por exemplo, se um span acompanha a operação de adicionar um item ao carrinho de compras em um sistema de e-commerce, você pode registrar o **ID do usuário**, o **ID do item** e o **ID do carrinho**.

- Você pode adicionar atributos durante ou após a criação de um span.
- É recomendável adicionar os atributos no momento da criação para que possam ser aproveitados pelo SDK na amostragem. Caso precise adicionar valores posteriormente, atualize o span.

Regras para atributos:

- As **chaves** devem ser strings não nulas.
- Os **valores** devem ser uma string, booleano, número (ponto flutuante ou inteiro) ou um array desses tipos.

Além disso, existem **Atributos Semânticos**, que são convenções padronizadas para nomes de atributos comuns. Utilizar nomes semânticos facilita a padronização de metadados entre sistemas.

Eventos do Span

Um **evento do span** pode ser visto como uma **mensagem de log estruturada** (ou anotação) associada a um span. Geralmente é usado para marcar um ponto singular e significativo no tempo durante a duração de um span.

Exemplo prático em um navegador web:

1. **Carregamento da página** → O span é mais apropriado, pois acompanha uma operação com início e fim.
2. **Página se torna interativa** → Um evento do span é mais adequado, pois representa um ponto específico no tempo.

Quando usar eventos do span versus atributos?
Se o **timestamp** específico for relevante, use um **evento do span**. Caso contrário, use **atributos**.

Links do Span

Os **links** permitem associar um span a um ou mais spans, indicando uma relação causal.

Por exemplo, em um sistema distribuído, algumas operações são rastreadas por um trace. Em resposta a essas operações, uma nova tarefa pode ser enfileirada para execução assíncrona. Para associar o trace da operação subsequente ao trace original, criamos um **link de span**.

Links são opcionais, mas são uma maneira eficaz de relacionar spans de diferentes traces.

Status do Span

Cada span possui um status com três valores possíveis:

1. **Unset**: (padrão) indica que a operação foi concluída sem erros.
2. **Error**: indica que ocorreu algum erro, como um HTTP 500 em um servidor.
3. **Ok**: explicitamente definido pelo desenvolvedor para marcar o span como bem-sucedido.

> **Nota:** Quando não houver um status **Ok** para spans que concluíram sem erros, considera-se um evento bem sucedido, pois o valor padrão **Unset** já cobre esse caso. **Ok** é usado para deixar explícito que um span é considerado bem-sucedido, sem ambiguidades.

Tipo do Span (Span Kind)

Ao criar um span, ele pode ser classificado como **Client**, **Server**, **Internal**, **Producer** ou **Consumer**. O tipo de span informa ao backend de rastreamento como o trace deve ser montado.

- **Client**: Representa uma chamada remota síncrona de saída, como uma solicitação HTTP ou consulta ao banco de dados.
- **Server**: Representa uma chamada remota síncrona recebida, como uma solicitação HTTP ou RPC.
- **Internal**: Representa operações que não cruzam limites de processo (ex: chamadas de funções locais ou middlewares).
- **Producer**: Representa a criação de uma tarefa a ser processada de forma assíncrona (ex: inserção em uma fila).
- **Consumer**: Representa o processamento de uma tarefa criada por um producer. Pode ser iniciado muito tempo após o término do span do producer.

Se o tipo não for especificado, ele é considerado **Internal**.

Métricas

Uma métrica é uma medida de um serviço capturada em tempo de execução. O momento em que a medição é registrada é conhecido como um **evento de métrica**, que inclui não apenas o valor medido, mas também o horário da captura e os metadados associados.

Métricas de aplicação e requisição são indicadores importantes de disponibilidade e desempenho. Métricas personalizadas podem oferecer insights sobre como os indicadores de disponibilidade impactam a experiência do usuário ou o negócio. Os dados coletados podem ser utilizados para alertar sobre interrupções ou para acionar decisões de escalonamento automático durante períodos de alta demanda.

Para compreender como as métricas funcionam quando se utiliza as bibliotecas de instrumentação compatíveis com o Priax, vamos analisar os componentes principais que ajudam na instrumentação do código.

Provedor de Meter (Meter Provider)

O **Meter Provider** é uma fábrica de **Meters**. Em aplicações, geralmente é inicializado uma única vez e seu ciclo de vida corresponde ao ciclo de vida da aplicação. A inicialização de um Meter Provider também envolve a configuração de **Recursos** e **Exportadores**. Esse é, normalmente, o primeiro passo ao configurar métricas no Priax. Em alguns SDKs compatíveis, um Meter Provider global já é inicializado automaticamente.

Meter

Um **Meter** é responsável por criar instrumentos de métrica que capturam medições sobre um serviço em tempo de execução. Os **Meters** são criados a partir de **Meter Providers**.

Exportador de Métricas (Metric Exporter)

Os **Exportadores de Métricas** enviam dados de métricas para um consumidor. Esse consumidor pode ser uma saída padrão (para depuração durante o desenvolvimento), o **Priax utiliza o Opentelemetry Collector**, ou qualquer backend de código aberto ou proprietário de sua escolha.

Instrumentos de Métrica (Metric Instruments)

No Priax, as medições são capturadas por **instrumentos de métrica**, definidos pelos seguintes atributos:

- **Nome**
- **Tipo**
- **Unidade** (opcional)
- **Descrição** (opcional)

O nome, a unidade e a descrição são definidos pelo desenvolvedor ou por convenções semânticas para métricas comuns, como requisições e processos.

Os **tipos de instrumentos compatíveis** incluem:

- **Counter**: Valor acumulado ao longo do tempo (ex.: um odômetro, que só aumenta).
- **Counter Assíncrono**: Similar ao Counter, mas coletado uma vez por exportação, útil quando você tem acesso apenas ao valor agregado.
- **UpDownCounter**: Valor acumulado que pode aumentar ou diminuir (ex.: o tamanho de uma fila).
- **UpDownCounter Assíncrono**: Similar ao UpDownCounter, mas coletado uma vez por exportação.
- **Gauge**: Mede um valor atual no momento da leitura (ex.: marcador de combustível de um carro). É síncrono.

- **Gauge Assíncrono**: Similar ao Gauge, mas coletado uma vez por exportação.
- **Histogram**: Agrega valores no lado do cliente, como latências de requisições. Útil para estatísticas de valores, como quantas requisições levaram menos de 1s.

Agregação

Além dos instrumentos de métrica, a **agregação** é um conceito importante. A agregação combina uma grande quantidade de medições em estatísticas exatas ou estimadas sobre eventos de métrica em uma janela de tempo. O protocolo OTLP transporta essas métricas agregadas.

A API do Priax fornece uma agregação padrão para cada instrumento, que pode ser personalizada usando **Views**. O objetivo do projeto Priax é fornecer agregações padrão que sejam compatíveis com ferramentas de visualização e backends de telemetria.

Diferentemente do rastreamento de requisições, que captura o ciclo de vida de uma requisição, as métricas fornecem informações estatísticas em agregados. Exemplos de uso incluem:

- Total de bytes lidos por um serviço, por tipo de protocolo.
- Total de bytes lidos e bytes por requisição.
- Duração de chamadas de sistema.
- Tamanhos de requisição para identificar tendências.
- Uso de CPU ou memória de um processo.
- Valores médios de saldo em contas.
- Quantidade atual de requisições ativas.

Views

As **Views** oferecem flexibilidade para personalizar a saída de métricas pelo SDK. É possível:

- Escolher quais instrumentos de métrica serão processados ou ignorados.
- Configurar a agregação e os atributos a serem reportados nas métricas.

Suporte por Linguagem

O suporte às implementações específicas da API e do SDK de Métricas, por linguagem, está conforme a tabela abaixo:

Linguagem	Status
C++	Estável
C#/.NET	Estável
Erlang/Elixir	Em desenvolvimento

Go	Estável
Java	Estável
JavaScript	Estável
PHP	Estável
Python	Estável
Ruby	Em desenvolvimento
Rust	Beta
Swift	Em desenvolvimento

Logs

Um **log** é um registro textual com marcação de tempo, podendo ser estruturado (recomendado) ou não estruturado, acompanhado de metadados opcionais. Entre todos os sinais de telemetria, os logs possuem o maior legado, já que a maioria das linguagens de programação inclui capacidades de logging nativas ou bibliotecas amplamente utilizadas.

Logs de Aplicação no Priax

O Priax não define uma API ou SDK específicos para criar logs. Em vez disso, os logs no Priax são aqueles já existentes, oriundos de frameworks de logging ou componentes de infraestrutura. Os SDKs e bibliotecas de instrumentação compatíveis com o Priax e a autoinstrumentação utilizam diversos componentes para correlacionar automaticamente logs com traces.

O suporte da biblioteca de instrumentação Priax para logs é projetado para ser totalmente compatível com sistemas existentes, oferecendo ferramentas para adicionar contexto adicional aos logs e manipulá-los em um formato comum, independentemente da origem.

Nas aplicações, os logs são criados usando qualquer biblioteca ou funcionalidade nativa de logging. Ao adicionar autoinstrumentação ou ativar um SDK de instrumentação compatível com o Priax, os logs são correlacionados automaticamente existentes com quaisquer traces e spans ativos, encapsulando o corpo do log com seus respectivos IDs.

Suporte por Linguagem

As bibliotecas de instrumentação atualmente compatíveis com o Priax suportam o envio de logs para as seguintes linguagens de programação:

Linguagem	Status
C++	Estável
C#/.NET	Estável
Erlang/Elixir	Em desenvolvimento
Go	Beta
Java	Estável
JavaScript	Em desenvolvimento
PHP	Estável
Python	Em desenvolvimento
Ruby	Em desenvolvimento
Rust	Beta
Swift	Em desenvolvimento

Logs Estruturados, Não Estruturados e Semi-Estruturados

Logs Estruturados

Logs estruturados seguem um formato consistente e legível por máquina, como JSON.

Exemplo em aplicação:

```
{
  "timestamp": "2024-08-04T12:34:56.789Z",
  "level": "INFO",
  "service": "autenticacao-usuario",
  "mensagem": "Login do usuário bem-sucedido",
  "contexto": {
    "userId": "12345",
    "ipAddress": "192.168.1.1"
  }
}
```

Logs Não Estruturados

Logs não estruturados não seguem um formato consistente. Apesar de serem mais legíveis para humanos, são difíceis de processar em larga escala.

Exemplo:
`[ERRO] 2024-08-04 12:45:23 - Falha ao conectar ao banco de dados. Timeout.`

Logs Semi-Estruturados

Logs semi-estruturados utilizam padrões consistentes, mas com formatações variadas entre sistemas.

Exemplo:
`2024-08-04T12:45:23Z level=ERROR service=autenticacao-usuario action=login mensagem="Senha inválida"`

Baggage

Informações contextuais propagadas entre sinais.

O **Baggage** representa informações contextuais que são mantidas junto ao contexto. O Baggage é uma estrutura de chave-valor que permite a propagação de dados arbitrários ao lado do contexto.

Com o Baggage, é possível transmitir dados entre serviços e processos, tornando essas informações acessíveis para serem adicionadas a traces, métricas ou logs nos serviços subsequentes.

Para que serve o Baggage

O Baggage é ideal para incluir informações que estão disponíveis no início de uma requisição, mas que precisam ser acessadas em estágios posteriores. Exemplos incluem:

- Identificação de Contas
- IDs de Usuários
- IDs de Produtos
- IPs de origem

Propagar essas informações usando o Baggage possibilita análises mais detalhadas na telemetria. Por exemplo, incluir um ID de Usuário em um span que monitora uma chamada ao banco de dados facilita responder perguntas como: *“Quais usuários estão enfrentando as chamadas mais lentas ao banco de dados?”*. Também é possível registrar informações de uma operação downstream e incluir o mesmo ID de Usuário nos dados de log.

Considerações de segurança do Baggage

Itens sensíveis no Baggage podem ser compartilhados com recursos não intencionais, como APIs de terceiros. Isso ocorre porque instrumentações automáticas incluem o Baggage na maioria das requisições de rede do seu serviço.

Especificamente, o Baggage e outras partes do contexto de trace são enviados em cabeçalhos HTTP, ficando visíveis para quem inspecionar o tráfego da rede. Se o tráfego estiver restrito à sua rede interna, esse risco pode não se aplicar, mas é importante lembrar que serviços downstream podem propagar o Baggage para fora da sua rede.

Além disso, não há verificações de integridade embutidas para garantir que os itens do Baggage sejam de sua origem. Por isso, tome cuidado ao acessá-los.

O Baggage não é o mesmo que atributos

Uma diferença importante é que o Baggage é um armazenamento separado de chave-valor, e não está associado diretamente aos atributos de spans, métricas ou logs, a menos que seja adicionado explicitamente.

Para adicionar entradas do Baggage como atributos, é necessário ler os dados do Baggage e incluí-los manualmente como atributos em spans, métricas ou logs.

Como um dos usos comuns do Baggage é adicionar dados como atributos de spans em um trace inteiro, várias linguagens oferecem **Baggage Span Processors** que automaticamente adicionam dados do Baggage como atributos na criação de spans.

Revision #5
Created 2024-08-16 00:49:19 UTC by Wagner B. Simonato
Updated 2024-09-10 23:57:57 UTC by Wagner B. Simonato