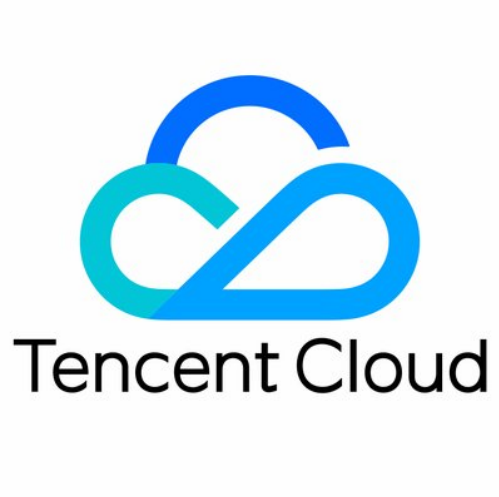
Application Performance Management Best Practices

Product Documentation



Contents

Best Practices
Configuring Client Sampling with Jaeger

Last updated：2023-12-25 15:52:41

This document describes how to configure client sampling with Jaeger.

Overview

When the number of access requests is high, reporting all trace data may greatly increase APM fees. In this case, data sampling is often used.

Note:

In sampling, certain data is sampled from all the collected trace data for analysis, which reduces the span volume and trace storage fees.

Sampling

Take a simple call relationship as an example: A > B > C (service A calls service B that calls service C). If service A doesn't receive any tracing information when called, its Jaeger library will create a trace, assign a trace ID, and decide whether to save the trace based on the sampling configuration. Both the sampling configuration decision and request will be sent to services B and C; therefore, you only need to configure sampling for service A.

Directions

Sampling policy

The Jaeger client supports four sampling policies as follows:

Constant (sampler.type=const): It samples either all or none traces when the sample rate is `1` or `0`.

Probabilistic (sampler.type=probabilistic): It makes a random sampling decision with the probability in the range of 0–1. For example, `0.5` indicates to sample 50% traces.

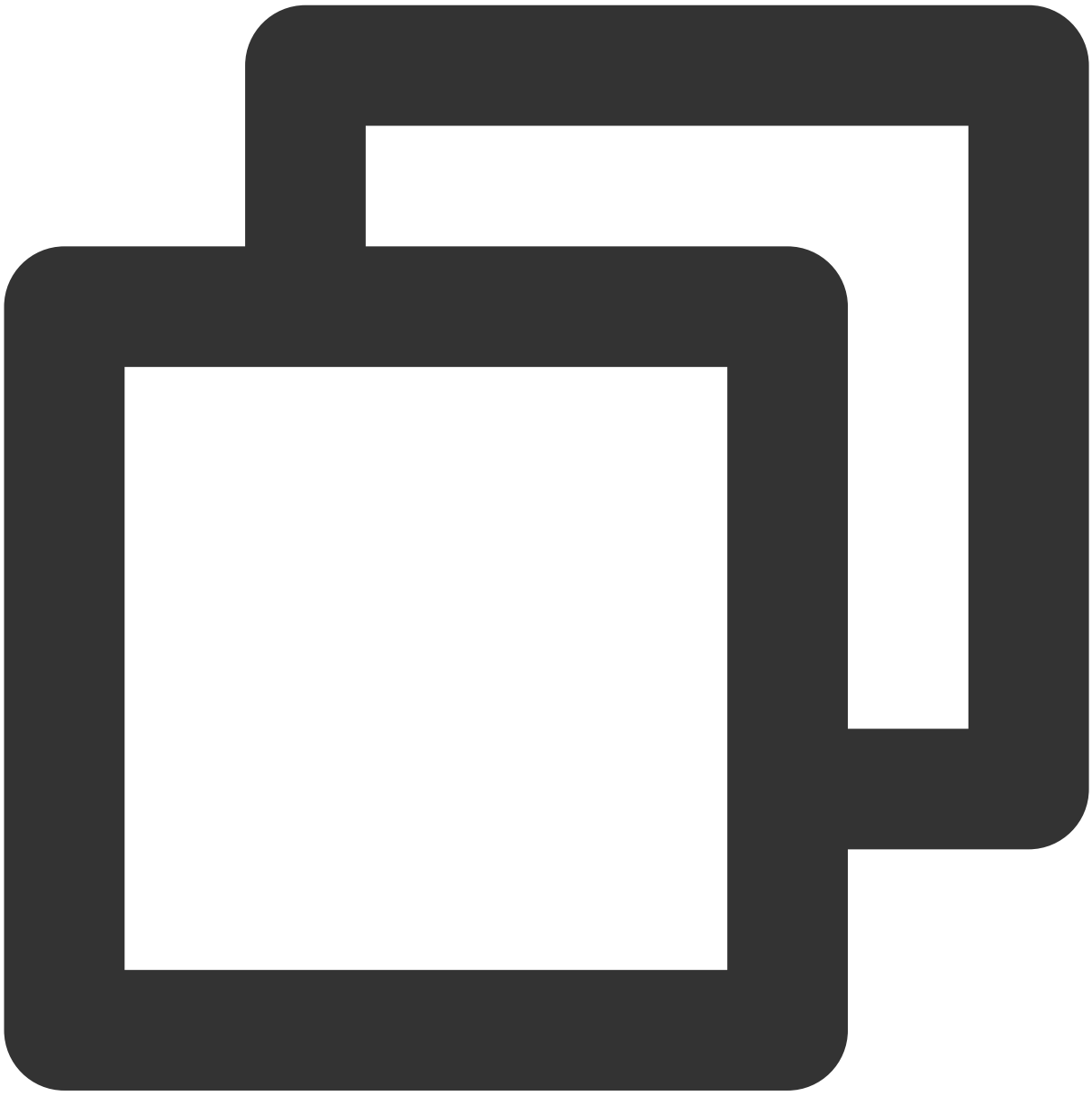
Rate Limiting (sampler.type=ratelimiting): It uses a rate limiter to ensure that traces are sampled with a certain constant rate. For example, `sampler.param = 2.0` indicates to sample requests with the rate of two traces per second.

Remote (sampler.type=remote): It is the default policy. It resembles `probabilistic` as the sampling probability but allows for dynamically getting the sample rate settings from the Jaeger agent. To minimize costs,

Jaeger adopts the 0.1% sampling policy, i.e., sampling 1 in 1,000 traces.

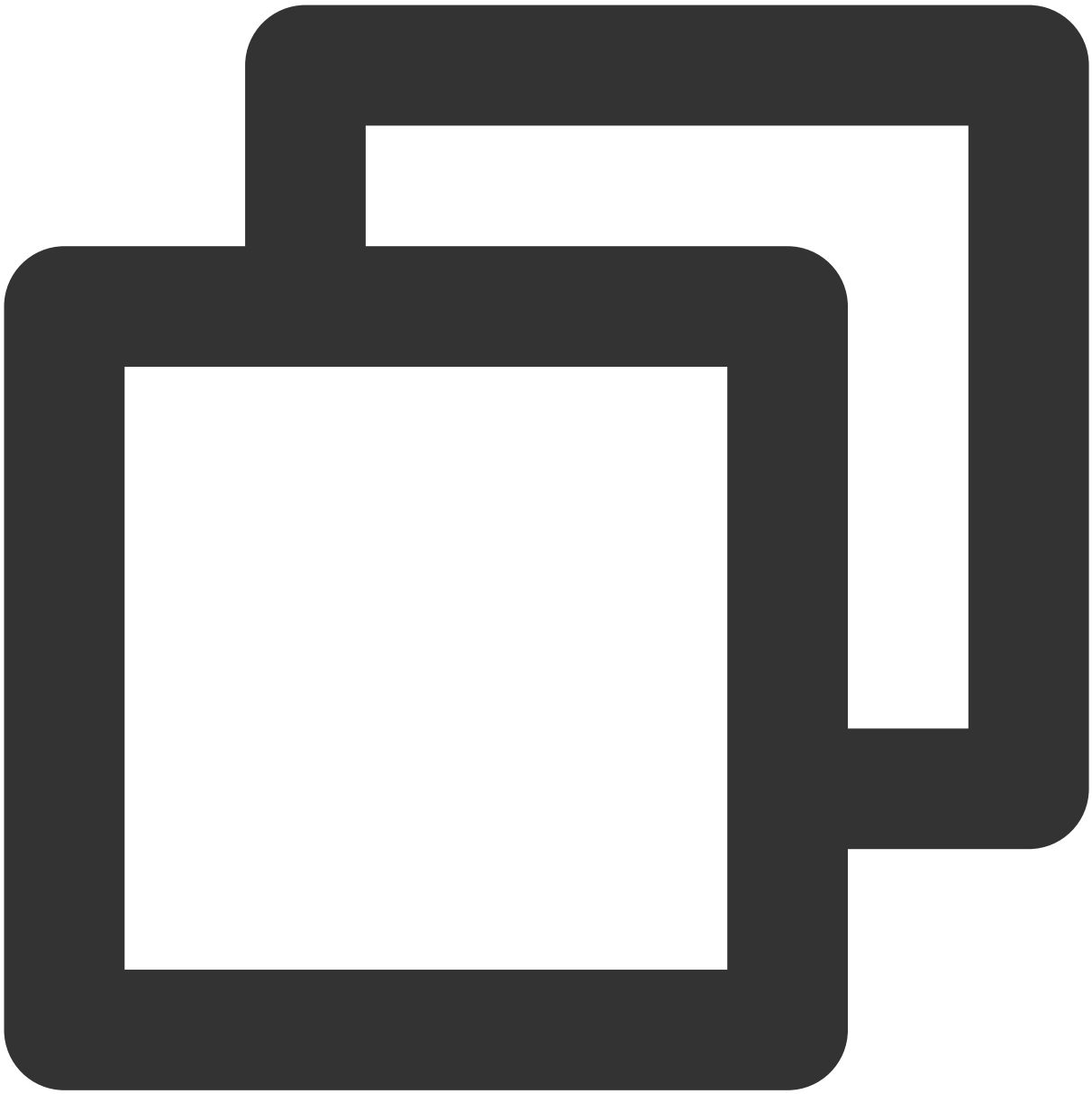
Java sample

1. Add the Jaeger library to the dependencies.



```
<dependency>
  <groupId>io.jaegertracing</groupId>
  <artifactId>jaeger-client</artifactId>
  <version>0.32.0</version>
</dependency>
```

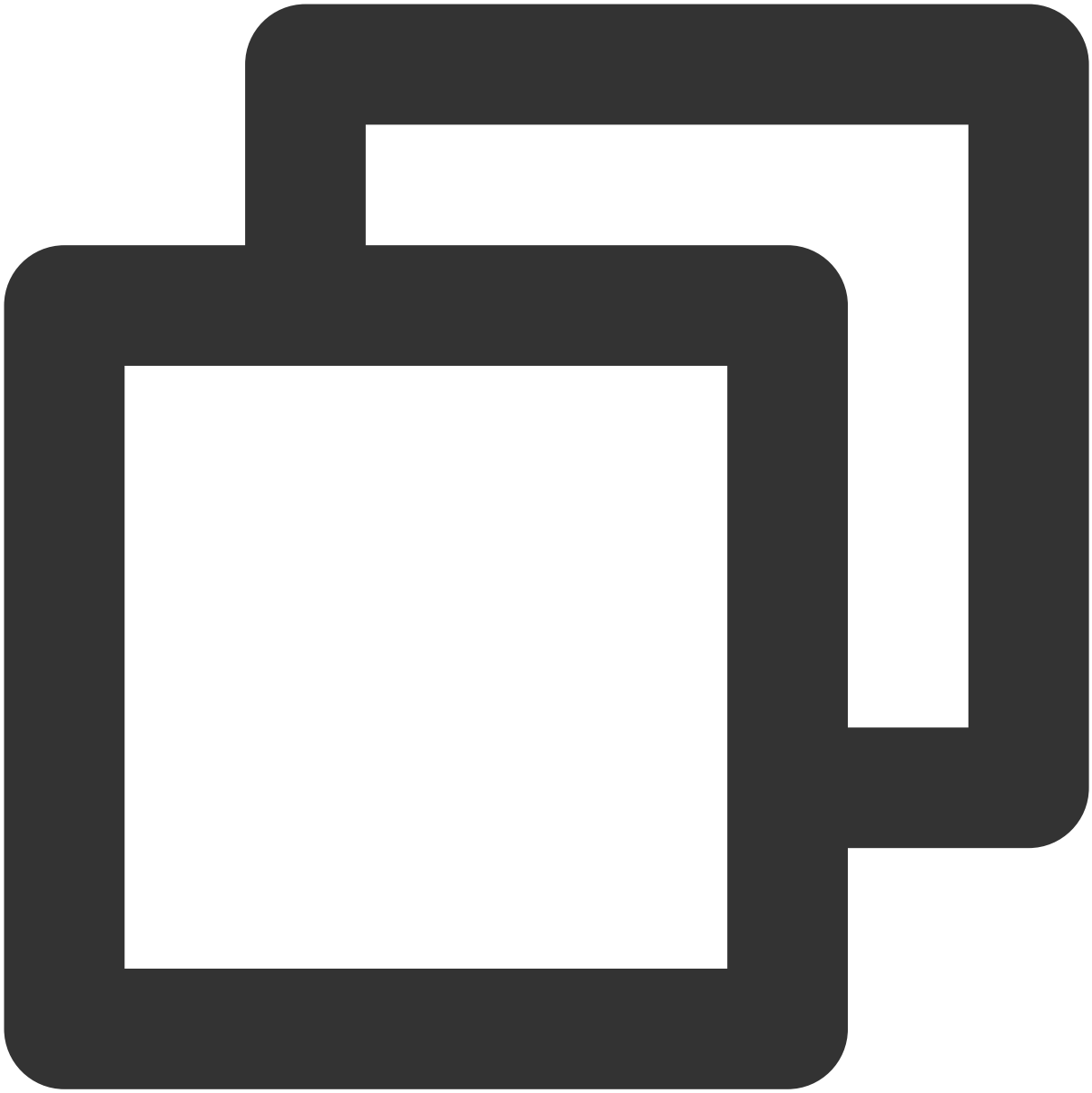
2. Below is the sample code:



```
import io.jaegertracing.Configuration;
import io.jaegertracing.Configuration.ReporterConfiguration;
import io.jaegertracing.Configuration.SamplerConfiguration;
import io.jaegertracing.internal.JaegerTracer;
import io.jaegertracing.internal.samplers.ConstSampler;
import io.opentracing.Span;
import io.opentracing.util.GlobalTracer;

...
```

```
SamplerConfiguration samplerConfig = SamplerConfiguration.fromEnv()
  .withType(ConstSampler.TYPE)
  .withParam(1);

ReporterConfiguration reporterConfig = ReporterConfiguration.fromEnv()
  .withLogSpans(true);

Configuration config = new Configuration("helloWorld")
  .withSampler(samplerConfig)
  .withReporter(reporterConfig);

GlobalTracer.register(config.getTracer());

...
Span parent = GlobalTracer.get().buildSpan("hello").start();
try (Scope scope = GlobalTracer.get().scopeManager()
      .activate(parent)) {
    Span child = GlobalTracer.get().buildSpan("world")
            .asChildOf(parent).start();
    try (Scope scope = GlobalTracer.get().scopeManager()
       .activate(child)) {
    }
}
```

Go sample



```
import (
    "github.com/opentracing/opentracing-go"
    "github.com/uber/jaeger-client-go"
    "github.com/uber/jaeger-client-go/config"
)

...

func main() {
    ...
    cfg := config.Configuration{
```

```
    Sampler: &config.SamplerConfig{
        Type:  "const",
        Param: 1,
    },
    Reporter: &config.ReporterConfig{
        LogSpans:            true,
        BufferFlushInterval: 1 * time.Second,
    },
    }
    tracer, closer, err := cfg.New(
        "your_service_name",
        config.Logger(jaeger.StdLogger),
    )
    opentracing.SetGlobalTracer(tracer)
    defer closer.Close()

    someFunction()
    ...
}

...

func someFunction() {
    parent := opentracing.GlobalTracer().StartSpan("hello")
    defer parent.Finish()
    child := opentracing.GlobalTracer().StartSpan(
    "world", opentracing.ChildOf(parent.Context()))
    defer child.Finish()
}
```

Note:

For more samples, see Client Library Features.

Configuring Java Application Data Collection with SkyWalking

Last updated：2023-12-25 15:58:14

This document describes how to configure Java application data collection with SkyWalking.

Overview

When the number of access requests is high, reporting all trace data may greatly increase APM fees. In this case, data sampling is often used.

Note:

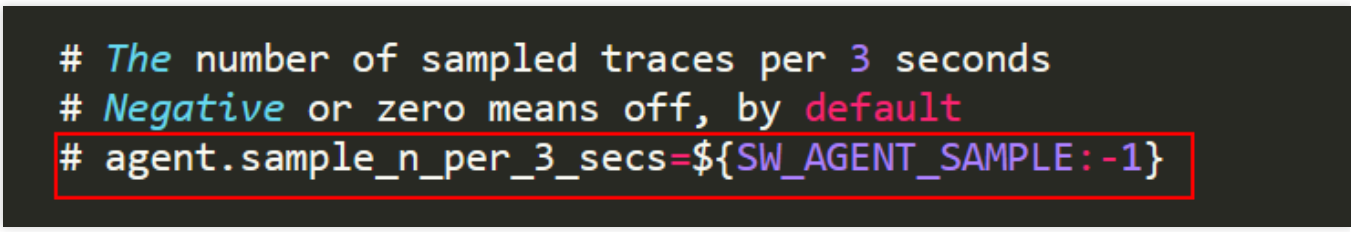
In sampling, certain data is sampled from all the collected trace data for analysis, which reduces the span volume and trace storage fees.

Prerequisites

You have reported the data of the Java application over the SkyWalking protocol.

Directions

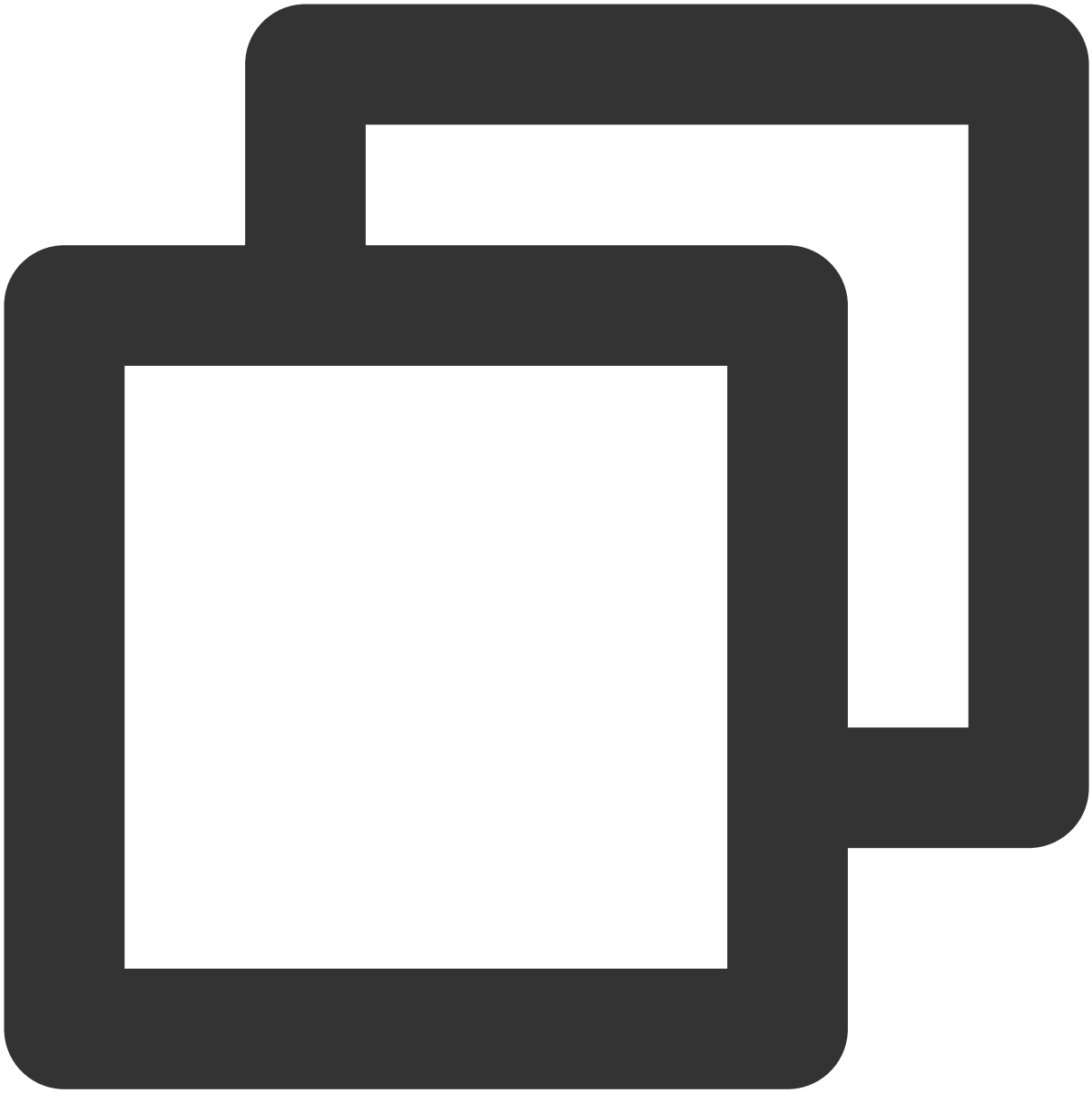
1. Open the `agent/config/agent.config` file and find the `agent.sample_n_per_3_secs=${SW_AGENT_SAMPLE:-1}` configuration item.



2. Modify the sample rate. **`agent.sample_n_per_3_secs` indicates the volume of trace data (TraceSegment) that can be collected every three seconds. If it is negative or `0`, all traces are collected, which is the default option.**

Example:

To collect 1,500 TraceSegments in three seconds, set as follows:



```
agent.sample_n_per_3_secs=${SW_AGENT_SAMPLE:1500}
```