Simplifying OpenTelemetry with Configuration

Observability Day
NORTH AMERICA

November 12, 2024
Salt Lake City

Jack Berg
Software Engineer
New Relic

Alex Boten
Staff Software Engineer
Honeycomb

Why is it complicated to configure OpenTelemetry?

(´･_･`)

Why is OpenTelemetry Configuration needed?

Observability Day
NORTH AMERICA

```go
endpoint := "api.honeycomb.io:443"
headers := map[string]string{
    "x-honeycomb-team": os.Getenv("HONEYCOMB_API_KEY"),
}

// Set up trace provider.
traceExporter, err := stdouttrace.New(
    stdouttrace.WithPrettyPrint())
if err != nil {
    handleErr(err)
    return
}

otlptracehttpExporter, err := otlptracehttp.New(ctx,
    otlptracehttp.WithEndpoint(endpoint),
    otlptracehttp.WithHeaders(headers),
    otlptracehttp.WithCompression(otlptracehttp.GzipCompression),
    otlptracehttp.WithTimeout(10000*time.Millisecond),
)
if err != nil {
    handleErr(err)
    return
}

tracerProvider := trace.NewTracerProvider(
    trace.WithBatcher(traceExporter,
        // Default is 5s. Set to 1s for demonstrative purposes.
        trace.WithBatchTimeout(time.Second)),
    trace.WithBatcher(otlptracehttpExporter),
)

shutdownFuncs = append(shutdownFuncs, tracerProvider.Shutdown)
otel.SetTracerProvider(tracerProvider)

// Set up meter provider.      You, 2 days ago • initial commit …
metricExporter, err := prometheus.New()
if err != nil {
    handleErr(err)
    return
}

otlpmetrichttpExporter, err := otlpmetrichttp.New(ctx,
    otlpmetrichttp.WithEndpoint(endpoint),
    otlpmetrichttp.WithHeaders(headers),
    otlpmetrichttp.WithCompression(otlpmetrichttp.GzipCompression),
    otlpmetrichttp.WithTimeout(10000*time.Millisecond))
if err != nil {
    handleErr(err)
    return
}

meterProvider := metric.NewMeterProvider(
    metric.WithReader(metricExporter),
    metric.WithReader(metric.NewPeriodicReader(otlpmetrichttpExporter)),
```

```python
import os
from opentelemetry.sdk.resources import SERVICE_NAME, Resource

from opentelemetry import trace
from opentelemetry.exporter.otlp.proto.http.trace_exporter import OTLPSpanExporter
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor, ConsoleSpanExporter

from opentelemetry import metrics
from opentelemetry.exporter.otlp.proto.http.metric_exporter import OTLPMetricExporter
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.sdk.metrics.export import PeriodicExportingMetricReader, ConsoleMetricExporter

resource = Resource(attributes={
    SERVICE_NAME: "your-service-name"
})

traceProvider = TracerProvider(resource=resource)
headers = {"x-honeycomb-team":os.environ["HONEYCOMB_API_KEY"]}
processor = BatchSpanProcessor(OTLPSpanExporter(endpoint="<traces-endpoint>/v1/traces", headers=headers))
traceProvider.add_span_processor(BatchSpanProcessor(ConsoleSpanExporter()))
traceProvider.add_span_processor(processor)
trace.set_tracer_provider(traceProvider)

readers = [
    PeriodicExportingMetricReader(OTLPMetricExporter(endpoint="<traces-endpoint>/v1/metrics")),
    PeriodicExportingMetricReader(ConsoleMetricExporter())
]
meterProvider = MeterProvider(resource=resource, metric_readers=[readers])
metrics.set_meter_provider(meterProvider)
```

Why is OpenTelemetry Configuration needed?

Observability Day
NORTH AMERICA

```javascript
const opentelemetry = require('@opentelemetry/sdk-node');
const {
  getNodeAutoInstrumentations,
} = require('@opentelemetry/auto-instrumentations-node');
const {
  OTLPTraceExporter,
} = require('@opentelemetry/exporter-trace-otlp-proto');
const {
  OTLPMetricExporter,
} = require('@opentelemetry/exporter-metrics-otlp-proto');
const { PeriodicExportingMetricReader } = require('@opentelemetry/sdk-metrics');

const sdk = new opentelemetry.NodeSDK({
  traceExporter: new OTLPTraceExporter({
    // optional - default url is http://localhost:4318/v1/traces
    url: '<your-otlp-endpoint>/v1/traces',
    // optional - collection of custom headers to be sent with each request, empty by default
    headers: {},
  }),
  metricReader: new PeriodicExportingMetricReader({
    exporter: new OTLPMetricExporter({
      url: '<your-otlp-endpoint>/v1/metrics',
      headers: {},
      concurrencyLimit: 1,
    }),
  }),
  instrumentations: [getNodeAutoInstrumentations()],
});
sdk.start();
```

```java
package otel;

import io.opentelemetry.exporter.logging.LoggingSpanExporter;
import io.opentelemetry.exporter.logging.otlp.OtlpJsonLoggingSpanExporter;
import io.opentelemetry.exporter.otlp.http.trace.OtlpHttpSpanExporter;
import io.opentelemetry.exporter.otlp.trace.OtlpGrpcSpanExporter;
import io.opentelemetry.sdk.trace.export.SpanExporter;
import java.time.Duration;

public class SpanExporterConfig {
  public static SpanExporter otlpHttpSpanExporter(String endpoint) {
    return OtlpHttpSpanExporter.builder()
        .setEndpoint(endpoint)
        .addHeader("api-key", "value")
        .setTimeout(Duration.ofSeconds(10))
        .build();
  }

  public static SpanExporter otlpGrpcSpanExporter(String endpoint) {
    return OtlpGrpcSpanExporter.builder()
        .setEndpoint(endpoint)
        .addHeader("api-key", "value")
        .setTimeout(Duration.ofSeconds(10))
        .build();
  }

  public static SpanExporter logginSpanExporter() {
    return LoggingSpanExporter.create();
  }

  public static SpanExporter otlpJsonLoggingSpanExporter() {
    return OtlpJsonLoggingSpanExporter.create();
  }
}
```

“authors MAY decide
what is the idiomatic approach”

৭(^‿^)৬

Environment variables to the rescue!

ಠ_ಠ

Environment Variables

Observability Day
NORTH AMERICA

- familiar
- already supported in many language implementations

Note: Support for environment variables is optional.

Feature	Go	Java	JS	Python	Ruby	Erlang	PHP
OTEL_SDK_DISABLED	-	+	-	+	-	-	+
OTEL_RESOURCE_ATTRIBUTES	+	+	+	+	+	+	+
OTEL_SERVICE_NAME	+	+	+	+	+	+	+
OTEL_LOG_LEVEL	-	-	+	-	+	-	+
OTEL_PROPAGATORS	-	+		+	+	+	+
OTEL_BSP_*	+	+	+	+	+	+	+
OTEL_BLRP_*		+					
OTEL_EXPORTER_OTLP_*	+	+		+	+	+	+
OTEL_EXPORTER_ZIPKIN_*	-	+		+	+	-	+
OTEL_TRACES_EXPORTER	-	+	+	+	+	+	+
OTEL_METRICS_EXPORTER	-	+		+	-	-	+
OTEL_LOGS_EXPORTER	-	+		+			+
OTEL_SPAN_ATTRIBUTE_COUNT_LIMIT	+	+	+	+	+	+	+
OTEL_SPAN_ATTRIBUTE_VALUE_LENGTH_LIMIT	+	+	+	+	+	+	+
OTEL_SPAN_EVENT_COUNT_LIMIT	+	+	+	+	+	+	+
OTEL_SPAN_LINK_COUNT_LIMIT	+	+	+	+	+	+	+
OTEL_EVENT_ATTRIBUTE_COUNT_LIMIT	+	-		+	+	+	+
OTEL_LINK_ATTRIBUTE_COUNT_LIMIT	+	-		+	+	+	+
OTEL_LOGRECORD_ATTRIBUTE_COUNT_LIMIT							+
OTEL_LOGRECORD_ATTRIBUTE_VALUE_LENGTH_LIMIT							+
OTEL_TRACES_SAMPLER	+	+	+	+	+	+	+
OTEL_TRACES_SAMPLER_ARG	+	+	+	+	+	+	+
OTEL_ATTRIBUTE_VALUE_LENGTH_LIMIT	+	+	+	+	+	-	+
OTEL_ATTRIBUTE_COUNT_LIMIT	+	+	+	+	+	-	+
OTEL_METRIC_EXPORT_INTERVAL	-	+		+			+
OTEL_METRIC_EXPORT_TIMEOUT	-	-		+			+
OTEL_METRICS_EXEMPLAR_FILTER	-	+					+
OTEL_EXPORTER_OTLP_METRICS_TEMPORALITY_PREFERENCE	+	+	+	+			+
OTEL_EXPORTER_OTLP_METRICS_DEFAULT_HISTOGRAM_AGGREGATION		+		+			
OTEL_EXPERIMENTAL_CONFIG_FILE							

Environment Variables limitations

- hard to express complex structured data
  - views (disable instruments, configure histogram buckets, etc)
  - custom processors, with order
  - multiple exporters, with independent batch processor config
  - prometheus exporter options
  - instrumentation
- lacks versioning
- limited values and validation
  - resource attributes with explicit types

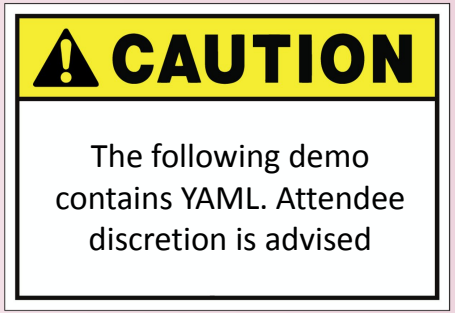
Declarative configuration

Declarative config **very** high level overview

- Alternative mechanism for configuring OTel
- Simplifies configuration
- Language agnostic
- Structured data model
- YAML file format
- Support for
  - environment variables expansion
  - complex configuration
  - instrumentation config
  - custom extension components

DEMO

٩(^‿^)۶


CAUTION
The following demo contains YAML. Attendee discretion is advised

Declarative Configuration Spec: Data Model, SDK, API

Reference Workflow

Input:

I want to export batches of spans over OTLP to some endpoint, with a “api-key” header set to a secret value.

Output:

```
OpenTelemetrySdk.builder()
  .setTracerProvider(SdkTracerProvider.builder()
    .addSpanProcessor(BatchSpanProcessor.builder(
      OtlpHttpSpanExporter.builder()
          .setEndpoint(System.getenv("OTLP_ENDPOINT"))
          .addHeader("api-key", System.getenv("API_KEY"))
          .build())
      .build())
    .build())
  .setMeterProvider(SdkMeterProvider.builder().build())
  .setLoggerProvider(SdkLoggerProvider.builder().build())
  .build();
```

Declarative Config Data Model

```yaml
file_format: "0.3"
disabled: ${OTEL_SDK_DISABLED:-false}
tracer_provider:
  processors:
    - batch:
        exporter:
          otlp:
            endpoint: ${OTLP_ENDPOINT}
            headers:
              - name: api-key
                value: ${API_KEY}
meter_provider: ...
logger_provider: ...
instrumentation: ...
```

- Types, properties, and semantics
- Defined using JSON Schema
- YAML file format
- Env var substitution

Declarative Config SDK

- Philosophy: common case should be easy, advanced case should be possible
- Primitives
  - `Parse`: accepts file, returns model
  - `Create`: accepts model, returns SDK
  - `ComponentProvider`: custom SDK extension points
- Typical use case:

```
OTEL_EXPERIMENTAL_CONFIG_FILE=/app/config.yaml
```

Output:

```
var sdk = create(parse(new File("/app/config.yaml")));

var tracerProvider = sdk.getTracerProvider();
var meterProvider = sdk.getMeterProvider();
var loggerProvider = sdk.getLoggerProvider();
```

Instrumentation Config API

```yaml
file_format: "0.3"
tracer_provider: ...
meter_provider: ...
logger_provider: ...
instrumentation:
  general:
    http:
      client:
        request_captured_headers:
          - Content-Type
          - Accept
  java:
    logback-appender:
      experimental-log-attributes: true
```

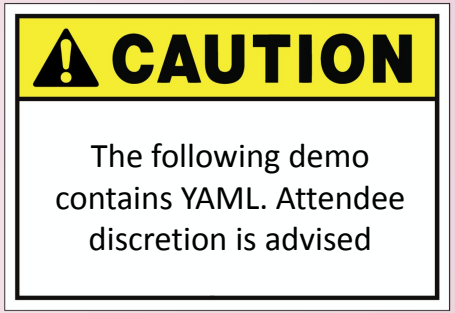
- All instrumentation can participate
- ConfigProvider

```java
configProvider.getInstrumentationConfig()
    .getStructured("java")
    .getStructured("logback-appender")
    .getBoolean("experimental-log-attributes");
```

- Standard config
- Domain-specific config

DEMO

٩(^‿^)۶


CAUTION
The following demo contains YAML. Attendee discretion is advised

Conclusion

- Declarative config is **language agnostic** and **highly expressive**
- Available today in a variety of languages
  - Some limitations, but users should give it a try
  - Contributors / maintainers should consider implementing
- Current / future config work is centered around declarative config
- Working towards stability, targeting 2025

Thanks

Config repo

Session feedback

Demo repo

Jack Berg
Software Engineer
New Relic

Alex Boten
Staff Software Engineer
Honeycomb

Appendix: resources

- open-telemetry/opentelemetry-configuration: JSON schema model definition
  - Schema
  - Examples: starter YAML templates
- Configuration OTEP #225: Initial enhancement proposal
- Declarative configuration specification: Specification for declarative configuration, including data model, API, and SDK
- Implementation tracker: Tracking status of language implementations

Appendix: configuration interfaces compared

	Configuration Interface		
Characteristic	Programmatic	Env Vars	Declarative Config
Zero-code	🔴	🟢	🟢
Language Agnostic	🔴	🟢	🟢
Expressiveness	🟢	🔴	🟢
Instrumentation config	🟢	🟡	🟢

Appendix: “not possible with env vars”

Use cases enabled by declarative config, and not possible with env var config.

- Export to multiple OTLP destinations
- Independently configure multiple batch processors
- Non-trivial sampler configs
- Configure views (disable instruments, configure explicit bucket bounds)
- Specify resource attribute types
- Configure advanced prometheus options (units, type suffix, scope info, resource labels)
- Configure custom extension components
- Configure instrumentation