CONFIGURATION GUIDES › NODE.JS, PHP, GO & COBOL

Manually instrument a Go application using OpenTelemetry and Applicare

Applicare 10.0 · Updated August 22, 2024 · go, opentelemetry · **Pending technical review**

Install the following packages:

```text
go get "go.opentelemetry.io/otel" \
  "go.opentelemetry.io/otel/exporters/stdout/stdoutmetric" \
  "go.opentelemetry.io/otel/exporters/stdout/stdouttrace" \
  "go.opentelemetry.io/otel/exporters/stdout/stdoutlog" \
  "go.opentelemetry.io/otel/sdk/log" \
  "go.opentelemetry.io/otel/log/global" \
  "go.opentelemetry.io/otel/propagation" \
  "go.opentelemetry.io/otel/sdk/metric" \
  "go.opentelemetry.io/otel/sdk/resource" \
  "go.opentelemetry.io/otel/sdk/trace" \
  "go.opentelemetry.io/otel/semconv/v1.24.0" \
  "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"\
  "go.opentelemetry.io/contrib/bridges/otelslog"
```

This installs the OpenTelemetry SDK components and `net/http` instrumentation.

Add the following packages to your Go application:

```
TEXT
import (
          "context"
          "fmt"
          "net"
          "os"
          "os/signal"
        "time"
          "net/http"
          "google.golang.org/grpc"
          "google.golang.org/grpc/credentials/insecure"
          "go.opentelemetry.io/otel"

          "go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc"
          "go.opentelemetry.io/otel/propagation"
          "go.opentelemetry.io/otel/sdk/resource"

         semconv "go.opentelemetry.io/otel/semconv/v1.25.0"
sdktrace "go.opentelemetry.io/otel/sdk/trace"
          "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"
)
```

You need to wrap the HTTP handlers with instrumented code.

YAML

```
var serviceName = semconv.ServiceNameKey.String(<APPLICARE_AGENT_NAME>)

func initConn() (*grpc.ClientConn, error) {
    conn, err := grpc.NewClient("localhost:4317",
        grpc.WithTransportCredentials(insecure.NewCredentials()),
    )
    if err != nil {
        return nil, fmt.Errorf("failed to create gRPC connection to collector: %w", err)
    }
    return conn, err
}

func initTracerProvider(ctx context.Context, res *resource.Resource, conn *grpc.ClientConn)
(func(context.Context) error, error) {
 traceExporter, err := otlptracegrpc.New(ctx, otlptracegrpc.WithGRPCConn(conn))
    if err != nil {
        return nil, fmt.Errorf("failed to create trace exporter: %w", err)
    }
    bsp := sdktrace.NewBatchSpanProcessor(traceExporter)
    tracerProvider := sdktrace.NewTracerProvider(
        sdktrace.WithSampler(sdktrace.AlwaysSample()),
        sdktrace.WithResource(res),
        sdktrace.WithSpanProcessor(bsp),
    )
    otel.SetTracerProvider(tracerProvider)
    otel.SetTextMapPropagator(propagation.TraceContext{})
    return tracerProvider.Shutdown, nil
}

func main() {
    if err := run(); err != nil {
        log.Fatalln(err)
    }
}

func run() (err error) {
    ctx, stop := signal.NotifyContext(context.Background(), os.Interrupt)
    defer stop()

    conn, err := initConn()

    if err != nil {
        log.Fatal(err)
    }
```

```
    res, err := resource.New(ctx,
        resource.WithAttributes(
            serviceName,
        ),
    )

    if err != nil {
        log.Fatal(err)
    }

    shutdownTracerProvider, err := initTracerProvider(ctx, res, conn)

    if err != nil {
        log.Fatal(err)
    }

    defer func() {
        if err := shutdownTracerProvider(ctx); err != nil {
            log.Fatalf("failed to shutdown TracerProvider: %s", err)
        }
    }()

// Start HTTP server.
    srv := &http.Server{
        Addr:         ":8080",
        BaseContext:  func(_ net.Listener) context.Context { return ctx },
        ReadTimeout:  time.Second,
        WriteTimeout: 10 * time.Second,
        Handler:      newHTTPHandler(),
    }

    srvErr := make(chan error, 1)
        go func() {
        srvErr <- srv.ListenAndServe()
    }()

    select {
        case err = <-srvErr:
        return
        case <-ctx.Done():
        stop()
    }

    err = srv.Shutdown(context.Background())
```

```
    return
}

   func newHTTPHandler() http.Handler {
       mux := http.NewServeMux()
       handleFunc := func(pattern string, handlerFunc func(http.ResponseWriter,
*http.Request)) {
        handler := otelhttp.WithRouteTag(pattern, http.HandlerFunc(handlerFunc))
        mux.Handle(pattern, handler)
    }

   // your handler function here ..........
   // ...........

    handler := otelhttp.NewHandler(mux, "/")
    return handler
}
```