Additional Resources

Observability Architecture

In this course, we implement observability according to the following architecture:

- **Instrumentation:** NPM libraries and Open Telemetry (OTel) libraries
- **Collecting, processing, and exporting:** Open Telemetry Collector
- **Visualization:** Grafana Cloud

Configuring Your Observability Stack (in video 01_04)

Configuring the Open Telemetry Collector

- OTel documentation
  - https://opentelemetry.io/docs/collector/
- Running OTel Collector in Docker
  - https://opentelemetry.io/docs/collector/getting-started/

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318
        cors:
          allowed_origins:
            - http://*
            # Origins can have wildcards with *, use * by itself to
match any origin.
            - https://*
  filelog:
    include: [ /conf/combined.log ]
    operators:
      - type: json_parser
        timestamp:
          parse_from: attributes.created
          layout: '%Y-%m-%d %H:%M:%S'

  prometheus:
    config:
      scrape_configs:
```

```
      - job_name: 'bestbags'
        scrape_interval: 2s
        static_configs:
          - targets: ['host.docker.internal:8000']
            labels:
              service: 'best-bags'
              group: 'application'

exporters:
  otlphttp:
    auth:
      authenticator: basicauth/otlp
    endpoint: https://otlp-gateway-prod-us-east-0.grafana.net/otlp *replace
depending on where grafana instance is deployed (in this case us-east-0)

extensions:
  basicauth/otlp:
    client_auth:
      username: *replace  Grafana username*
      password: *replace Grafana password*

processors:
  batch:

service:
  extensions: [basicauth/otlp]
  telemetry:
    logs:
      level: "debug"
  pipelines:
    metrics:
      receivers: [prometheus]
      exporters: [otlphttp]
      processors: [batch]
    traces:
      receivers: [otlp]
      exporters: [otlphttp]
      processors: [batch]
    logs:
      receivers: [filelog]
      exporters: [otlphttp]
      processors: [batch]
```

collector-config.yaml

```
version: "3.8"
services:
      collector:
    image: otel/opentelemetry-collector-contrib:0.83.0
```

```
    command: ["--config=/conf/collector-config.yaml"]
    volumes:
      - ./collector-config.yaml:/conf/collector-config.yaml
    ports:
      - "4317:4317"
      - "4318:4318"
      - "4320:4320"
```

docker-compose.yaml

Instrumenting your application (Node.js)

- Metrics (in video 02_03)
  - Prom-client npm library

```
# Initialise prom-client library
const promClient = require('prom-client');
const register = promClient.register;

# Collect Default Node JS prometheus metrics
const collectDefaultMetrics = promClient.collectDefaultMetrics;
collectDefaultMetrics({
   register,
  labels: {'service': 'best-bags'}
});

#Create custom metrics
const product_views = new promClient.Counter({
  name: 'product_views',
  help: 'Counts the number of times a product is viewed',
  labelNames: ['product_id', 'product_name', 'product_price']
});

product_views.labels({ product_id: req.params.id, product_name: product.title,
product_price: product.price  }).inc();
```

  - OTLP Metrics Exporter
    - HTTP
    - Proto
- Traces (in video 03_02)
  - OTLPTraceExporter library
    - HTTP

```
const opentelemetry = require("@opentelemetry/api");
const { Resource } = require("@opentelemetry/resources");
```

```
const { SemanticResourceAttributes } = require("@opentelemetry/semantic-
conventions");
const { WebTracerProvider } =
require("@opentelemetry/sdk-trace-web");
const { registerInstrumentations } = require("@opentelemetry/instrumentation");
const { getNodeAutoInstrumentations } = require("@opentelemetry/auto-
instrumentations-node");
const { HttpInstrumentation } = require("@opentelemetry/instrumentation-http");
const { ExpressInstrumentation } = require("opentelemetry-instrumentation-
express");
const { ConsoleSpanExporter } = require('@opentelemetry/sdk-trace-node');
const { BatchSpanProcessor } = require("@opentelemetry/sdk-trace-base");
const {
    OTLPTraceExporter,
} = require("@opentelemetry/exporter-trace-otlp-http");

const {
  HTTPTraceExporter,
} = require("@opentelemetry/exporter-trace-otlp-http");

// Optionally register automatic instrumentation libraries
registerInstrumentations({
    instrumentations: [
        new getNodeAutoInstrumentations(),
        new HttpInstrumentation(),
        new ExpressInstrumentation({
          requestHook: (span, reqInfo) => {
            span.setAttribute('request-
headers',JSON.stringify(reqInfo.req.headers))
          }
        })
    ],
});

const resource = Resource.default().merge(
    new Resource({
    [SemanticResourceAttributes.SERVICE_NAME]: 'best-bags',
    [SemanticResourceAttributes.SERVICE_VERSION]: "0.1.0",
    })
);

const provider = new WebTracerProvider({
  resource: resource,
});

//const traceExporter = new ConsoleSpanExporter();

// const console_processor = new BatchSpanProcessor(traceExporter);
// provider.addSpanProcessor(console_processor);
```

```
const exporter = new OTLPTraceExporter({
    url: “http://collector:4318/v1/traces”,
    //headers: {},
});
const processor = new BatchSpanProcessor(exporter);
provider.addSpanProcessor(processor);

provider.register();
```

  - Create custom spans and attributes

```
app.get(‘/metrics’, async (req, res) => {

let tracer = otel.trace.getTracer(‘best-bags’);
let current_span = otel.trace.getSpan(otel.context.active());

let trace_id = current_span.spanContext().traceId;
let payload = {
traceId : trace_id,
}

const parentSpan = tracer.startSpan(‘metrics’, { attributes: { foo: ‘bar’ }
});
parentSpan.end();

logger.response(res, 200, payload)

  res.set(‘Content-Type’, ‘application/json’);
  res.send(await register.metrics());
});
```

- Logs (in video 04_04)
  - Winston npm ibrary

```
#Initialise Winston Library
const winston = require(‘winston’);

#Create Winston file logger to log to file
const fileLogger = new winston.transports.File({
    filename: ‘./combined.log’,
    format: winston.format.json(), });
```