OpenTelemetry Developer Handbook – Azure, GCP & AWS

CHAPTER 1

What is OpenTelemetry?

OpenTelemetry (OTel) is an open-source observability framework and toolkit that gives you a single, vendor-neutral way to generate, collect and export *telemetry data* — the signals your application produces to tell you what it is doing and how healthy it is.

It is a **CNCF Graduated project** (the highest maturity level), widely adopted by Google, Microsoft, AWS, Datadog, and hundreds of others.

?

WHY SHOULD YOU CARE AS A JUNIOR DEVELOPER?

When something breaks in production, you need to know *where* it broke, *why*, and *how long* it has been broken. OpenTelemetry gives you that information automatically, with one consistent standard.

The Three Pillars of Observability

?

Traces

A trace follows a single request across multiple services. Each step is a **span**.

?

Metrics

Numeric measurements over time: request count, error rate, memory usage, custom KPIs.

?

Logs

Timestamped event records. OTel correlates logs with the exact trace and span they belong to.

?	**OTEL COLLECTOR IS YOUR BEST FRIEND** The Collector receives data from your app, transforms or filters it, and forwards it to one or many backends. Switch cloud vendors without changing application code.

CHAPTER 2

Getting Started – Your First Instrumented App

We will use **Node.js** as the example. The same concepts apply to Python, Java, Go, .NET, etc.

Step 1 – Install the Core SDK Packages

```
npm install @opentelemetry/api @opentelemetry/sdk-node @opentelemetry/auto-instrumentations-node
```

Step 2 – Create instrumentation.js

```
// instrumentation.js — load BEFORE your app
const {{ NodeSDK }} = require('@opentelemetry/sdk-node');
const {{ getNodeAutoInstrumentations }} = require('@opentelemetry/auto-instrumentations-node');
const {{ Resource }} = require('@opentelemetry/resources');
const {{ SemanticResourceAttributes }} = require('@opentelemetry/semantic-conventions');
const {{ ConsoleSpanExporter }} = require('@opentelemetry/sdk-trace-node');

const sdk = new NodeSDK({{
  resource: new Resource({{
    [SemanticResourceAttributes.SERVICE_NAME]: 'my-first-service',
    [SemanticResourceAttributes.SERVICE_VERSION]: '1.0.0',
  }}),
  traceExporter: new ConsoleSpanExporter(),
  instrumentations: [getNodeAutoInstrumentations()],
}});
sdk.start();
process.on('SIGTERM', () => sdk.shutdown().finally(() => process.exit(0)));
```

Step 3 – Run your app

```
node -r ./instrumentation.js app.js
```

?	**TRACES WORKING!** ConsoleSpanExporter is only for development. The next chapters replace it with a real cloud exporter.

CHAPTER

3

Connecting to Microsoft Azure

Az

Azure Monitor + Application Insights

The Azure-native observability backend for OTel

1

Create an Application Insights Resource

Azure Portal ? "Application Insights" ? Create. Copy the **Connection String** from the resource overview.

2

Install the Azure Monitor exporter

```
npm install @azure/monitor-opentelemetry-expor
```

3

Update instrumentation.js

```
const {{ AzureMonitorTraceExporter }} = requir
const {{ AzureMonitorMetricExporter }} = requi
const {{ PeriodicExportingMetricReader }} = re
const connectionString = process.env.APPLICATI
const sdk = new NodeSDK({{
  traceExporter: new AzureMonitorTraceExporter
  metricReader: new PeriodicExportingMetricRea
    exporter: new AzureMonitorMetricExporter({
  }}),
  instrumentations: [getNodeAutoInstrumentatic
}});
sdk.start();
```

4

Set environment variable and run

```
# Linux / macOS
export APPLICATIONINSIGHTS_CONNECTION_STRING="
node -r ./instrumentation.js app.js

# Windows PowerShell
$env:APPLICATIONINSIGHTS_CONNECTION_STRING = "
node -r ./instrumentation.js app.js
```

5

Verify in Azure Portal

Azure Portal ? Application Insights ? Transaction Search, Application Map, Performance, Logs (KQL).

```
requests
| where timestamp > ago(1h) and duration > 500
| project timestamp, name, duration, resultCod
| order by duration desc | take 50
```

CHAPTER

4

Connecting to Google Cloud (GCP)

G

Cloud Trace + Cloud Monitoring

Google Cloud's distributed tracing and metrics platform

1

Enable APIs and create a Service Account

```
gcloud services enable cloudtrace.googleapis.c
gcloud iam service-accounts create otel-export
gcloud projects add-iam-policy-binding YOUR_PR
gcloud iam service-accounts keys create ./gcp-
```

2

Install and configure GCP exporters

```
npm install @google-cloud/opentelemetry-cloud-
```

```
const {{ TraceExporter }} = require('@google-c
const {{ MetricExporter }} = require('@google-
const projectId = process.env.GOOGLE_CLOUD_PRO
const sdk = new NodeSDK({{
  traceExporter: new TraceExporter({{ projectI
  metricReader: new PeriodicExportingMetricRea
    exporter: new MetricExporter({{ projectId
  }}),
  instrumentations: [getNodeAutoInstrumentatio
}});
sdk.start();
```

3

Set credentials and run

```
export GOOGLE_APPLICATION_CREDENTIALS="./gcp-c
export GOOGLE_CLOUD_PROJECT="your-gcp-project-
node -r ./instrumentation.js app.js
```

Google Cloud Console ? Cloud Trace ? Trace Explorer ? click any trace for the full span view.

CHAPTER

5

Connecting to Amazon Web Services (AWS)

AWS

AWS X-Ray + CloudWatch + ADOT

AWS Distro for OpenTelemetry — AWS's official OTel distribution

?

AWS X-RAY USES A SPECIAL TRACE FORMAT

X-Ray requires timestamp-based trace IDs. You must include AWSXRayIdGenerator or traces will not appear correctly.

1

Install ADOT packages

```
npm install @opentelemetry/id-generator-aws-xr
```

2

Update instrumentation.js

```
const {{ AWSXRayIdGenerator }} = require('@ope
const {{ AWSXRayPropagator }} = require('@oper
const {{ OTLPTraceExporter }} = require('@oper
const {{ propagation }} = require('@opentelem
propagation.setGlobalPropagator(new AWSXRayPr
const sdk = new NodeSDK({{
  idGenerator: new AWSXRayIdGenerator(),  // C
  traceExporter: new OTLPTraceExporter({{ url:
  instrumentations: [getNodeAutoInstrumentatio
}});
sdk.start();
```

3

Run and verify in AWS Console

```
export AWS_REGION=us-east-1
node -r ./instrumentation.js app.js
```

AWS Console ? CloudWatch ? X-Ray ? Traces. Check X-Ray ? Service Map for auto-generated topology.

CHAPTER

6

Best Practices & Quick Reference

Quick Comparison: Azure vs GCP vs AWS

FEATURE	AZURE MONITOR	GCP CLOUD TRACE	AWS X-RAY
Trace backend	Application Insights	Cloud Trace	AWS X-Ray
Metric backend	Azure Monitor Metrics	Cloud Monitoring	Amazon CloudWatch
Auth method	Connection String	Service Account JSON	IAM Role / Keys
Query language	KQL (Kusto)	Filter expressions	CloudWatch Insights
Free tier	5 GB/month	2.5M spans/month	100K traces/month
Special note	None	Enable APIs in console	X-Ray ID generator required

Common Errors and Fixes

ERROR	CAUSE	FIX
`No spans exported`	SDK not started before app	`node -r ./instrumentation.js app.js`
`401 Unauthorized (Azure)`	Wrong connection string	Check APPLICATIONINSIGHTS_CONNECTION_STRING
`PERMISSION_DENIED (GCP)`	Missing SA roles	`Add roles/cloudtrace.agent`
Traces missing in X-Ray	Missing ID generator	Add idGenerator: new AWSXRayIdGenerator()
`ECONNREFUSED :4317`	Collector not running	Start the Collector container first

?	**NEXT STEPS** 1. Add custom spans to your key business functions. 2. Add custom metrics (orders.processed, queue.depth). 3. Set up alerts on error rate and p99 latency. 4. Explore OTel Collector processors: filter, attributes, tail_sampling. 5. Read the official docs at opentelemetry.io.

Revision #1
Created 2026-07-03 06:19:01 UTC by Mohammad M Javid
Updated 2026-07-03 07:02:51 UTC by Mohammad M Javid