Groundcover & OpenTelemetry Integration Guide

Overview

This guide helps you integrate your services with **Groundcover** (observability platform) using **OpenTelemetry**.

What you'll do: Configure your application to send metrics and traces to Groundcover.

Two approaches:

1. **Direct to Groundcover** - Send data directly from your app
2. **Via OTEL Collector** (Recommended) - Send to OTEL Collector, which forwards to Groundcover

Note: This guide uses Node.js examples, but the same concepts apply to Python, Java, Go, and other languages.

Prerequisites: Check with DevOps

Before starting integration, verify with your DevOps team:

1. Groundcover Agent Installation

- **Check**: Groundcover agent should be running in your cluster

2. OpenTelemetry (OTEL) Installation

- **Check**: OTEL Operator and OTEL Stack should be installed

---

Integration Approaches

Important Note: Groundcover accepts data in **OTLP (OpenTelemetry Protocol)** format. This means you **must use OpenTelemetry SDKs** in your application code to format and send the data, regardless of which option you choose.

Option 1: Direct to Groundcover

App → OpenTelemetry SDK → Groundcover (direct)

- Your app uses OTEL SDKs to format data
- Sends directly to Groundcover endpoint
- Requires Groundcover API key in your app
- Simpler architecture, but less flexible

Option 2: Via OpenTelemetry Collector (Recommended)

App → OpenTelemetry SDK → OTEL Collector → Groundcover

- Your app uses OTEL SDKs to format data
- Sends to OTEL Collector (in-cluster service)
- Collector forwards to Groundcover
- No API key needed in your app (handled by collector)
- Better data processing, batching, and error handling
- Centralized configuration

Option 1: Direct Integration with Groundcover

This option is for teams that want their application to send data **directly to Groundcover**, without using the OTEL Collector.

When to use this option

- You have a simple setup or a small number of services
- You are comfortable putting the Groundcover API key in the application (via Kubernetes Secret)
- You don't need centralized control over exporters or pipelines

High-level steps for developers

- **Step 1 – Confirm prerequisites with DevOps**
  - Groundcover agent is installed and healthy in the cluster
  - You have the **Groundcover OTLP endpoint** and **API key**

- **Step 2 – Add OpenTelemetry SDK to your service**
  - For Node.js: install OTEL SDK packages (metrics + traces)
  - For other languages: follow the official OpenTelemetry docs for your language
  - Goal: your service can produce OTLP metrics/traces
- **Step 3 – Configure exporter to send directly to Groundcover**
  - Set the OTLP endpoint to Groundcover's inCloud endpoint (e.g. `bta4ka.platform.grcv.io:443`)
  - Configure the API key in headers or environment variables
  - In code, configure the OTEL exporter to use this endpoint
- **Step 4 – Wire it into your startup**
  - Ensure OTEL initialization runs **before** your app starts handling traffic
  - Typically by requiring an `instrumentation` module in your entrypoint
- **Step 5 – (Optional) Add custom metrics/traces**
  - Use OTEL APIs to create counters, histograms, and custom spans
  - These will appear in Groundcover once data starts flowing

For exact code examples per language, follow:

- Ingestion Endpoints
- Groundcover OpenTelemetry direct integration docs
- OpenTelemetry language-specific docs

Option 2: Integration via OpenTelemetry Collector (Recommended)

This is the **recommended pattern** and matches how your clusters are already configured.

Your application sends data to the **OTEL Collector service inside the cluster**, and the Collector forwards everything to Groundcover.

When to use this option

- You want a **standard pattern** for all services
- You don't want to expose the Groundcover API key to each app
- You want to be able to change exporters, sampling, or processing **centrally** in OTEL Collector

High-level steps for developers

- **Step 1 – Confirm prerequisites with DevOps**
  - OTEL Operator and OTEL Stack are installed (namespace `observability`)
  - Collector service is available, e.g.:
    - `otel-stack-deployment-collector.observability.svc.cluster.local:4317` (gRPC)
    - `otel-stack-deployment-collector.observability.svc.cluster.local:4318` (HTTP)
  - DevOps confirms the Collector is already configured to export to Groundcover
- **Step 2 – Add OpenTelemetry SDK to your service**
  - For Node.js: install OTEL SDK packages (metrics + traces)
  - For other languages: follow the official OTEL docs
  - The service will emit OTLP to an endpoint in the same cluster
- **Step 3 – Configure exporter to send to OTEL Collector**
  - Set `OTEL_EXPORTER_OTLP_ENDPOINT` to the Collector service
    - Example:
      - `OTEL_EXPORTER_OTLP_ENDPOINT=<http://otel-stack-deployment-collector.observability.svc.cluster.local:4317>`
  - No Groundcover API key is needed in the app (Collector knows how to talk to Groundcover)
- **Step 4 – Wire it into your startup**
  - Ensure OTEL initialization runs before your app starts
  - Same pattern as Option 1, but the exporter target is the Collector, not Groundcover

- **Step 5 – (Optional) Add custom metrics/traces**
  - Use OTEL APIs to define business metrics and spans
  - They will show up in Groundcover via the Collector

The key idea: **developers only need to know the OTEL Collector service URL** and how to enable OTEL in their language. All Groundcover-specific wiring is handled in the Collector configuration by DevOps.

Simple Node.js Example

Here's a minimal example to get you started:

1. Install packages:

```
npm install @opentelemetry/api @opentelemetry/sdk-metrics @opentelemetry/exporter-metrics-otlp-grpc @opentelemetry/sdk-trace-node
@opentelemetry/exporter-trace-otlp-grpc @opentelemetry/sdk-node
```

2. Create `src/instrumentation/otel.ts`:

```
import { metrics } from '@opentelemetry/api';
import { OTLPMetricExporter } from '@opentelemetry/exporter-metrics-otlp-grpc';
import { OTLPTraceExporter } from '@opentelemetry/exporter-trace-otlp-grpc';
import { MeterProvider, PeriodicExportingMetricReader } from '@opentelemetry/sdk-metrics';
import { NodeSDK } from '@opentelemetry/sdk-node';

const OTEL_COLLECTOR_URL = process.env.OTEL_EXPORTER_OTLP_ENDPOINT ||
  'http://otel-stack-deployment-collector.observability.svc.cluster.local:4317';

// Setup metrics exporter
const metricExporter = new OTLPMetricExporter({ url: OTEL_COLLECTOR_URL });
const metricReader = new PeriodicExportingMetricReader({
  exporter: metricExporter,
  exportIntervalMillis: 5000
});
const meterProvider = new MeterProvider({ readers: [metricReader] });
metrics.setGlobalMeterProvider(meterProvider);

// Setup trace exporter
const traceExporter = new OTLPTraceExporter({ url: OTEL_COLLECTOR_URL });

// Initialize SDK
const sdk = new NodeSDK({
  traceExporter: traceExporter,
  autoDetectResources: true,
});

sdk.start();
console.log('OpenTelemetry initialized. Sending to OTEL Collector.');
```

3. Load at startup in `package.json`:

```
{
  "scripts": {
    "start": "node --require ./dist/instrumentation/otel.js dist/app.js"
  }
}
```

4. Set environment variable in your deployment:

```
env:
  - name: OTEL_EXPORTER_OTLP_ENDPOINT
    value: 'http://otel-stack-deployment-collector.observability.svc.cluster.local:4317'
```

That's it! Your metrics and traces will flow: **App → OTEL Collector → Groundcover**

Auto-Instrumentation (Optional)

For automatic instrumentation of common libraries (Express, HTTP, Redis, etc.):

Install Auto-Instrumentation Package

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

Update Instrumentation File

```
import { getNodeAutoInstrumentations } from '@opentelemetry/auto-instrumentations-node';

const sdk = new NodeSDK({
  resource: resource,
  traceExporter: traceExporter,
  autoDetectResources: true,
```

```
  instrumentations: [
    getNodeAutoInstrumentations({
      // Disable specific instrumentations if needed
      // '@opentelemetry/instrumentation-fs': { enabled: false },
    }),
  ],
});
```

This automatically instruments:

- HTTP/HTTPS requests
- Express.js
- Redis (ioredis, redis)
- MongoDB
- PostgreSQL
- And many more...

Example: Express.js Integration

```
import express from 'express';
import { metrics } from '@opentelemetry/api';

const app = express();
const meter = metrics.getMeter('your-service-name', '1.0.0');

const requestCounter = meter.createCounter('http.requests.total');
const requestDuration = meter.createHistogram('http.request.duration');

app.use((req, res, next) => {
  const startTime = Date.now();

  res.on('finish', () => {
    const duration = Date.now() - startTime;
    requestCounter.add(1, {
      method: req.method,
      route: req.route?.path || req.path,
      status: res.statusCode.toString(),
    });
    requestDuration.record(duration, {
      method: req.method,
      route: req.route?.path || req.path,
    });
  });

  next();
});

app.get('/api/users', (req, res) => {
  res.json({ users: [] });
});

app.listen(3000);
```

Verification Steps

1. Check Application Logs

Look for OpenTelemetry initialization messages:

```
OpenTelemetry initialized. Sending to collector at http://otel-stack-deployment-collector.observability.svc.cluster.local:4317
```

2. Verify Metrics are Being Sent

Check OTEL Collector logs:

```
kubectl logs -n observability -l app.kubernetes.io/name=opentelemetry-kube-stack -c otel-collector --tail=50
```

3. Check Groundcover Dashboard

- Log in to Groundcover dashboard
- Navigate to your service
- Verify metrics and traces are appearing

Quick Reference

Environment Variables

Variable	Option 1 (Direct)	Option 2 (OTEL)
Endpoint	`GROUNDCOVER_ENDPOINT` (required)	`OTEL_EXPORTER_OTLP_ENDPOINT` (required)
API Key	`GROUNDCOVER_API_KEY` (required)	Not needed (handled by collector)

Endpoints

- **OTEL Collector (gRPC)**: `<http://otel-stack-deployment-collector.observability.svc.cluster.local:4317>`
- **OTEL Collector (HTTP)**: `<http://otel-stack-deployment-collector.observability.svc.cluster.local:4318>`
- **Groundcover**: `bta4ka.platform.grcv.io:443` (configured in collector)

Support

For issues or questions:

1. Check this guide first
2. Review OpenTelemetry documentation: Documentation
3. Contact DevOps team for infrastructure issues
