CertSecure Backend – OpenTelemetry (OTLP) Integration Guide

This guide describes how to export CertSecure Backend application logs over OpenTelemetry to an OTLP-compatible observability or SIEM platform. CertSecure Backend has built-in OpenTelemetry support and sends its logs over OTLP to an OpenTelemetry Collector; the Collector forwards them to the platform of your choice.

Note: Local file logging (logs/certsecure_backend.log) runs independently of OpenTelemetry and is unaffected if the Collector or destination platform is unavailable.

Step 1: Stand Up an OTLP Collector Endpoint

- Deploy an OpenTelemetry Collector reachable from the CertSecure backend. Running it on the CertSecure backend host itself is recommended, because the backend-to-Collector hop is unencrypted and unauthenticated.
- Enable the OTLP gRPC receiver on port 4317. The backend exports over gRPC only; a Collector configured with the OTLP/HTTP receiver (port 4318) alone will not receive any data.
- Choose a distribution:
  - **Core (otelcol)** – sufficient when the destination accepts OTLP and you forward with the otlp or otlphttp exporter.
  - **Contrib (otelcol-contrib)** – required for vendor-specific exporters such as datadog, splunk_hec, elasticsearch, loki, azuremonitor or awscloudwatchlogs.
- Install the Collector:

```
# Set to the current Collector release listed at
# https://github.com/open-telemetry/opentelemetry-collector-releases/releases
OTELCOL_VERSION=<collector-version>
curl --proto '=https' --tlsv1.2 -fOL \
  https://github.com/open-telemetry/opentelemetry-collector-releases/releases/
download/v${OTELCOL_VERSION}/otelcol_${OTELCOL_VERSION}_linux_amd64.tar.gz
tar -xvf otelcol_${OTELCOL_VERSION}_linux_amd64.tar.gz
sudo install -m 0755 otelcol /usr/local/bin/otelcol
sudo mkdir -p /etc/otelcol
```

Note: The CertSecure Linux installer can provision the core Collector for you, including a default configuration file and service definition. If you use it, replace the exporter section of that default configuration with the destination for your environment.

Note: This integration uses the logs pipeline, which is what the configuration below sets up.

Step 2: Configure the Collector Pipeline

- Create the Collector configuration (for example /etc/otelcol/config.yaml). The receiver and processor sections are the same regardless of destination:

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 127.0.0.1:4317   # use 0.0.0.0:4317 only for a remote backend
processors:
  memory_limiter:
    check_interval: 1s
```

```
    limit_percentage: 80
    spike_limit_percentage: 20
  batch:
    send_batch_size: 512
    timeout: 5s
exporters:
  otlphttp:   # see the destination options below
    endpoint: https://otlp.example.com
    headers:
      authorization: "Bearer ${env:OTLP_TOKEN}"
    tls:
      insecure: false                  # keep TLS on, and never set
      insecure_skip_verify: false       # insecure_skip_verify to true in
production
  # debug:                   # enable temporarily during validation
  #   verbosity: detailed
service:
  pipelines:
    logs:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [otlphttp]
  telemetry:
    logs:
      level: info
```

- Create a dedicated unprivileged account for the Collector so that it does not run as root:

```
sudo useradd --system --no-create-home --shell /usr/sbin/nologin otelcol
sudo chown root:root /etc/otelcol/config.yaml
sudo chmod 644 /etc/otelcol/config.yaml
```

- Keep credentials out of the configuration file. Place them in an environment file that only root and the Collector account can read, and reference them with ${env:VAR}:

```
echo 'OTLP_TOKEN=<your-token>' | sudo tee /etc/otelcol/otelcol.env
sudo chown root:otelcol /etc/otelcol/otelcol.env
sudo chmod 640 /etc/otelcol/otelcol.env
```

- Create the systemd unit /etc/systemd/system/otelcol.service:

```
[Unit]
Description=OpenTelemetry Collector
After=network.target

[Service]
User=otelcol
Group=otelcol
EnvironmentFile=/etc/otelcol/otelcol.env
ExecStart=/usr/local/bin/otelcol --config /etc/otelcol/config.yaml
Restart=always
RestartSec=5
# Hardening -- the Collector needs no privileges beyond reading its own config
NoNewPrivileges=true
```

```
ProtectSystem=strict
ProtectHome=true
PrivateTmp=true
PrivateDevices=true

[Install]
WantedBy=multi-user.target
```

- Enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable --now otelcol
systemctl status otelcol
```

Note: Only the exporter section changes between destinations – the receiver, processors and CertSecure configuration are identical in every case. Use otlp or otlphttp for OTLP-native platforms; vendor exporters such as datadog, splunk_hec, elasticsearch or azuremonitor require the contrib distribution. For Datadog specifically, see the CertSecure Backend Datadog Integration Guide.

Step 3: Ensure Network Connectivity

- Backend to Collector (OTLP/gRPC, TCP 4317). If the Collector runs on the CertSecure backend host, bind it to 127.0.0.1:4317 – no firewall change is needed and the traffic never leaves the host. This is the recommended deployment.
- Remote Collector. If the Collector runs on a separate host, bind it to 0.0.0.0:4317, open the port, and restrict it to the CertSecure backend's IP address:

```
sudo firewall-cmd --permanent --add-port=4317/tcp
sudo firewall-cmd --reload
```

- Collector to destination. Allow outbound access from the Collector host to the destination platform – typically HTTPS on TCP 443. Behind a proxy, add HTTPS_PROXY to the Collector's EnvironmentFile.

Note: The CertSecure backend's OTLP exporter is plaintext gRPC with no TLS and no authentication, and this is not configurable. Never expose port 4317 to an untrusted network – keep it on loopback or behind a firewall rule scoped to the backend host. The backend cannot send OTLP directly to a TLS-protected or token-authenticated endpoint; the Collector is the security boundary, terminating the plaintext hop and applying TLS and credentials on the outbound leg.

Step 4: Configure the CertSecure Backend (settings.yaml)

- Edit settings.yaml in the CertSecure Backend installation directory. It is read when the service starts.
- Apply the following configuration:

```
service_name: "certsecure_backend"
enable_signals:
  logging: true       # turns on OTLP log export
  tracing: false
  metrics: false
otlp_grpc_endpoint: "127.0.0.1:4317"   # host:port, no scheme
root_level: "INFO"
resource_attributes:
  deployment.environment: "production"
  service.version: "<product-version>"
  service.instance.id: "certsecure-app-01"
```

```
log_handlers:
  - logger_name: "certsecure_backend"
    handler_type: "RotatingFileHandler"
    file_path: "logs/certsecure_backend.log"
    level: "INFO"
    maxBytes: 100000000
    backupCount: 10
    format_string: "%(asctime)s %(levelname)s %(module)s %(thread)d - %
(funcName)s:%(lineno)s:: %(message)s"
    datefmt: '%Y-%m-%d %H:%M:%S'
```

- Field notes:
- **service_name** – identifies the service in the destination backend. Sent as the service.name resource attribute.
- **enable_signals** – set logging to true to export logs to the Collector. Tracing and metrics are separate signals and are not part of this integration; leave them false.
- **otlp_grpc_endpoint** – host:port with no scheme. Required whenever any signal is enabled; if it is missing, the service will not start.
- **root_level** – the minimum severity exported. Applies to the local log files and to the records sent to the Collector alike. Keep it at INFO or higher in production: DEBUG substantially increases both the volume and the detail leaving the host.
- **resource_attributes** – merged into the OpenTelemetry Resource on every record, and therefore sent with all of them. Use descriptive labels such as environment or instance identifiers only; never place credentials, tokens or personal data here.
- **log_handlers** – at least one entry is required or the service will not start. Local file logging is independent of the export to the Collector.

Note: The level setting must be named root_level. If your settings.yaml contains a key named telemetry_level, rename it – that name is not recognised, so the level silently stays at INFO and a warning is recorded in logging_manager_errors.log.

Attributes added automatically to every record (unless overridden in resource_attributes): service.name, service.version, host.name, process.owner, process.pid.

Note: OpenTelemetry configuration for the backend is file-based. It is not configured from the CertSecure Manager UI, and changing it requires editing settings.yaml on the server followed by a service restart.

Step 5: Restart the CertSecure Backend

- Restart the service so the new settings.yaml is loaded:

```
./certsecure-linux.sh stop
./certsecure-linux.sh start
```

Note: settings.yaml is read only when the service starts, so every change to it requires a restart to take effect.

Step 6: Validate the Pipeline

- Collector is listening:

```
ss -lntp | grep 4317
```

- Collector is receiving records – temporarily add the debug exporter to the logs pipeline and watch the service log:

```
sudo journalctl -u otelcol -f
```

- CertSecure is emitting – confirm logs/certsecure_backend.log is growing, then check logging_manager_errors.log in the same installation directory. Telemetry configuration and start-up problems are recorded there; it is the first place to look when nothing arrives.
- Destination is receiving – search the destination platform for the service name configured in settings.yaml (service.name = certsecure_backend). Records should appear within roughly 5-10 seconds of being written (5 second batch flush on the backend plus 5 seconds on the Collector).

Note: Remove the debug exporter from the pipeline once validation is complete. At verbosity: detailed it writes the full content of every record to the Collector's own log, where it is readable by anyone with access to the system journal.

Log Export Behaviour

- Every log record at or above the configured level is exported, including records raised by the components CertSecure depends on – not only CertSecure's own messages.
- Records are batched before they are sent. By default up to 2048 records are held, sent in batches of 512, flushed every 5 seconds, with a 30 second delivery timeout.
- Delivery is best-effort. If the Collector is unreachable, delivery is retried and records are discarded once the buffer is full. CertSecure is never blocked or slowed by an unavailable Collector, and local log files continue to be written as normal.
- Pending records are flushed when the service is shut down cleanly.