# Collecting Prometheus Histograms and Exporting to OTLP

## Introduction

Modern distributed systems generate large volumes of telemetry to help engineers monitor performance, detect anomalies, and ensure reliability. A common pattern is to expose metrics in Prometheus format at a `/metrics` endpoint, including histograms that capture request latency and other timing distributions.

Some observability backends, however, accept metrics only in OTLP (OpenTelemetry Protocol). In these cases, Prometheus-native histograms must be collected, converted, and exported through an intermediate component before ingestion.

**Purpose of the guide**
This document explains how to:

- Scrape Prometheus-native histograms from `/metrics`.
- Transform them into OTLP format.
- Export them to an OTLP-compatible backend using the OpenTelemetry Collector.

## Prerequisites

Before you begin, ensure you have:

- A running application exposing metrics at `/metrics`.
- Prometheus-style histograms in those metrics.
- OpenTelemetry Collector installed.
- Access to an OTLP-compatible backend.
- Basic familiarity with YAML configuration.

## Collecting and Transforming Prometheus Histograms

This task explains how to collect Prometheus histograms and export them in OTLP format.
We majorly have 3 steps to achive this.

1. Configure Prometheus Receiver
2. Transform Histograms
3. Configure OTLP Exporter
4. Define the metrics pipeline.
5. Run and validate the setup.

**1. Configure Prometheus Receiver**
The Prometheus receiver in the OpenTelemetry Collector is responsible for scraping metrics from your application's `/metrics` endpoint.

```
receivers:
  prometheus:
    config:
      scrape_configs:
        - job_name: 'my-app'          # Label for this scrape job
          scrape_interval: 15s        # How often to scrape
          static_configs:
        - targets: ['localhost:8080']  # Replace with your app's metrics endpoint
```

Metrics are scraped at the defined interval and ingested into the Collector.

**2. Transform Histograms**
Prometheus histograms need normalization into OTLP format. The Collector handles this automatically, but processors can refine behavior.
processors:

```
processors:
  cumulativetodelta:                # Convert cumulative to delta
    include:                        # Apply to selected metrics
      metrics:                      # Metric list
        - http_request_duration_seconds  # Request latency histogram
```

Histogram buckets are converted into delta values, improving backend compatibility.

**3. Configure OTLP Exporter**
The OTLP exporter sends metrics to your backend.

```
exporters:
  otlp:
    endpoint: "backend:4317"    # Replace with your backend address
    protocol: grpc              # gRPC is the default protocol
```

Metrics flow seamlessly into the OTLP backend.

**4. Define the Metrics Pipeline**
Tie receivers, processors, and exporters together.

```
service:
  pipelines:
    metrics:
      receivers: [prometheus]
      processors: [cumulativetodelta]
      exporters: [otlp]
```

A complete telemetry flow is established: scrape, transform, and export.

**5. Run and Validate**

- Start the Collector: `otelcol --config config.yaml`
- Check logs for successful scrape and export messages
- Verify metrics in your backend dashboard

Prometheus histograms are available in OTLP format for monitoring, alerting, and troubleshooting.

**Validation Strategy**
Each stage of the pipeline should be verified independently:

- The `/metrics` endpoint should return histogram data when queried directly.
- The Collector logs should show successful scrapes and exports.
- The backend should display latency distributions rather than only counters or gauges.

Introducing a temporary debug or logging exporter during rollout can help confirm that metrics are converted as expected before sending traffic to production systems.

**Security and Connectivity**
OTLP endpoints are commonly protected by TLS and authentication headers. Ensure certificates are trusted and credentials are configured before enabling export. Network policies and firewalls must allow outbound connections from the Collector to the backend.

**Conclusion**
By configuring the Prometheus receiver, adding optional processors, and defining an OTLP exporter pipeline, you ensure that Prometheus-native histograms are fully integrated into modern observability platforms. This workflow preserves critical distribution data, enabling accurate monitoring, alerting, and troubleshooting across distributed systems.