Bridging Prometheus and OTLP: Converting Native Histograms for OTLP Backends

Overview

Prometheus remains the standard for exposing application metrics, with most services exporting telemetry via a `/metrics` endpoint. At the same time, many observability backends like Datadog, New Relic, Honeycomb, and others standardize on OpenTelemetry Protocol (OTLP) as their ingestion format and do not accept Prometheus exposition directly.

This guide documents how to collect Prometheus-native histograms exposed at a `/metrics` endpoint and transform them into OTLP before forwarding them to an OTLP-only backend. The focus is on where the transformation occurs in the telemetry pipeline, why it is required, and how to configure it correctly using open-source tooling.

Key takeaways

- Prometheus-native histograms are widely used but not universally consumable
- OTLP-only backends require explicit transformation
- Prometheus remains responsible for scraping
- The OpenTelemetry Collector performs protocol and data-model translation
- Histogram semantics are preserved across systems when configured correctly

Prerequisites

This guide assumes familiarity with Prometheus-style metrics and Kubernetes-based observability pipelines. Before starting, ensure you have the following in place:

Technical knowledge

- Working knowledge of **Prometheus metrics**, including counters, gauges, and histograms
- Basic understanding of **OTLP** and the OpenTelemetry data model
- Familiarity with **Kubernetes** concepts such as Deployments, Services, and port-forwarding
- Comfort using the command line and reading YAML-based configuration

Tool

- **Kubernetes cluster** (v1.24+ recommended) with network access to pull public container images
 (Any managed or self-hosted cluster works; examples assume a managed cluster.)
- **kubectl** installed and configured to access the cluster
- **Prometheus** deployed in the cluster and scraping targets via `/metrics`
- **OpenTelemetry Collector** (contrib distribution) available for metric ingestion and transformation
- An **OTLP-compatible backend** (metrics-only is sufficient) that does **not** accept Prometheus exposition format directly

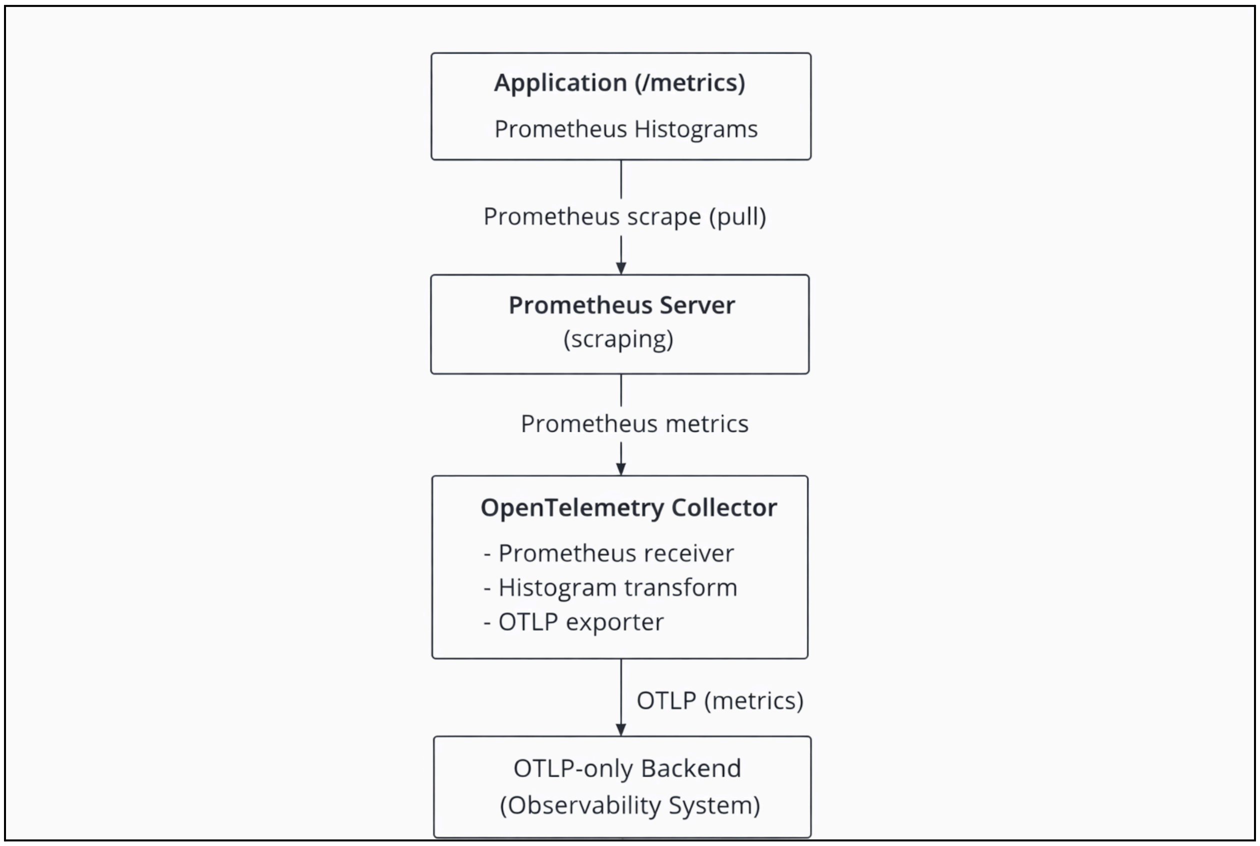
Application requirements

- A workload that exposes **Prometheus-native histograms** at a `/metrics` endpoint
 (for example, an application instrumented using `client_golang`, `client_python`, or similar libraries)
- The histogram must be exported using Prometheus' native histogram format (not pre-aggregated summaries)

Access

- Permission to deploy workloads and services in the cluster
- Ability to port-forward services locally for validation and debugging
- Permission to configure Prometheus scrape jobs and OpenTelemetry Collector pipelines

Architecture and data flow



At a high level, the telemetry pipeline looks like this:

1. An application exposes Prometheus metrics, including native histograms, at `/metrics`
2. Prometheus scrapes the endpoint using its standard pull model
3. The OpenTelemetry Collector ingests the scraped metrics
4. The Collector transforms Prometheus histogram data into the OTLP data model
5. Metrics are exported to an OTLP-only backend

Exposing Prometheus-Native Histograms at `/metrics`

The example application used in this guide exposes synthetic request latency metrics as a Prometheus histogram. The /metrics endpoint includes the familiar histogram components:

Metrics	**Description**	**Representation**

_bucket	Cumulative counts per latency bucket	The number of observations with values less than or equal to the bucket's upper bound (`le` label). Each bucket includes all observations from previous buckets.
_count	Total observations	The total number of recorded observations across all buckets. Equivalent to the `+Inf` bucket value.
_sum	Sum of observed values	The cumulative sum of all observed values (for example, total request latency in seconds), used to calculate averages.

Each histogram is labeled by request route (for example, /login, /checkout, /) for latency distributions to be analyzed per endpoint. When accessing the application's /metrics endpoint directly, you should see output similar to the following:

These represent a Prometheus-native histogram. This means that the histogram is defined using explicit bucket boundaries and cumulative counts rather than OTLP's exponential or explicit histogram representation.

Validating Histogram data in Prometheus

Before introducing any transformation logic, confirm that Prometheus is scraping the application correctly and that histogram data behaves as expected.

Confirming Target Health

Navigate to **Status** → **Targets** in the Prometheus UI and verify that the demo application target is marked as `UP`. This confirms:

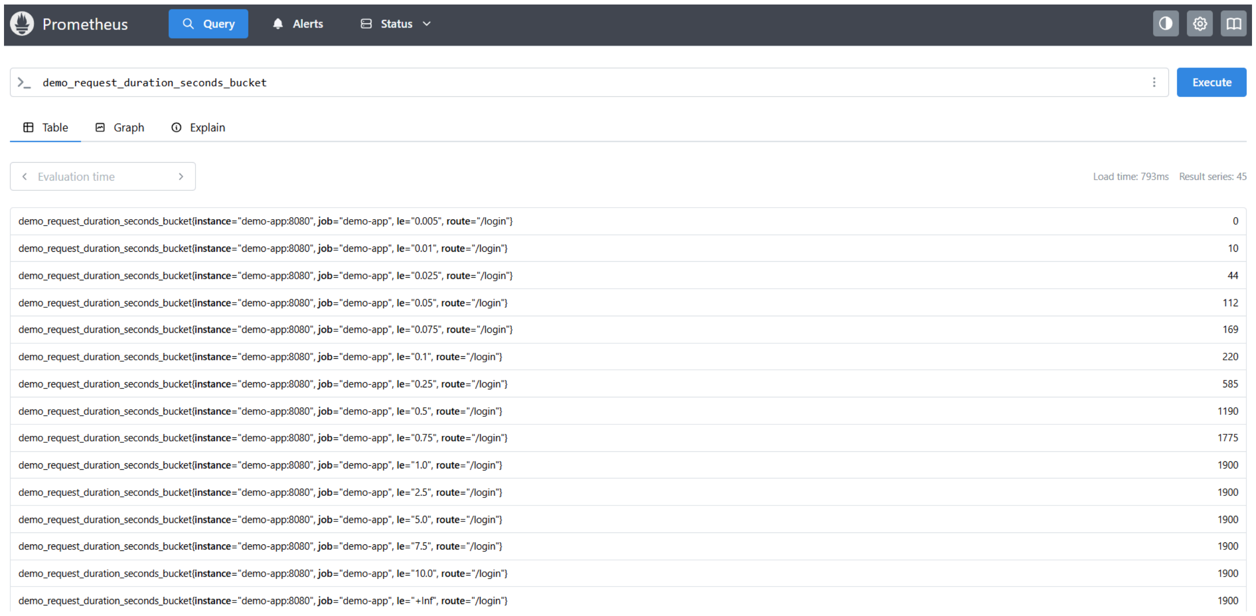
- Prometheus can reach the `/metrics` endpoint
- Scrapes are succeeding
- Labels such as `job` and `instance` are applied correctly

Querying Histogram Buckets

Next, query the histogram buckets directly:

```SQL
demo_request_duration_seconds_bucket
```

This returns cumulative bucket counts across all routes and bucket boundaries.



```
demo_request_duration_seconds_bucket
```

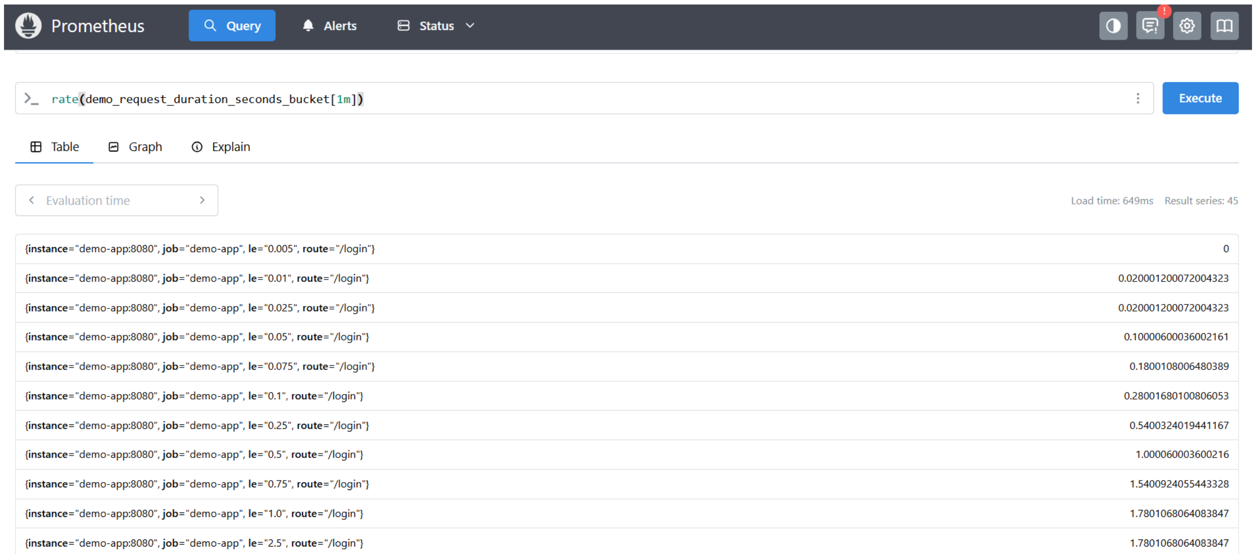
Load time: 793ms   Result series: 45

```
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.005", route="/login"}    0
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.01", route="/login"}     10
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.025", route="/login"}    44
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.05", route="/login"}     112
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.075", route="/login"}    169
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.1", route="/login"}      220
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.25", route="/login"}     585
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.5", route="/login"}      1190
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="0.75", route="/login"}     1775
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="1.0", route="/login"}      1900
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="2.5", route="/login"}      1900
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="5.0", route="/login"}      1900
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="7.5", route="/login"}      1900
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="10.0", route="/login"}     1900
demo_request_duration_seconds_bucket{instance="demo-app:8080", job="demo-app", le="+Inf", route="/login"}     1900
```

To validate the ingestion rate and histogram activity, query the per-second rate:

```SQL
rate(demo_request_duration_seconds_bucket[1m])
```

This confirms that bucket counters are increasing over time and that Prometheus is correctly processing histogram updates.



```
rate(demo_request_duration_seconds_bucket[1m])
```

Load time: 649ms   Result series: 45

{**instance**="demo-app:8080", **job**="demo-app", **le**="0.005", **route**="/login"}	0
{**instance**="demo-app:8080", **job**="demo-app", **le**="0.01", **route**="/login"}	0.020001200072004323
{**instance**="demo-app:8080", **job**="demo-app", **le**="0.025", **route**="/login"}	0.020001200072004323
{**instance**="demo-app:8080", **job**="demo-app", **le**="0.05", **route**="/login"}	0.10000600036002161
{**instance**="demo-app:8080", **job**="demo-app", **le**="0.075", **route**="/login"}	0.1800108006480389
{**instance**="demo-app:8080", **job**="demo-app", **le**="0.1", **route**="/login"}	0.28001680100806053
{**instance**="demo-app:8080", **job**="demo-app", **le**="0.25", **route**="/login"}	0.5400324019441167
{**instance**="demo-app:8080", **job**="demo-app", **le**="0.5", **route**="/login"}	1.000060003600216
{**instance**="demo-app:8080", **job**="demo-app", **le**="0.75", **route**="/login"}	1.5400924055443328
{**instance**="demo-app:8080", **job**="demo-app", **le**="1.0", **route**="/login"}	1.7801068064083847
{**instance**="demo-app:8080", **job**="demo-app", **le**="2.5", **route**="/login"}	1.7801068064083847

Now, Prometheus is functioning correctly. But the data is still in Prometheus' native format.

Why transformation is required before OTLP Export

Prometheus exposition format and OTLP metrics use fundamentally different data models.

Prometheus histograms:

- Represent latency distributions using cumulative bucket counters
- Rely on query-time aggregation
- Encode histogram structure implicitly through metric naming and labels

OTLP histograms:

- Encode histogram structure explicitly in the metric payload
- Support defined aggregation temporality
- Are consumed directly by downstream backends without PromQL

Because of these differences:

- Prometheus cannot natively emit OTLP
- OTLP backends cannot ingest the Prometheus exposition format
- Exporting Prometheus metrics "as-is" is insufficient

The OpenTelemetry Collector provides the necessary translation layer. It converts Prometheus histogram buckets into OTLP histogram data structures while preserving metric meaning.

Configuring the OpenTelemetry Collector for Prometheus Ingestion

The OpenTelemetry Collector acts as a **metrics gateway**, ingesting Prometheus data and exporting OTLP.

The Collector pipeline follows a standard pattern: `receivers → processors → exporters`.

Prometheus Receiver

The Prometheus receiver is configured to scrape the same endpoints Prometheus would normally scrape. This allows the Collector to ingest Prometheus-formatted metrics without changing application instrumentation.

At this stage:

- Histogram buckets remain in Prometheus form
- No semantic transformation has occurred yet
- Metrics are still cumulative counters

This separation of concerns ensures Prometheus remains the system of record for scraping, while the Collector handles downstream interoperability.

Transforming Prometheus Histograms to OTLP

Histogram transformation occurs inside the OpenTelemetry Collector pipeline.

During this stage, the Collector:

- Interprets `_bucket`, `_count`, and `_sum` series
- Reconstructs histogram structure
- Converts cumulative counters into OTLP histogram data points
- Preserves labels as OTLP attributes

Transformation considerations include:

- Aggregation temporality (cumulative vs delta)
- Bucket boundary preservation
- Cardinality control to avoid label explosion

Once transformed, the histogram is no longer a collection of counters. It becomes a structured OTLP histogram that downstream systems can process without PromQL. This is the most important conceptual step in the pipeline.

Exporting Metrics to an OTLP-only backend

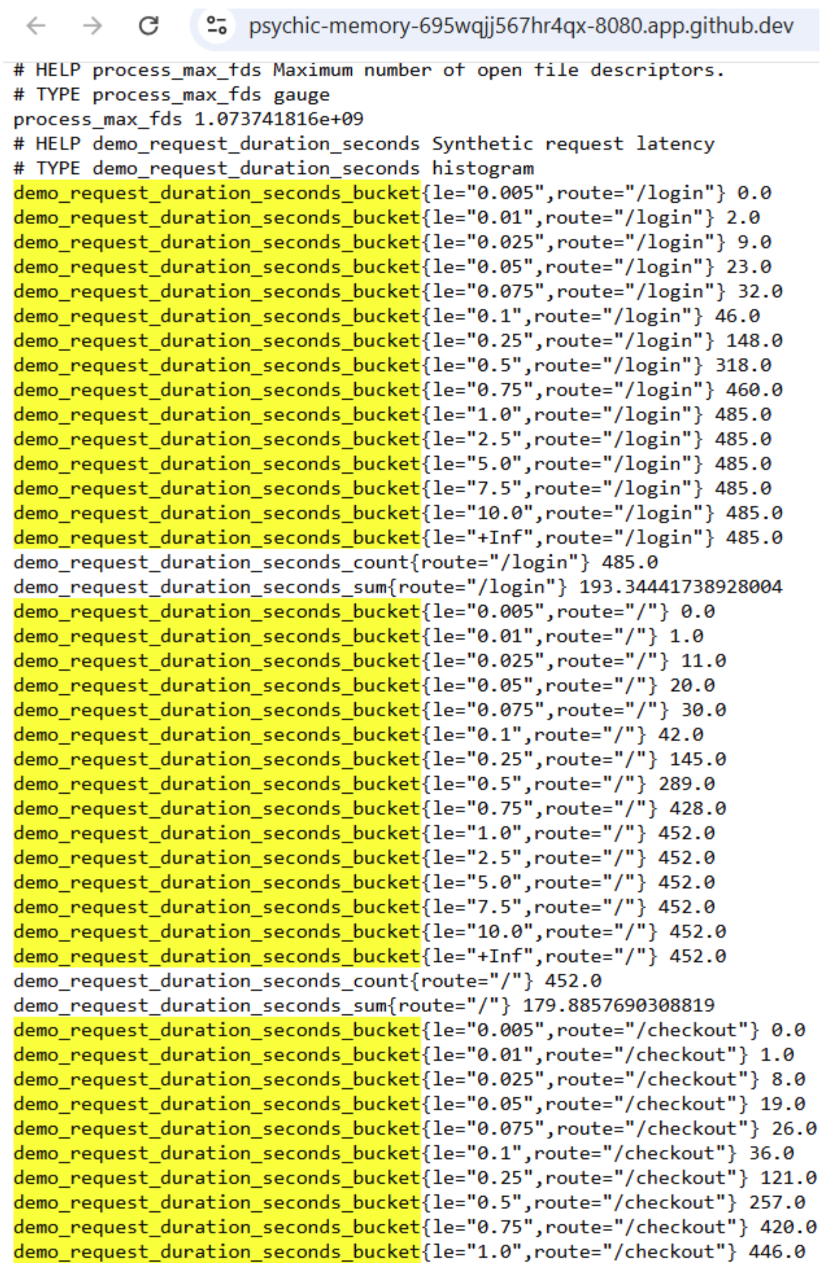
After transformation, metrics are forwarded using an OTLP exporter.

OTLP exporters support:

- gRPC (`otlp`)
- HTTP (`otlphttp`)

The backend now receives:

- Explicit histogram data
- Standardized OTLP attributes
- Metrics that no longer depend on Prometheus query semantics



```
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.073741816e+09
# HELP demo_request_duration_seconds Synthetic request latency
# TYPE demo_request_duration_seconds histogram
demo_request_duration_seconds_bucket{le="0.005",route="/login"} 0.0
demo_request_duration_seconds_bucket{le="0.01",route="/login"} 2.0
demo_request_duration_seconds_bucket{le="0.025",route="/login"} 9.0
demo_request_duration_seconds_bucket{le="0.05",route="/login"} 23.0
demo_request_duration_seconds_bucket{le="0.075",route="/login"} 32.0
demo_request_duration_seconds_bucket{le="0.1",route="/login"} 46.0
demo_request_duration_seconds_bucket{le="0.25",route="/login"} 148.0
demo_request_duration_seconds_bucket{le="0.5",route="/login"} 318.0
demo_request_duration_seconds_bucket{le="0.75",route="/login"} 460.0
demo_request_duration_seconds_bucket{le="1.0",route="/login"} 485.0
demo_request_duration_seconds_bucket{le="2.5",route="/login"} 485.0
demo_request_duration_seconds_bucket{le="5.0",route="/login"} 485.0
demo_request_duration_seconds_bucket{le="7.5",route="/login"} 485.0
demo_request_duration_seconds_bucket{le="10.0",route="/login"} 485.0
demo_request_duration_seconds_bucket{le="+Inf",route="/login"} 485.0
demo_request_duration_seconds_count{route="/login"} 485.0
demo_request_duration_seconds_sum{route="/login"} 193.34441738928004
demo_request_duration_seconds_bucket{le="0.005",route="/"} 0.0
demo_request_duration_seconds_bucket{le="0.01",route="/"} 1.0
demo_request_duration_seconds_bucket{le="0.025",route="/"} 11.0
demo_request_duration_seconds_bucket{le="0.05",route="/"} 20.0
demo_request_duration_seconds_bucket{le="0.075",route="/"} 30.0
demo_request_duration_seconds_bucket{le="0.1",route="/"} 42.0
demo_request_duration_seconds_bucket{le="0.25",route="/"} 145.0
demo_request_duration_seconds_bucket{le="0.5",route="/"} 289.0
demo_request_duration_seconds_bucket{le="0.75",route="/"} 428.0
demo_request_duration_seconds_bucket{le="1.0",route="/"} 452.0
demo_request_duration_seconds_bucket{le="2.5",route="/"} 452.0
demo_request_duration_seconds_bucket{le="5.0",route="/"} 452.0
demo_request_duration_seconds_bucket{le="7.5",route="/"} 452.0
demo_request_duration_seconds_bucket{le="10.0",route="/"} 452.0
demo_request_duration_seconds_bucket{le="+Inf",route="/"} 452.0
demo_request_duration_seconds_count{route="/"} 452.0
demo_request_duration_seconds_sum{route="/"} 179.8857690308819
demo_request_duration_seconds_bucket{le="0.005",route="/checkout"} 0.0
demo_request_duration_seconds_bucket{le="0.01",route="/checkout"} 1.0
demo_request_duration_seconds_bucket{le="0.025",route="/checkout"} 8.0
demo_request_duration_seconds_bucket{le="0.05",route="/checkout"} 19.0
demo_request_duration_seconds_bucket{le="0.075",route="/checkout"} 26.0
demo_request_duration_seconds_bucket{le="0.1",route="/checkout"} 36.0
demo_request_duration_seconds_bucket{le="0.25",route="/checkout"} 121.0
demo_request_duration_seconds_bucket{le="0.5",route="/checkout"} 257.0
demo_request_duration_seconds_bucket{le="0.75",route="/checkout"} 420.0
demo_request_duration_seconds_bucket{le="1.0",route="/checkout"} 446.0
```

Prometheus-native histogram metrics exposed at the `/metrics` *endpoint, showing cumulative bucket counts, total count, and sum.*

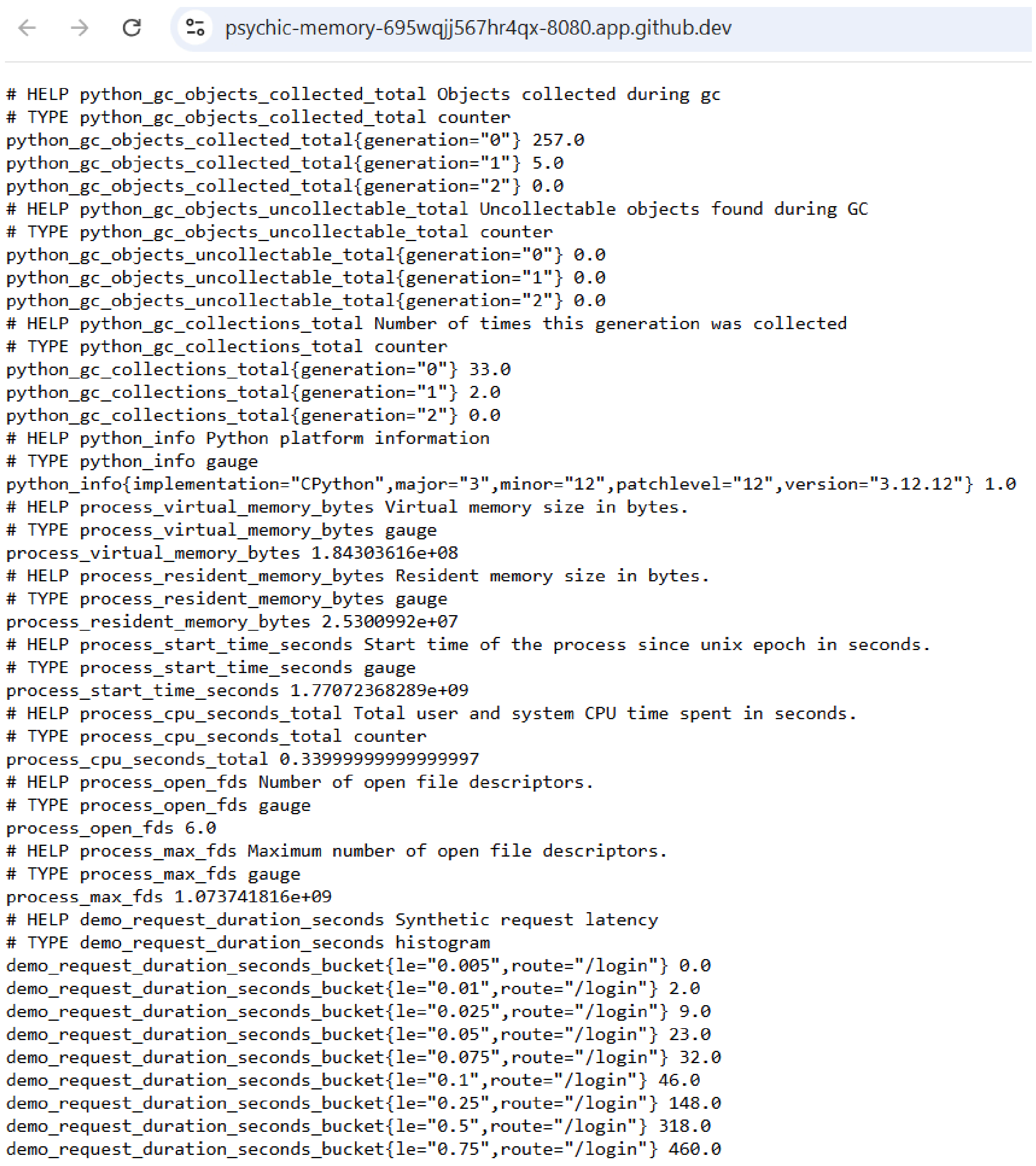
At this point, Prometheus is no longer required for visualization or querying in the backend. Its role is limited to scraping and initial aggregation.

Verifying the End-to-End Pipeline

Verification should happen at multiple layers.

Application Layer

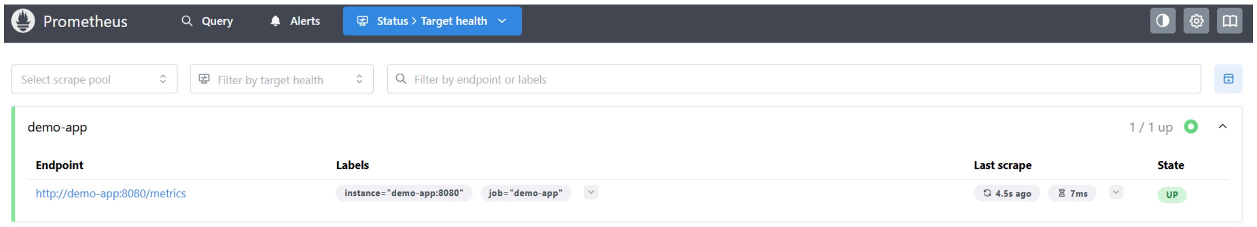
Confirm the /metrics endpoint still exposes histogram data.



psychic-memory-695wqjj567hr4qx-8080.app.github.dev

```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 257.0
python_gc_objects_collected_total{generation="1"} 5.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable objects found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 33.0
python_gc_collections_total{generation="1"} 2.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="12",patchlevel="12",version="3.12.12"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.84303616e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.5300992e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.77072368289e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.33999999999999997
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 6.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.073741816e+09
# HELP demo_request_duration_seconds Synthetic request latency
# TYPE demo_request_duration_seconds histogram
demo_request_duration_seconds_bucket{le="0.005",route="/login"} 0.0
demo_request_duration_seconds_bucket{le="0.01",route="/login"} 2.0
demo_request_duration_seconds_bucket{le="0.025",route="/login"} 9.0
demo_request_duration_seconds_bucket{le="0.05",route="/login"} 23.0
demo_request_duration_seconds_bucket{le="0.075",route="/login"} 32.0
demo_request_duration_seconds_bucket{le="0.1",route="/login"} 46.0
demo_request_duration_seconds_bucket{le="0.25",route="/login"} 148.0
demo_request_duration_seconds_bucket{le="0.5",route="/login"} 318.0
demo_request_duration_seconds_bucket{le="0.75",route="/login"} 460.0
```

Prometheus Layer



demo-app — 1 / 1 up

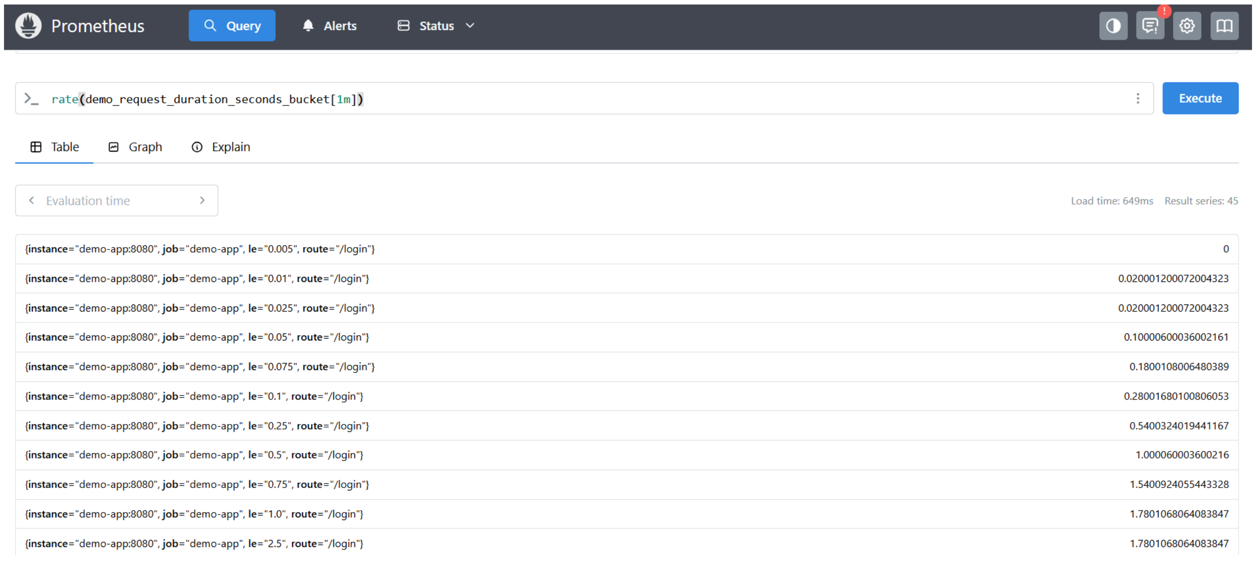
Endpoint	Labels	Last scrape	State
http://demo-app:8080/metrics	instance="demo-app:8080" job="demo-app"	4.5s ago 7ms	UP

Prometheus target health confirming successful scraping of the demo application

Verify:

- Target remains UP
- Histogram buckets continue to increment
- Rate queries behave as expected

OTLP transformation validation



```
rate(demo_request_duration_seconds_bucket[1m])
```

Load time: 649ms   Result series: 45

```
{instance="demo-app:8080", job="demo-app", le="0.005", route="/login"}    0
{instance="demo-app:8080", job="demo-app", le="0.01", route="/login"}     0.020001200072004323
{instance="demo-app:8080", job="demo-app", le="0.025", route="/login"}    0.020001200072004323
{instance="demo-app:8080", job="demo-app", le="0.05", route="/login"}     0.10000600036002161
{instance="demo-app:8080", job="demo-app", le="0.075", route="/login"}    0.1800108006480389
{instance="demo-app:8080", job="demo-app", le="0.1", route="/login"}      0.28001680100806053
{instance="demo-app:8080", job="demo-app", le="0.25", route="/login"}     0.5400324019441167
{instance="demo-app:8080", job="demo-app", le="0.5", route="/login"}      1.000060003600216
{instance="demo-app:8080", job="demo-app", le="0.75", route="/login"}     1.5400924055443328
{instance="demo-app:8080", job="demo-app", le="1.0", route="/login"}      1.7801068064083847
{instance="demo-app:8080", job="demo-app", le="2.5", route="/login"}      1.7801068064083847
```

Histogram bucket data continues to be available in Prometheus while being transformed and exported as OTLP

Query histogram metrics again using PromQL to confirm:

- Buckets are still present
- Data continues to flow while being exported downstream

At this point, the pipeline is confirmed end-to-end:

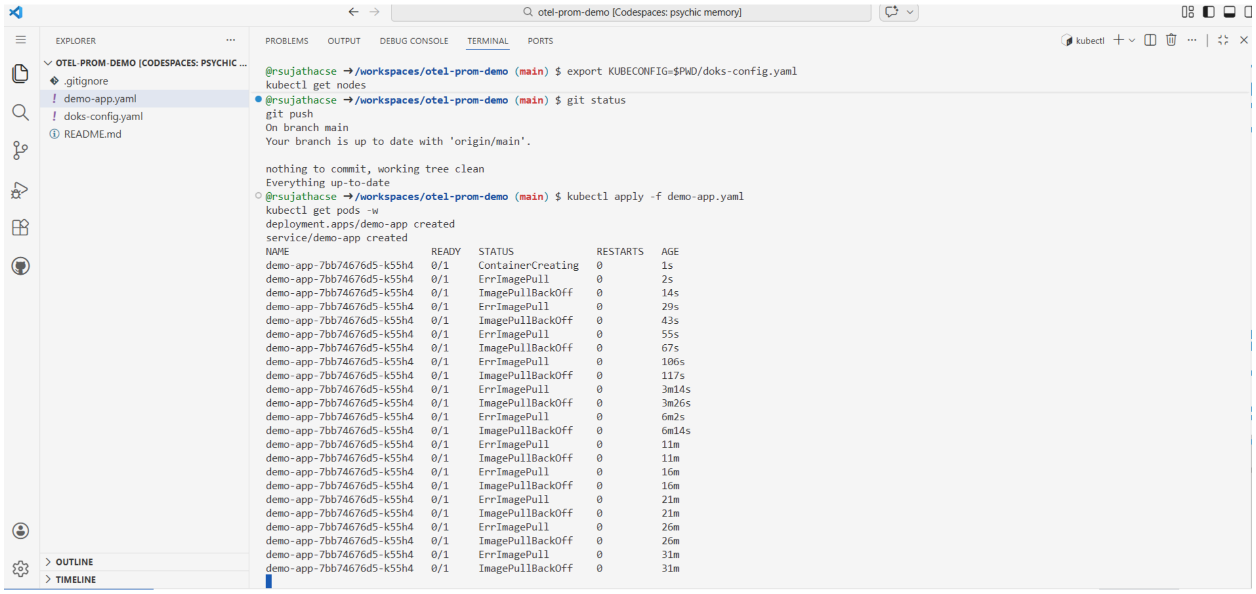
- Metrics originate at `/metrics`

- Prometheus scrapes successfully
- The OpenTelemetry Collector transforms histogram data
- OTLP metrics are exported without loss of fidelity

Edge cases and debugging tips

Debugging is easier when each pipeline stage is validated independently.

- **ImagePullBackOff or scrape failures**: Validate container images and network access



```
@rsujathacse →/workspaces/otel-prom-demo (main) $ export KUBECONFIG=$PWD/doks-config.yaml
kubectl get nodes
@rsujathacse →/workspaces/otel-prom-demo (main) $ git status
git push
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
Everything up-to-date
@rsujathacse →/workspaces/otel-prom-demo (main) $ kubectl apply -f demo-app.yaml
kubectl get pods -w
deployment.apps/demo-app created
service/demo-app created
NAME                        READY   STATUS              RESTARTS   AGE
demo-app-7bb74676d5-k55h4   0/1     ContainerCreating   0          1s
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          2s
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          14s
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          29s
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          43s
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          55s
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          67s
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          106s
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          117s
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          3m14s
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          3m26s
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          6m2s
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          6m14s
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          11m
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          11m
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          16m
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          16m
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          21m
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          21m
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          26m
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          26m
demo-app-7bb74676d5-k55h4   0/1     ErrImagePull        0          31m
demo-app-7bb74676d5-k55h4   0/1     ImagePullBackOff    0          31m
```

- **Histogram missing in Prometheus**: Confirm instrumentation and scrape configuration
- **Metrics not exported**: Check Collector exporter logs
- **Unexpected bucket counts**: Validate aggregation temporality
- **High cardinality**: Review label dimensions before exporting