# OpenTelemetry. The Domino Effect: How a Minor Feature Redefined the Game. My story.

Sergii Bieliaievskyi · Follow

7 min read · Feb 7, 2024

Listen Share

From JIRA ticket to an article. Story #3

Photo by Luke Chesser on Unsplash

> An article shouldn't always be a work of art; otherwise, an author might wait ages for a suitable topic. I believe a story created from a JIRA ticket (or a similar system) could also be interesting and find its audience.

Today I would like to discuss OpenTelemetry. But if you think this is going to be yet another “bla-bla” article describing how to install OTEL(OpenTelemetry) you are wrong (I hope…).

## The basis.

- Kubernetes cluster.
- OpenTelemetry ver 0.91.0
- Grafana Loki ver 2.8.1 (Spoilers!!!)

Photo by Simon Delalande on Unsplash

OpenTelemetry is an open-source observability framework designed to simplify the collection, processing, and visualization of telemetry data in distributed systems. Its significance lies in providing a unified standard for instrumenting code across various programming languages, enabling seamless monitoring and tracing of applications. By offering a vendor-agnostic approach, OpenTelemetry addresses the challenge of interoperability among different monitoring tools, fostering a more collaborative and adaptable ecosystem. One key problem it solves is the complexity associated with gaining insights into the performance and behavior of microservices architectures. OpenTelemetry standardizes instrumentation practices, allowing developers to effortlessly incorporate telemetry into their applications, regardless of the underlying technologies. Additionally, it plays a pivotal role in troubleshooting and debugging, facilitating the identification and resolution of performance bottlenecks or errors in distributed environments. As a result, OpenTelemetry contributes to enhanced system reliability, faster issue resolution, and an overall improved user experience in modern, cloud-native applications.

No doubt, that is what OpenTelemetry was created for. But let's forget about this for a moment and try to look at OTEL as a Kubernetes workload. The majority of users don't use a bare Kubernetes cluster without add-ons like cert-manager, coredns, ingress controller, kyverno, sealed secrets, karpenter and so on and so forth. A lot of pods must be running in a fresh kubernetes cluster, even before commercial workloads are scheduled. This is especially important in a cloud environment, where we pay for everything we run or use. Wouldn't it be nice to reduce the number of running pods by installing OpenTelemetry and benefiting from observability along the way? Let me explain what I mean. Can you imagine a kubernetes cluster without monitoring - prometheus, grafana, ELK/Loki? Probably no. Such things like promtail(takes logs from hosts and sends them to Loki), eventrouter(stores kubernetes events in Loki database. I wrote about it HERE), prometheus node_exporter are crucial part of every monitoring. OpenTelemetry is distributed together with embedded receivers. The most interesting for us are:

- **filelog**=promtail
- **k8sobjects**=eventrouter
- **hostmetrics**=node_exporter.

As they are embedded, only a single container will be deployed, which is good from a cost management standpoint.

I chose, the so-called, Agent deployment. (https://opentelemetry.io/docs/collector/deployment/agent/)

Here is a full config file:

```
exporters:
    debug: {}
    logging: {}
    loki:
      default_labels_enabled:
        exporter: true
        job: true
      endpoint: http://loki-write.monitoring.svc.cluster.local:3100/loki/api/
    otlp:
      endpoint: jaeger-tracing-collector.monitoring.svc:4317
      tls:
        insecure: true
        insecure_skip_verify: true
    prometheus:
      endpoint: 0.0.0.0:55999
  extensions:
    health_check: {}
  processors:
    batch: {}
    k8sattributes:
      extract:
        labels:
        - from: pod
          key_regex: (.*)
          tag_name: $$1
        metadata:
        - k8s.namespace.name
        - k8s.deployment.name
        - k8s.statefulset.name
        - k8s.daemonset.name
        - k8s.cronjob.name
        - k8s.job.name
        - k8s.node.name
        - k8s.pod.name
        - k8s.pod.uid
        - k8s.pod.start_time
      filter:
        node_from_env_var: K8S_NODE_NAME
      passthrough: false
      pod_association:
      - sources:
        - from: resource_attribute
          name: k8s.pod.ip
      - sources:
        - from: resource_attribute
          name: k8s.pod.uid
      - sources:
        - from: connection
    memory_limiter:
      check_interval: 5s
      limit_percentage: 80
      spike_limit_percentage: 25
    resource:
      attributes:
      - action: insert
        from_attribute: k8s.pod.name
        key: pod_name
      - action: insert
        key: loki.resource.labels
        value: pod_name,app
  receivers:
    filelog:
      exclude: []
      include:
      - /var/log/pods/*/*/*.log
      include_file_name: false
      include_file_path: true
      operators:
      - id: parser-containerd
        output: containerd-recombine
        regex: ^(?P<time>[^ Z]+Z)\s(?P<stream>stdout|stderr)\s(?P<logtag>[^ ]
        type: regex_parser
      - combine_field: attributes.log
        combine_with: ""
        id: containerd-recombine
        is_last_entry: attributes.logtag == 'F'
        max_log_size: 102400
        output: extract_metadata_from_filepath
        source_identifier: attributes["log.file.path"]
        type: recombine
      - id: extract_metadata_from_filepath
        parse_from: attributes["log.file.path"]
        regex: ^.*\/(?P<namespace>[^_]+)_(?P<pod_name>[^_]+)_(?P<uid>[a-f0-9\
        type: regex_parser
      - from: attributes.container_name
        to: resource["k8s.container.name"]
        type: move
      - from: attributes.namespace
        to: resource["k8s.namespace.name"]
        type: move
      - from: attributes.pod_name
        to: resource["k8s.pod.name"]
        type: move
      - from: attributes.restart_count
        to: resource["k8s.container.restart_count"]
        type: move
      - from: attributes.uid
        to: resource["k8s.pod.uid"]
        type: move
      - from: attributes.log
        to: body
        type: move
      preserve_leading_whitespaces: false
      preserve_trailing_whitespaces: true
      start_at: end
    k8sobjects:
      objects:
      - mode: pull
        name: pods
      - mode: watch
        name: events
    otlp:
      protocols:
        grpc:
          endpoint: ${env:MY_POD_IP}:4317
        http:
          endpoint: ${env:MY_POD_IP}:4318
    prometheus:
      config:
        scrape_configs:
        - job_name: opentelemetry-collector
          scrape_interval: 10s
          static_configs:
          - targets:
            - ${env:MY_POD_IP}:8888
  service:
    extensions:
    - health_check
    pipelines:
      logs:
        exporters:
        - loki
        processors:
        - k8sattributes
        - resource
        receivers:
        - filelog
        - k8sobjects
      metrics:
        exporters:
        - prometheus
        processors:
        - k8sattributes
        - memory_limiter
        - batch
        receivers:
        - otlp
      traces:
        exporters:
        - otlp
        processors:
        - k8sattributes
        - memory_limiter
        - batch
        receivers:
        - otlp
    telemetry:
      metrics:
        address: ${env:MY_POD_IP}:8888
```

Trying to scrutinize, you'll probably notice that the hostmetrics configuration is missing. I wish all three receivers could substitute promtail, eventrouter and node_exporter. As you might guess hostmetrics turned out not to be a node_exporter alternative. To prove this, I can even post here what hostmetrics returns:

```
# HELP system_cpu_load_average_15m Average CPU Load over 15 minutes.
# TYPE system_cpu_load_average_15m gauge
system_cpu_load_average_15m{job="kube-state-metrics",metricsrc="opentelemetry"}
# HELP system_cpu_load_average_1m Average CPU Load over 1 minute.
# TYPE system_cpu_load_average_1m gauge
system_cpu_load_average_1m{job="kube-state-metrics",metricsrc="opentelemetry"}
# HELP system_cpu_load_average_5m Average CPU Load over 5 minutes.
# TYPE system_cpu_load_average_5m gauge
system_cpu_load_average_5m{job="kube-state-metrics",metricsrc="opentelemetry"}
# HELP system_cpu_time_seconds_total Total seconds each logical CPU spent on ea
# TYPE system_cpu_time_seconds_total counter
system_cpu_time_seconds_total{cpu="cpu0",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu0",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu0",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu0",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu0",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu0",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu0",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu0",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu1",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu1",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu1",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu1",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu1",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu1",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu1",job="kube-state-metrics",metricsrc="op
system_cpu_time_seconds_total{cpu="cpu1",job="kube-state-metrics",metricsrc="op
# HELP system_disk_io_bytes_total Disk bytes transferred.
# TYPE system_disk_io_bytes_total counter
system_disk_io_bytes_total{device="nvme0n1",direction="read",job="kube-state-me
system_disk_io_bytes_total{device="nvme0n1",direction="write",job="kube-state-m
system_disk_io_bytes_total{device="nvme0n1p1",direction="read",job="kube-state-
```

```
system_disk_io_bytes_total{device="nvme0n1p1",direction="write",job="kube-state
system_disk_io_bytes_total{device="nvme0n1p128",direction="read",job="kube-stat
system_disk_io_bytes_total{device="nvme0n1p128",direction="write",job="kube-sta
# HELP system_disk_io_time_seconds_total Time disk spent activated. On Windows,
# TYPE system_disk_io_time_seconds_total counter
system_disk_io_time_seconds_total{device="nvme0n1",job="kube-state-metrics",met
system_disk_io_time_seconds_total{device="nvme0n1p1",job="kube-state-metrics",m
system_disk_io_time_seconds_total{device="nvme0n1p128",job="kube-state-metrics"
# HELP system_disk_merged_total The number of disk reads/writes merged into sin
# TYPE system_disk_merged_total counter
system_disk_merged_total{device="nvme0n1",direction="read",job="kube-state-metr
system_disk_merged_total{device="nvme0n1",direction="write",job="kube-state-met
system_disk_merged_total{device="nvme0n1p1",direction="read",job="kube-state-me
system_disk_merged_total{device="nvme0n1p1",direction="write",job="kube-state-m
system_disk_merged_total{device="nvme0n1p128",direction="read",job="kube-state-
system_disk_merged_total{device="nvme0n1p128",direction="write",job="kube-state
# HELP system_disk_operation_time_seconds_total Time spent in disk operations.
# TYPE system_disk_operation_time_seconds_total counter
system_disk_operation_time_seconds_total{device="nvme0n1",direction="read",job=
system_disk_operation_time_seconds_total{device="nvme0n1",direction="write",job
system_disk_operation_time_seconds_total{device="nvme0n1p1",direction="read",jo
system_disk_operation_time_seconds_total{device="nvme0n1p1",direction="write",j
system_disk_operation_time_seconds_total{device="nvme0n1p128",direction="read",
system_disk_operation_time_seconds_total{device="nvme0n1p128",direction="write"
# HELP system_disk_operations_total Disk operations count.
# TYPE system_disk_operations_total counter
system_disk_operations_total{device="nvme0n1",direction="read",job="kube-state-
system_disk_operations_total{device="nvme0n1",direction="write",job="kube-state
system_disk_operations_total{device="nvme0n1p1",direction="read",job="kube-stat
system_disk_operations_total{device="nvme0n1p1",direction="write",job="kube-sta
system_disk_operations_total{device="nvme0n1p128",direction="read",job="kube-st
system_disk_operations_total{device="nvme0n1p128",direction="write",job="kube-s
# HELP system_disk_pending_operations The queue size of pending I/O operations.
# TYPE system_disk_pending_operations gauge
system_disk_pending_operations{device="nvme0n1",job="kube-state-metrics",metric
system_disk_pending_operations{device="nvme0n1p1",job="kube-state-metrics",metr
system_disk_pending_operations{device="nvme0n1p128",job="kube-state-metrics",me
# HELP system_disk_weighted_io_time_seconds_total Time disk spent activated mul
# TYPE system_disk_weighted_io_time_seconds_total counter
system_disk_weighted_io_time_seconds_total{device="nvme0n1",job="kube-state-met
system_disk_weighted_io_time_seconds_total{device="nvme0n1p1",job="kube-state-m
system_disk_weighted_io_time_seconds_total{device="nvme0n1p128",job="kube-state
# HELP system_filesystem_inodes_usage FileSystem inodes used.
# TYPE system_filesystem_inodes_usage gauge
system_filesystem_inodes_usage{device="/dev/nvme0n1p1",job="kube-state-metrics"
system_filesystem_inodes_usage{device="/dev/nvme0n1p1",job="kube-state-metrics"
# HELP system_filesystem_usage_bytes Filesystem bytes used.
# TYPE system_filesystem_usage_bytes gauge
system_filesystem_usage_bytes{device="/dev/nvme0n1p1",job="kube-state-metrics",
system_filesystem_usage_bytes{device="/dev/nvme0n1p1",job="kube-state-metrics",
system_filesystem_usage_bytes{device="/dev/nvme0n1p1",job="kube-state-metrics",
# HELP system_memory_usage_bytes Bytes of memory in use.
# TYPE system_memory_usage_bytes gauge
```

```
system_memory_usage_bytes{job="kube-state-metrics",metricsrc="opentelemetry",st
system_memory_usage_bytes{job="kube-state-metrics",metricsrc="opentelemetry",st
system_memory_usage_bytes{job="kube-state-metrics",metricsrc="opentelemetry",st
system_memory_usage_bytes{job="kube-state-metrics",metricsrc="opentelemetry",st
system_memory_usage_bytes{job="kube-state-metrics",metricsrc="opentelemetry",st
system_memory_usage_bytes{job="kube-state-metrics",metricsrc="opentelemetry",st
# HELP system_network_connections The number of connections.
# TYPE system_network_connections gauge
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
system_network_connections{job="kube-state-metrics",metricsrc="opentelemetry",p
# HELP system_network_dropped_total The number of packets dropped.
# TYPE system_network_dropped_total counter
system_network_dropped_total{device="eth0",direction="receive",job="kube-state-
system_network_dropped_total{device="eth0",direction="transmit",job="kube-state
system_network_dropped_total{device="lo",direction="receive",job="kube-state-me
system_network_dropped_total{device="lo",direction="transmit",job="kube-state-m
# HELP system_network_errors_total The number of errors encountered.
# TYPE system_network_errors_total counter
system_network_errors_total{device="eth0",direction="receive",job="kube-state-m
system_network_errors_total{device="eth0",direction="transmit",job="kube-state-
system_network_errors_total{device="lo",direction="receive",job="kube-state-met
system_network_errors_total{device="lo",direction="transmit",job="kube-state-me
# HELP system_network_io_bytes_total The number of bytes transmitted and receiv
# TYPE system_network_io_bytes_total counter
system_network_io_bytes_total{device="eth0",direction="receive",job="kube-state
system_network_io_bytes_total{device="eth0",direction="transmit",job="kube-stat
```

```
system_network_packets_total{device="eth0",direction="transmit",job="kube-state
system_network_packets_total{device="lo",direction="receive",job="kube-state-me
system_network_packets_total{device="lo",direction="transmit",job="kube-state-m
```

If we skip all commented lines and then count the total number of remaining lines, it turns out that the output contains only 93 lines. In contrast, the node-exporter returns 1164 lines. As you might guess, metric names are different as well. This puts