Telemetry Showdown

Press start

Henrik Rexed
CloudNative Advocate

Henrik Rexed

Cloud Native Advocate, CNCF Ambassador

- 15+ years of Performance engineering
- Owner of : IsitObservble

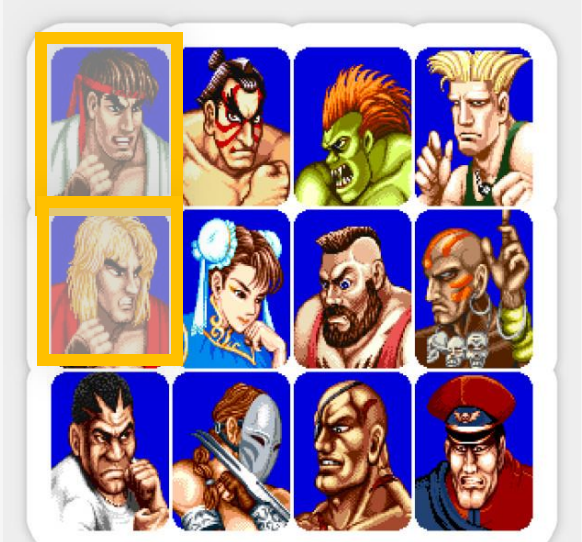
  Producer of : Perfbytes

DISCLAIMER

- No birds or telescopes were harmed in the making of this presentation
- The intention behind this talk track is not to assign blame to any CNCF project.
- This session is made to help the community in choosing their telemetry agent.

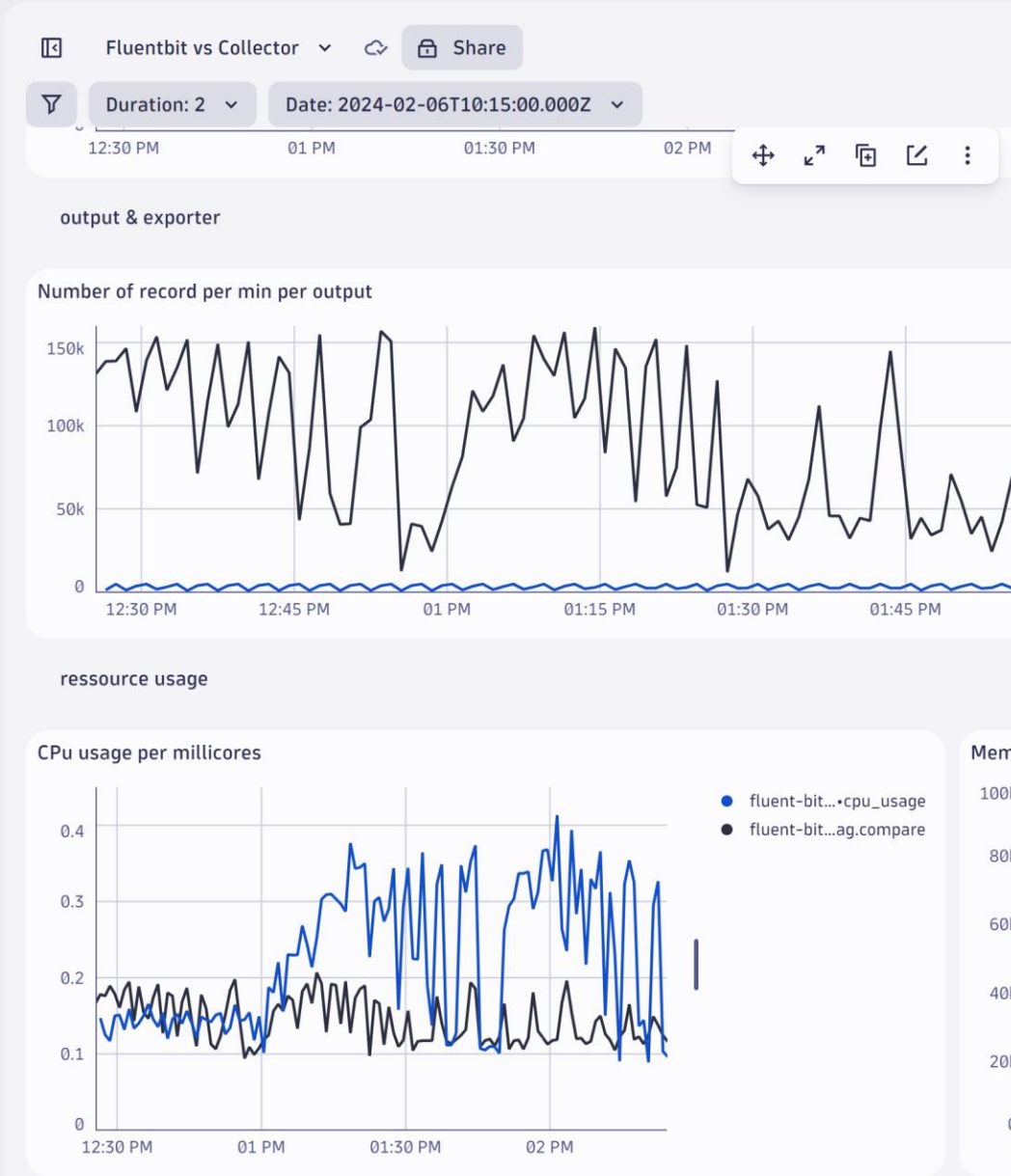
Observabilty fighter

Player Select



If you stay with me you will ...

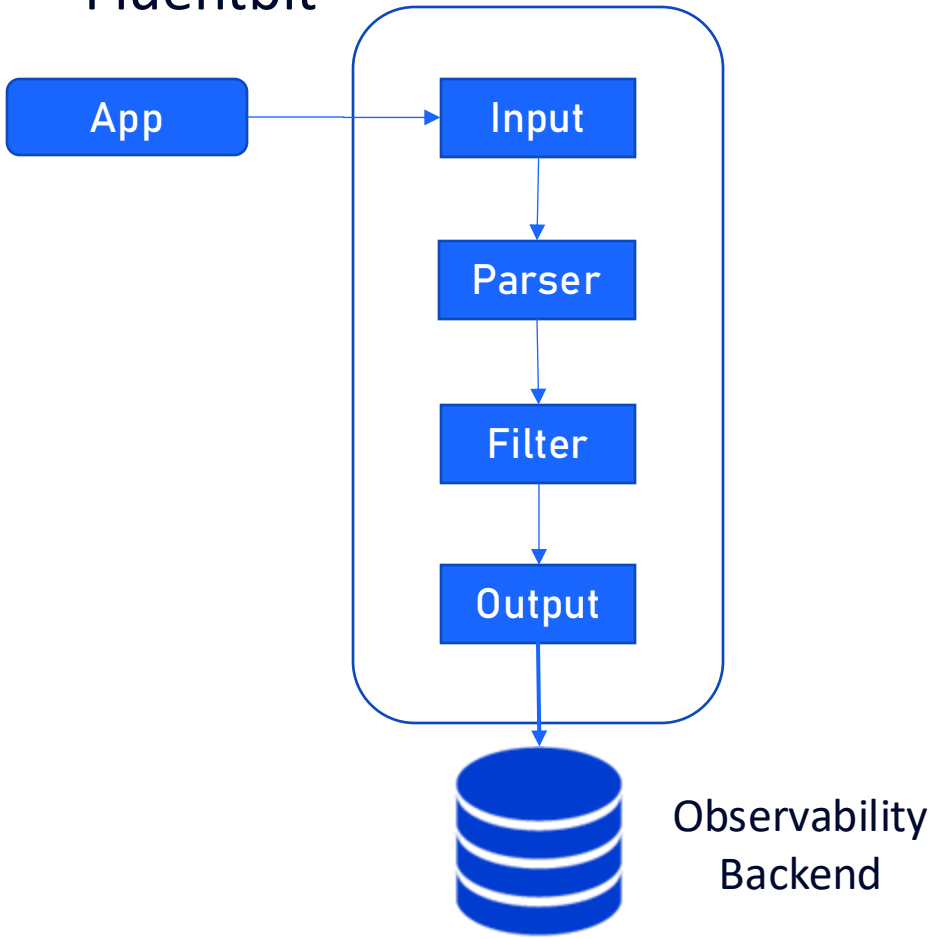
- Understand the difference between fluentbit and the OpenTelemetry collector by comparing :
  - Desing Experience
  - The plugins for :
    - Logs
    - Metrics
    - Traces
- See various Benchmarking results
- Recommendation on which agents needs to used under specific conditions



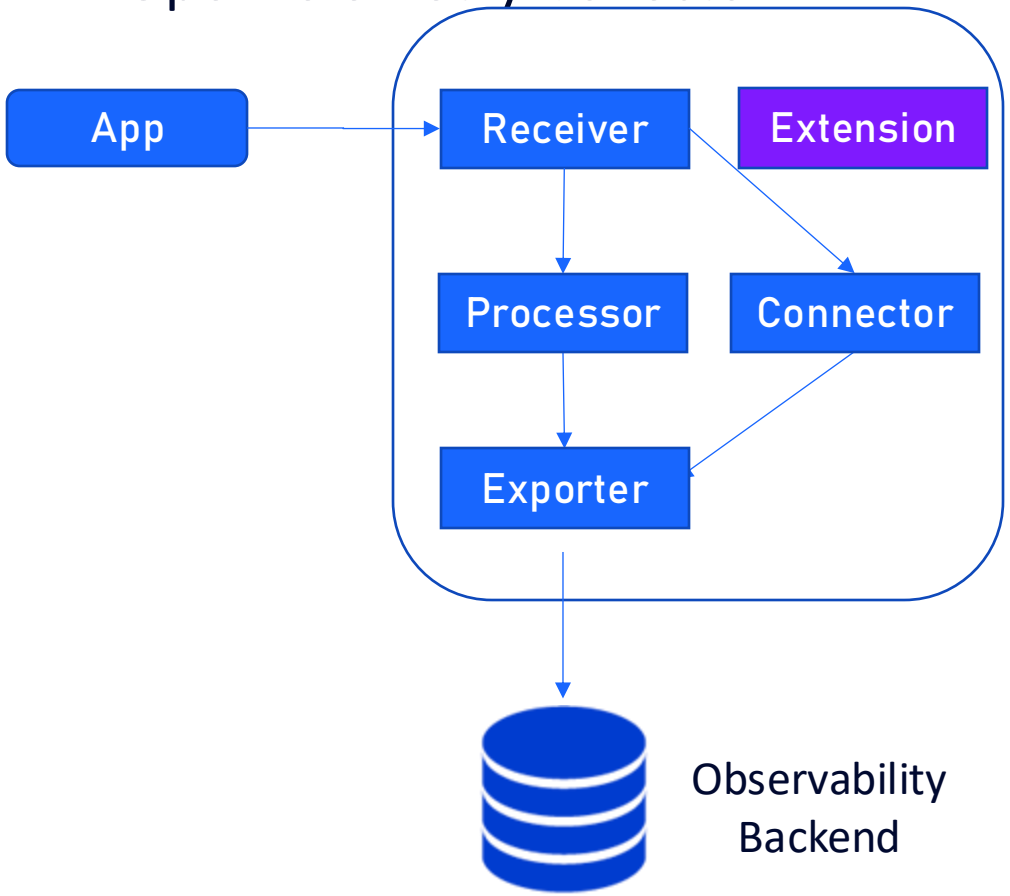
Round 1
Design

Design Patterns

- Fluentbit

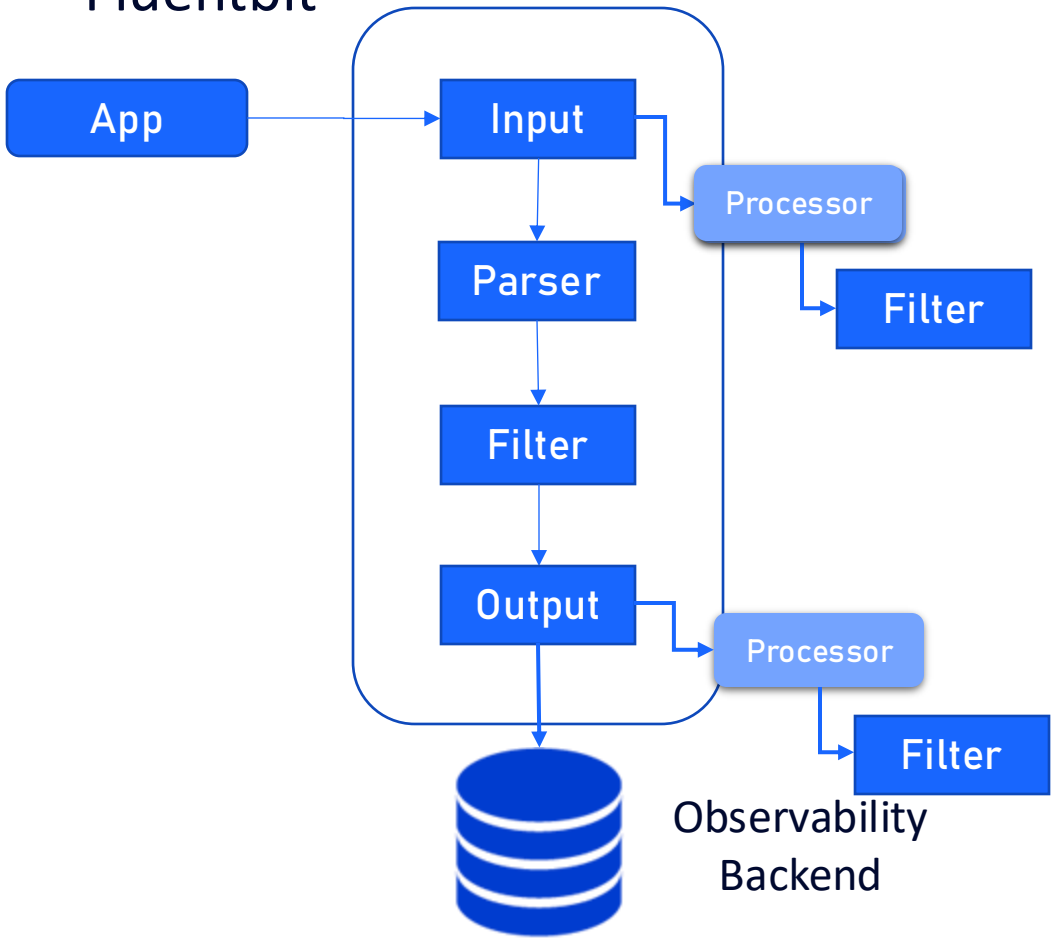


- OpenTelemetry Collector

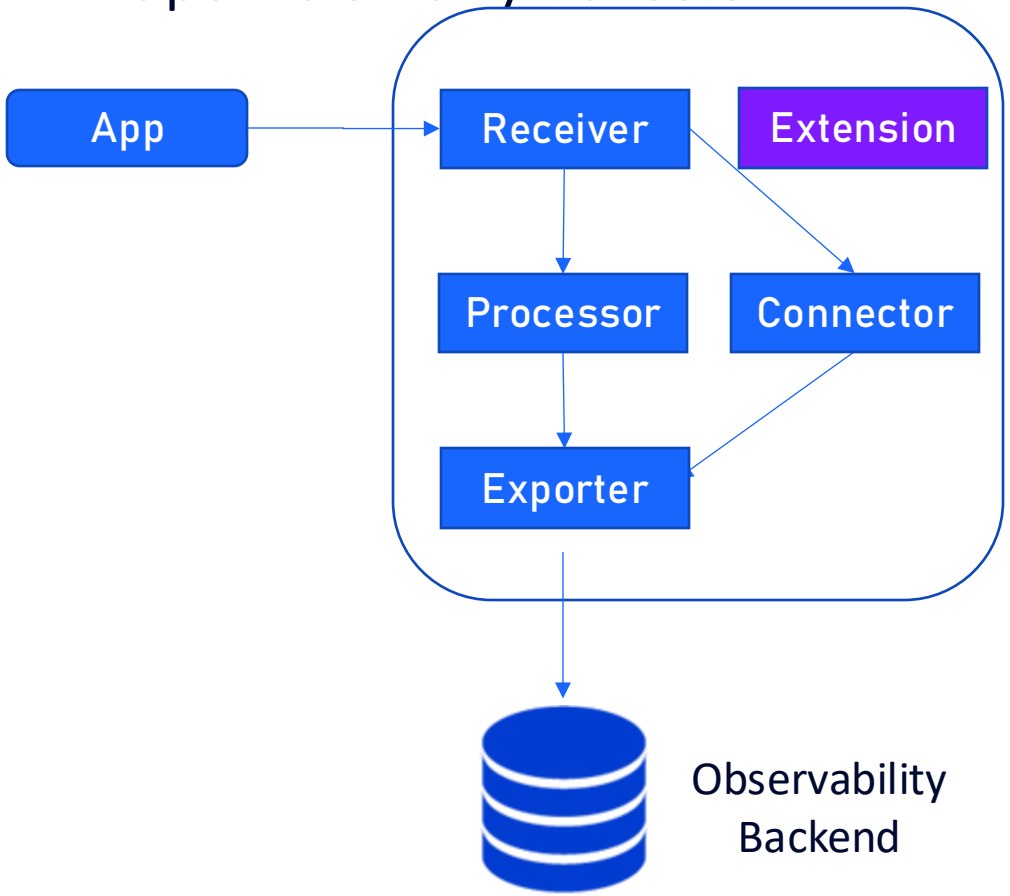


Design Patterns since Fluentbit 2.x

- Fluentbit

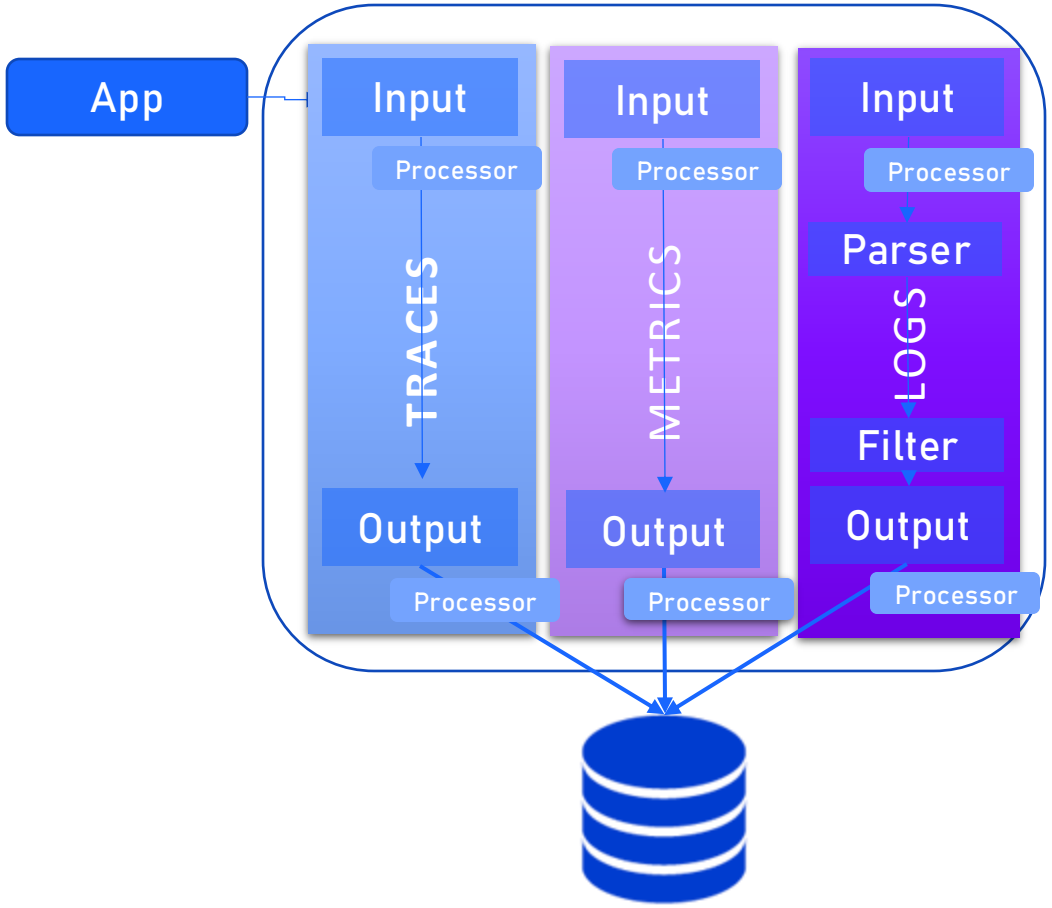


- OpenTelemetry Collector

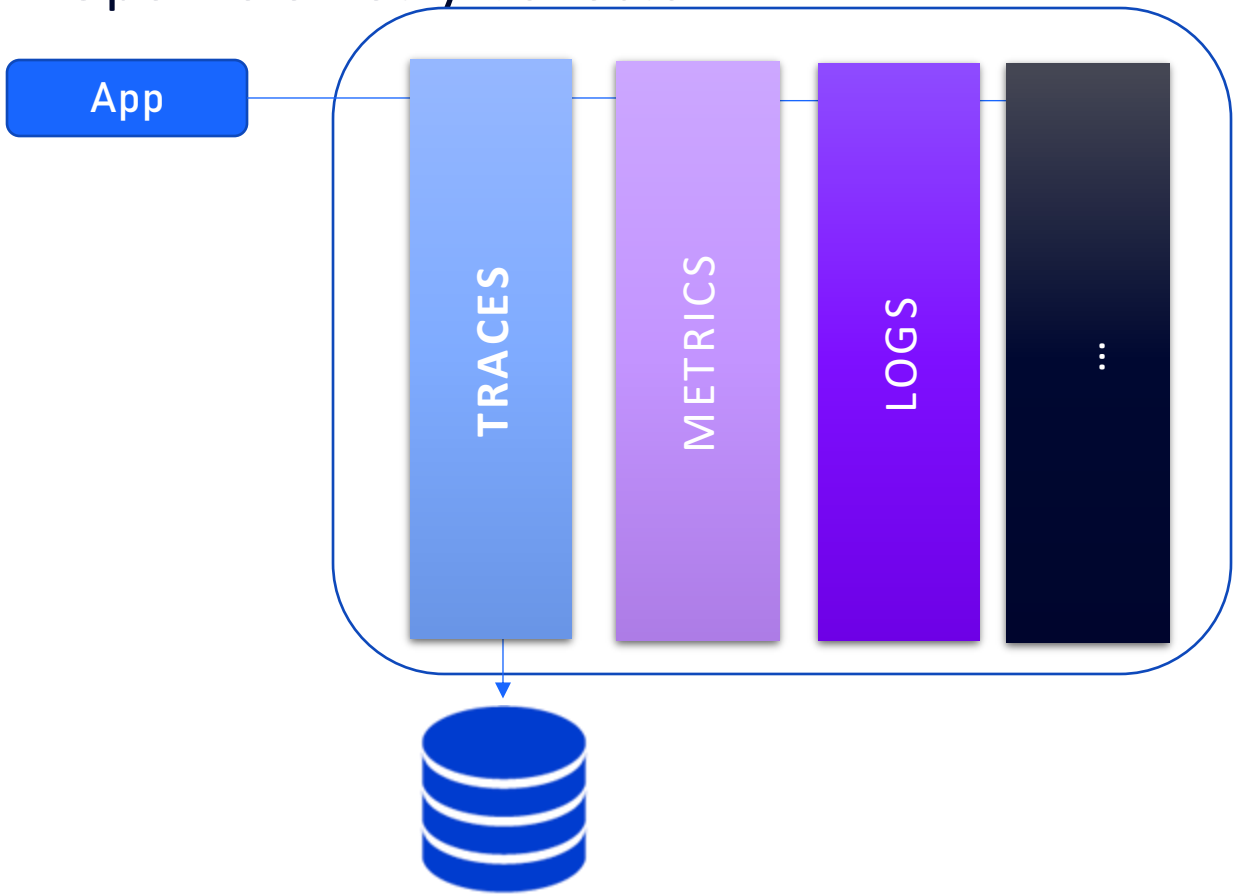


Design Patterns

- Fluentbit

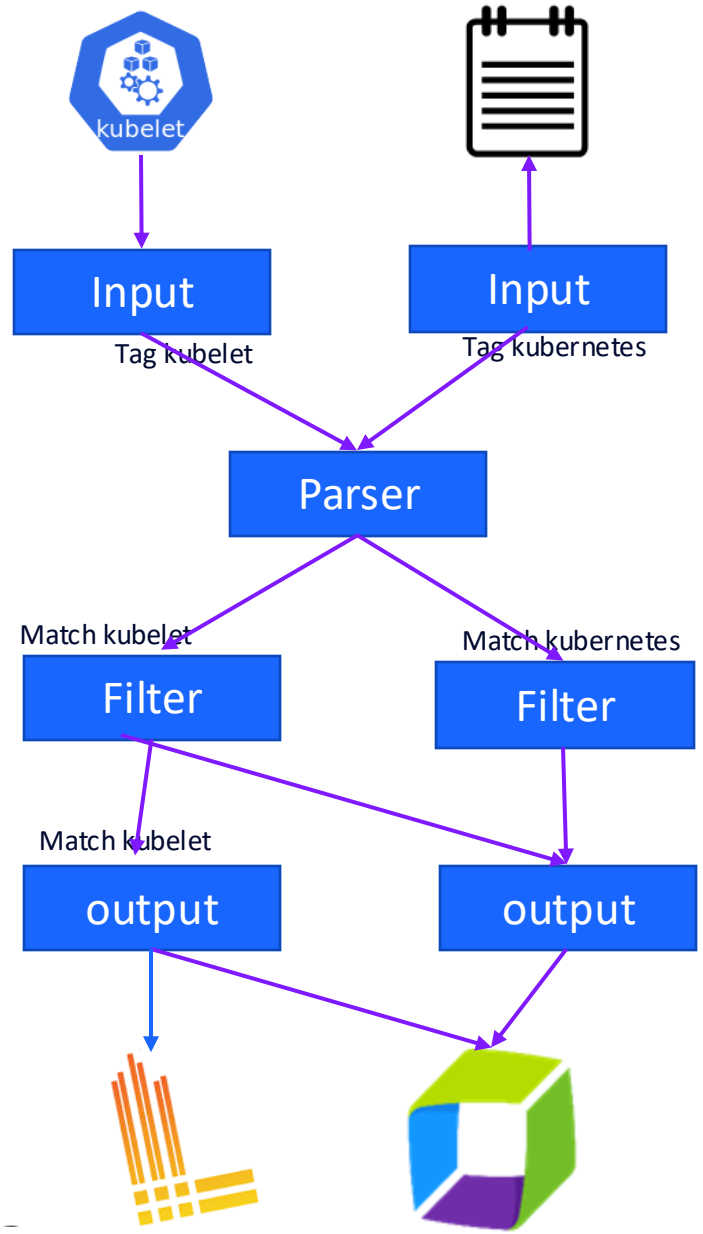


- OpenTelemetry Collector



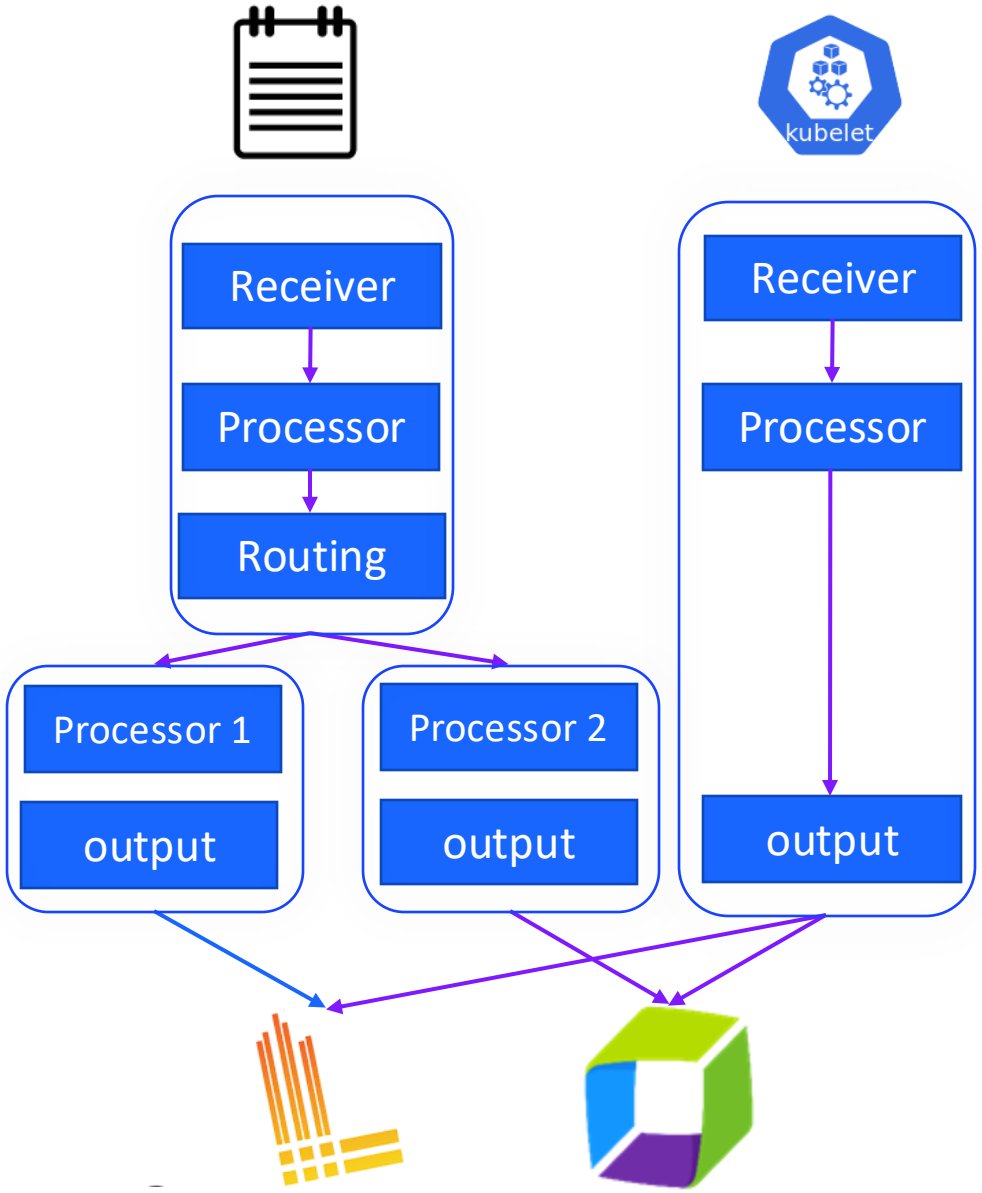
Fluentbit : Tag is key

- In Fluentbit the pipeline flow rely on Tag
- Input plugins will assign a Tag
- Tag will determine witch content will be routed to specific filter, parser, output plugins
- Tag is the key concept to define your pipeline flow



OpenTelemetry Collector: Connector

- The collector provides features helping us to build complex pipeline. The flow would be different based on the content by :
  - Building separate pipelines for specific format
  - Or taking advantage of the routing connector (and probably transform)



Fluentbit Pipeline format

- Before

```
[INPUT]
  Name tail
  Path /var/log/containers/*.log
  Parser docker
  Tag kube.*
  Mem_Buf_Limit 5MB
  Skip_Long_Lines On

[FILTER]
  Name Kubernetes
  Match kube.*
  Merge_Log On
  Merge_Log_Trim On

[FILTER]
  Name modify
  Match kube.*
  Rename log content
  Rename kubernetes_pod_name k8s.pod.name
  Rename kubernetes_namespace_name k8s.namespace.name
  Remove time Remove kubernetes_container_hash
  Add k8s.cluster.name Onlineboutique

[OUTPUT]
  Name http
  Match kube.*
  host <HOST OF THE DYNATRACE ACTIVE GATE>
  port 9999
  Retry_Limit false
```

- After

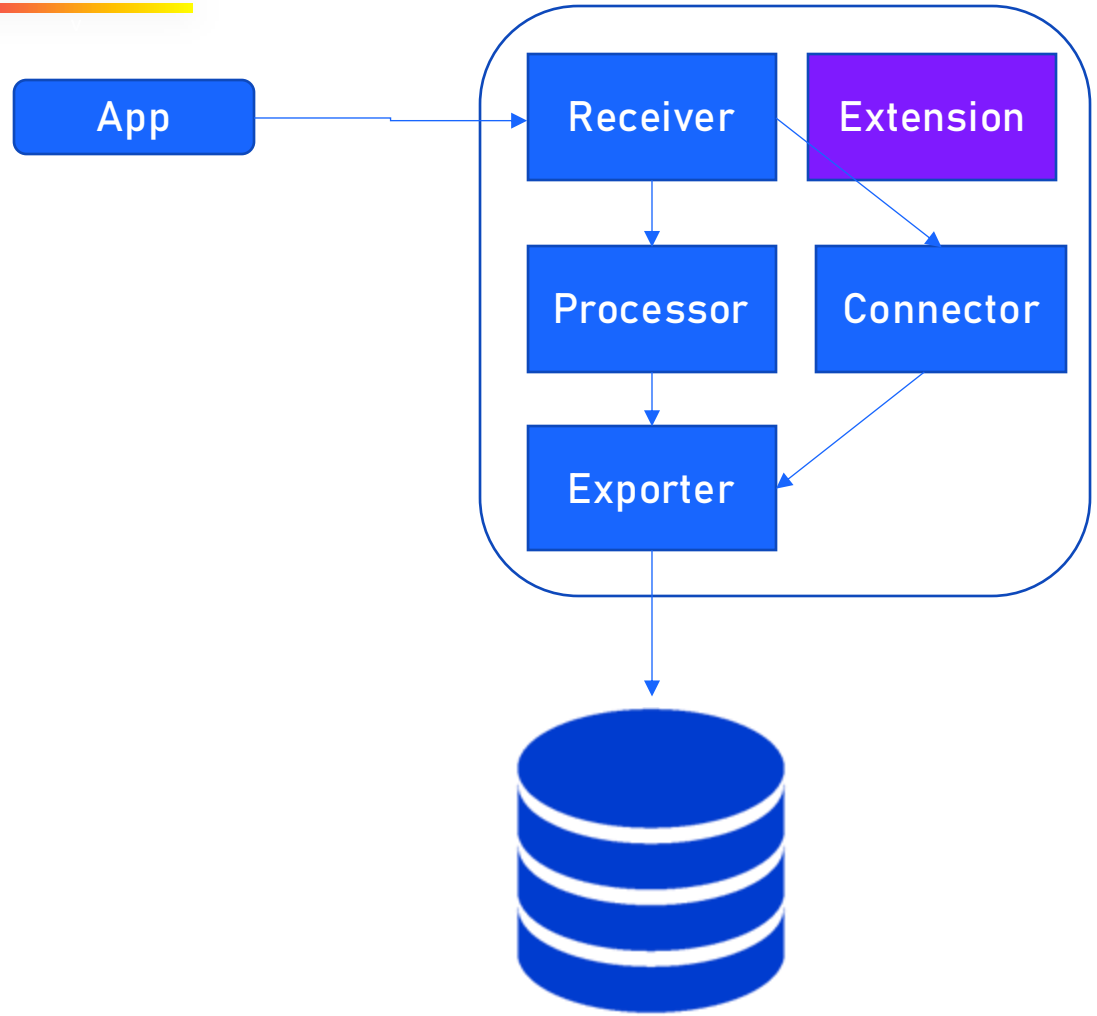
```
pipeline:
  inputs:
    - name: tail
      path: /var/log/containers/*.log
      multiline.parser: docker, cri
      tag: kube.*
      mem_Buf_Limit: 5MB
      skip_Long_Lines: On
      processors:
        logs:
          - name: modify
            match: "kube.*"
            add:
              - k8s.cluster.name ${CLUSTERNAME}
              - dt.kubernetes.cluster.id ${CLUSTER_ID}

  filters:
    - name: kubernetes
      match: kube.*
      merge_log: on
      keep_log: off
      k8s-logging.parser : on
      k8S-logging.exclude: on

    - name: modify
      match: kube.*
      rename:
        - log content
        - kubernetes_pod_name k8s.pod.name
        - kubernetes_namespace_name k8s.namespace.name
      remove:
        - kubernetes_container_image
        - kubernetes_container_hash

  outputs:
    - name: http
      host: ${DT_ENDPOINT_HOST}
      port: 443
      match: “kube.*"
```

Collector Pipeline format



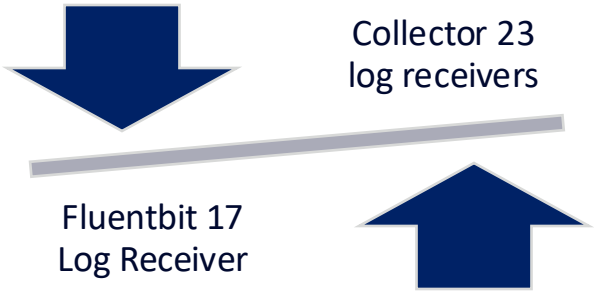

```yaml
receivers:
filelog:
  include:
    - /var/log/pods/*/*/*.log
  start_at: beginning
  include_file_path: true
  include_file_name: false
processors:
 batch:
  send_batch_max_size: 1000
  timeout: 30s
  send_batch_size : 800
 k8sattributes:
  auth_type: "serviceAccount"
  passthrough: false
  filter:
    node_from_env_var: K8S_NODE_NAME
 memory_limiter:
  check_interval: 1s
  limit_percentage: 70
  spike_limit_percentage: 30
exporters:
 otlphttp:
  endpoint: $DT_ENDPOINT/api/v2/otlp
  headers:
    Authorization: "Api-Token $DT_API_TOKEN"
service:
 pipelines:
  logs:
    receivers: [filelog]
    processors: [memory_limiter,k8sattributes,batch]
    exporters: [otlphttp]
 telemetry:
  metrics:
    address: $MY_POD_IP:8888
```

Round 2
Logging

Plugins Required to receive Logs

- To receive logs we need the following plugins :
  - UDP
  - TCP
  - FluentForward
  - OpenTelemetry
  - Syslog
  - Kafka
  - And of course Read from a file

UDP	✓	✓
TCP	✓	✓
Fluent	✓	✓
Syslog	✓	✓
File	✓	✓
OpenTelemetry	✓	✓
Kafka	✓	✓



Plugins Required to Process Logs

- To process logs we need the following plugins :
  - Enrich the logs with:
    - Extra resources
    - Kubernetes metadata
  - Parse the content
  - Drop
  - Batch the logs
  - …etc

Resource	Modify	resource
Kubernetes	✓	✓
Filter	grep	filter
Parse	Regexp, json, parser…etc	Transform (with OTTL)
Batch	Throttle	Batch

Round 3
Metrics

Plugins required to receive metrics

- To receive metrics we mainly need the following plugins :
  - Collectd
  - Statsd
  - OpenTelemetry
  - Prometheus
  - Host Metrics (Windows, Mac, Linux)

Collectd	✓	✓
Statsd	✓	✓
Prometheus	✓	✓
OpenTelemetry	✓	✓
Host	✓	✓

Plugins required to Process metrics

- To process metrics we mainly need the following plugins :
  - Enrich the metric with:
    - Extra resources
    - Kubernetes metadata
  - Drop
  - Convert

Ressource	labels	✓
Transform		✓
Filter	metric_selector	✓
CumulativetoDelta		✓
DeltaToRate		✓
kubernetes		✓

Impossible to convert metrics with Fluentbit

Round 4
Traces

Plugins required to receive traces

- To receive traces we mainly need the following plugins :
  - OpenTelemetry
  - Zipkin
  - OpenCensus
  - Kafka
  - ..Etc

OpenTelemetry	✓	✓
zipkin		✓
OpenCensus		✓
Kafka		✓

Plugins required to Process traces

- To process traces we mainly need the following plugins :
  - Enrich the trace with:
    - Extra resources
    - Kubernetes metadata
  - Manage the sampling decisions
  - Drop

Kubernetes		✓
Ressource	Content_modifier	✓
TailSampling		✓
probabilisticSampling		✓
Filter		✓
Transform		✓

Round 5
Performance

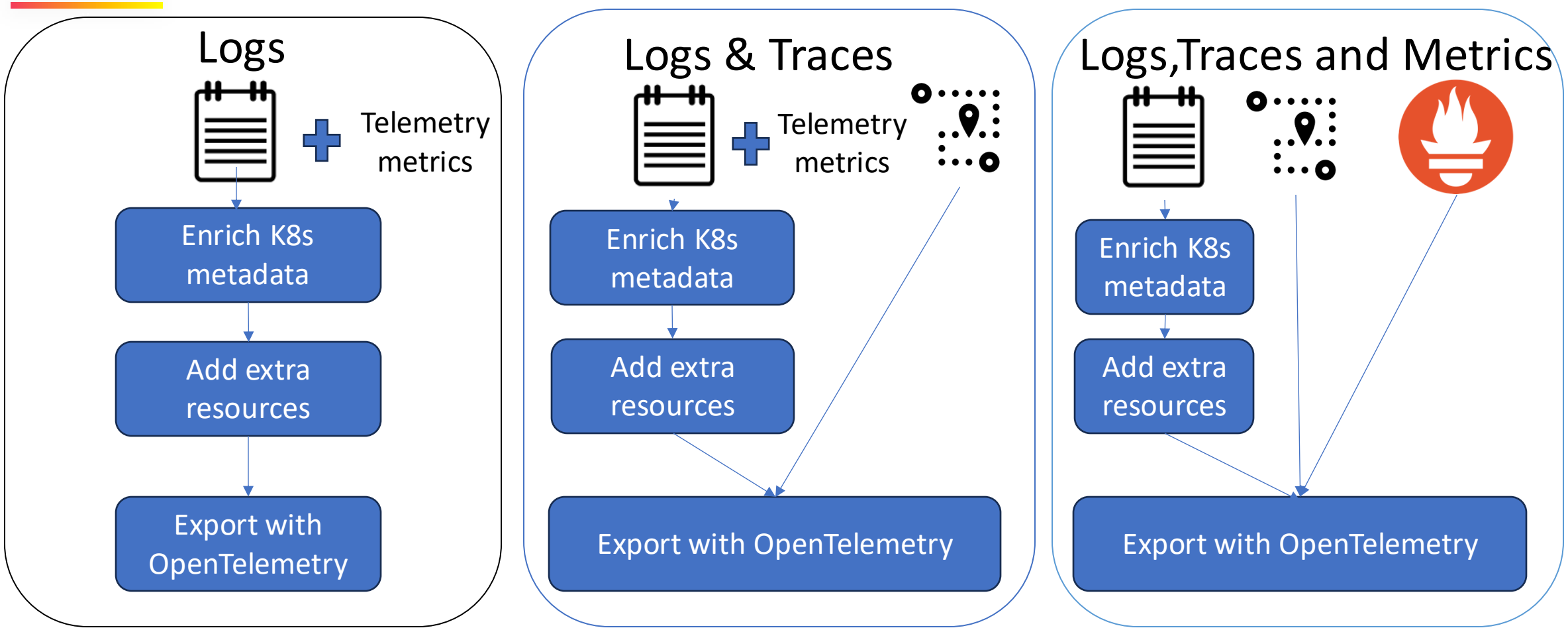
The various tests required to compare

- Fluentbit
  - Collecting logs
  - Collecting logs , traces
  - Collecting Logs, traces and metrics

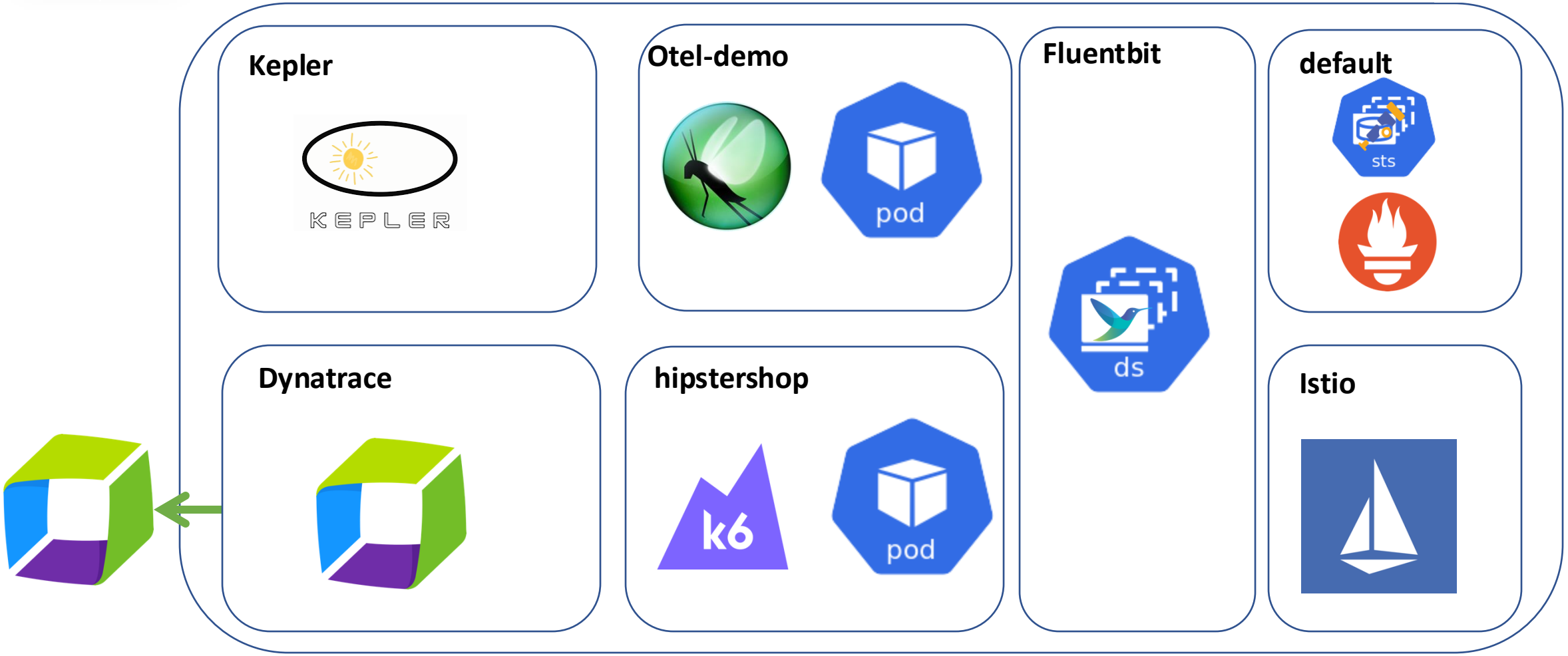
- Collector with processing at the receiver
  - Collecting logs
  - Collecting logs , traces
  - Collecting Logs, traces and metrics

- Collector with processing after receiving
  - Collecting logs
  - Collecting logs , traces
  - Collecting Logs, traces and metrics

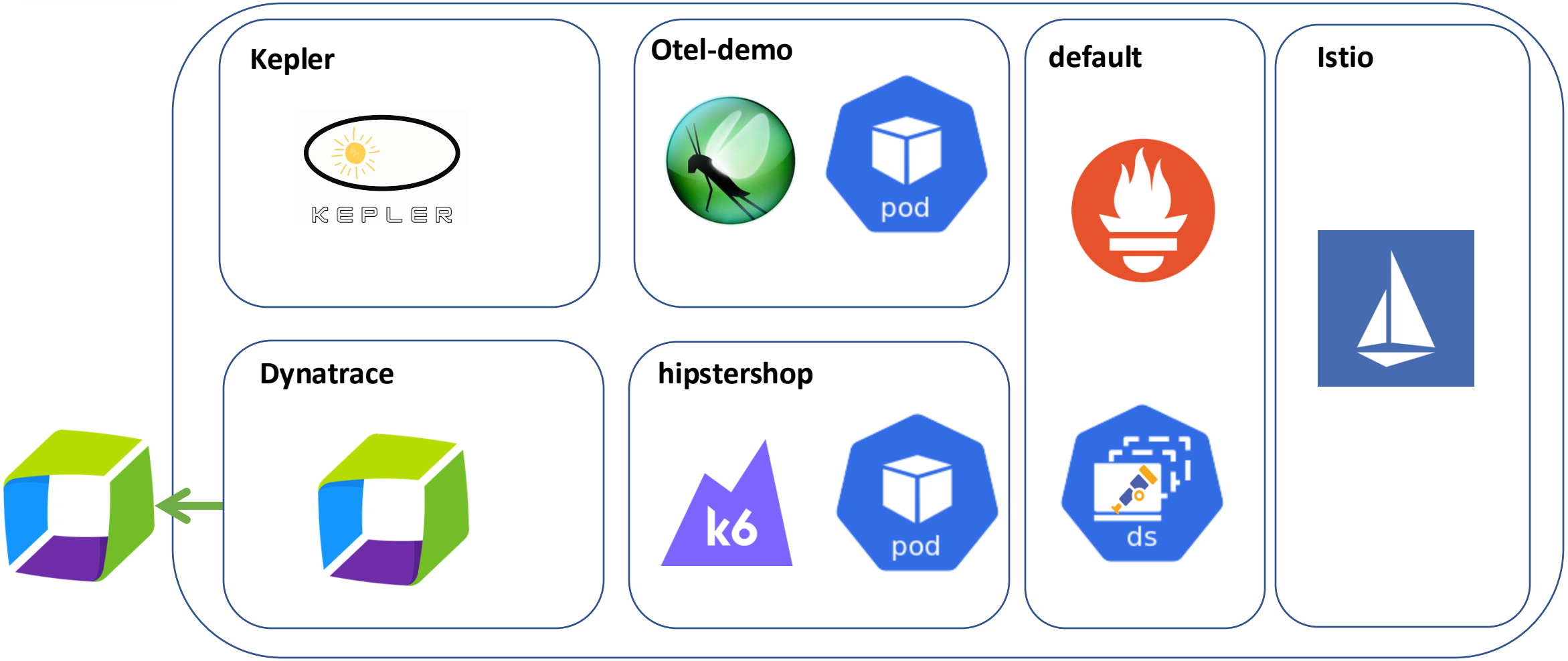
The pipelines



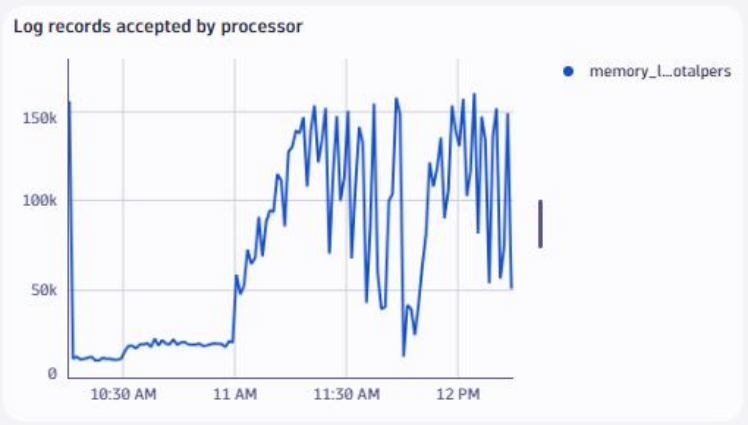
Fluentbit Test Architecture

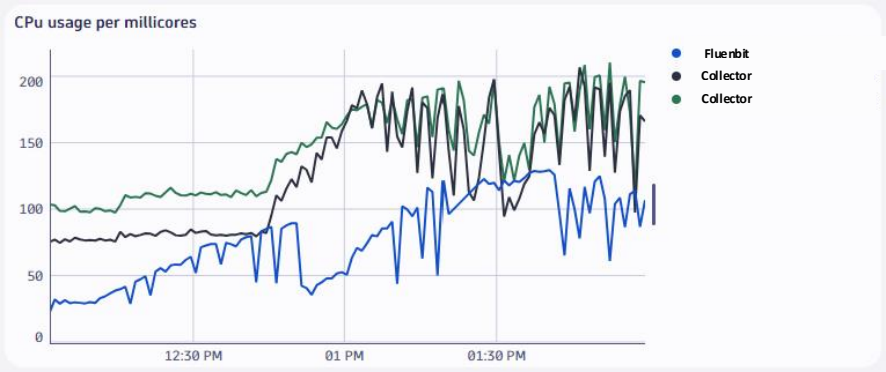


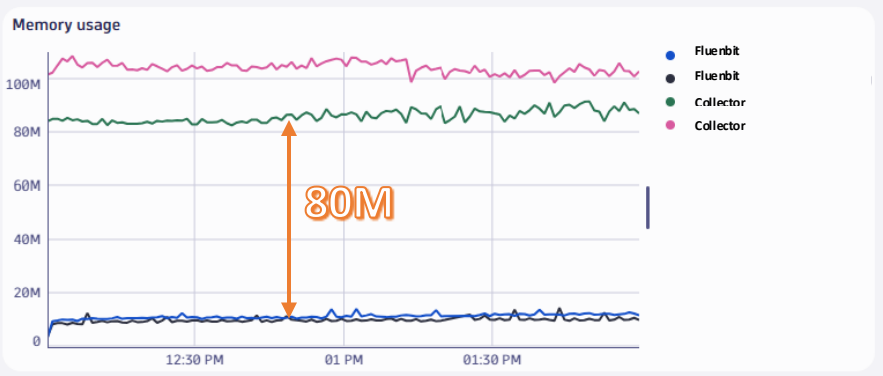
Collector Test Architecture



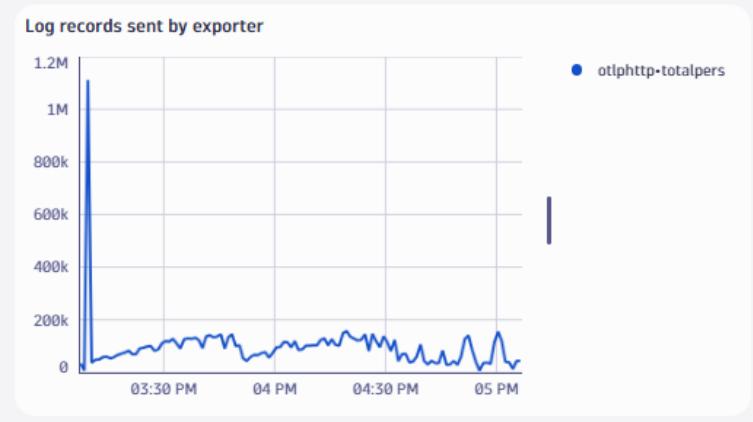
Rampup Tests (2hours) - Logs

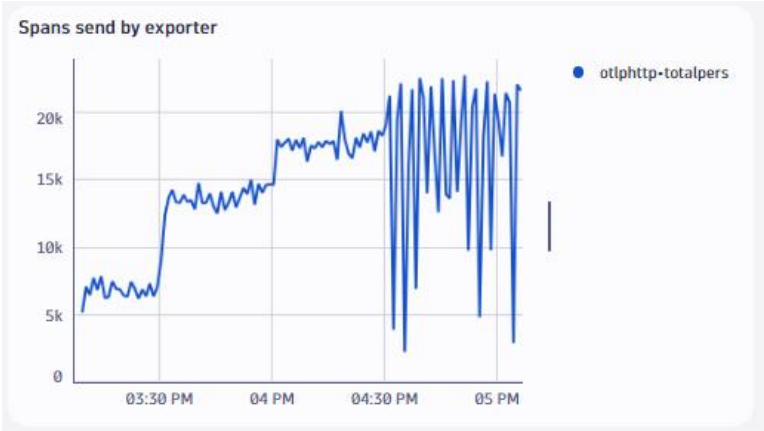


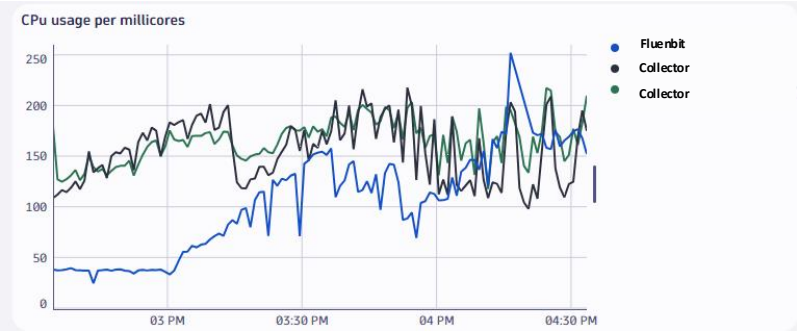


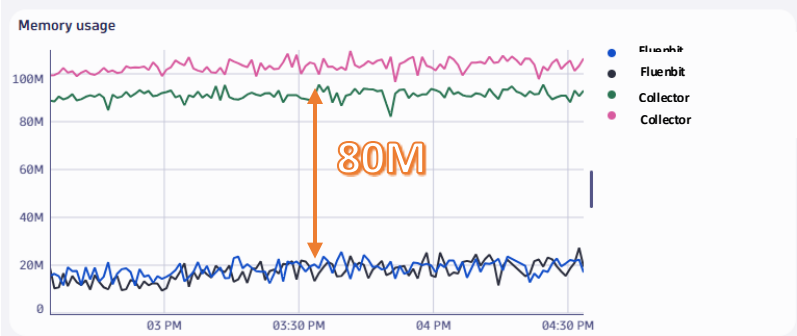


Rampup Tests (2hours) – Logs, Traces

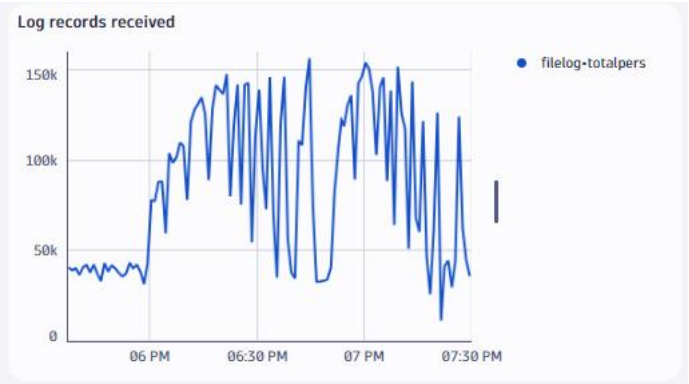


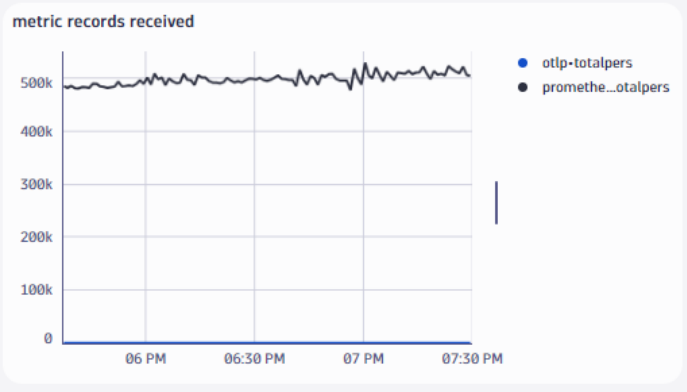


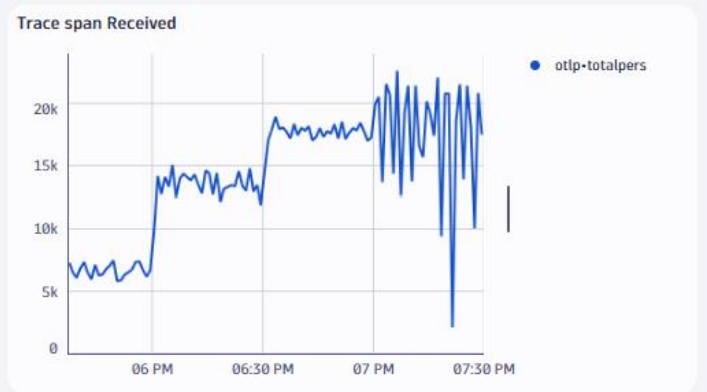


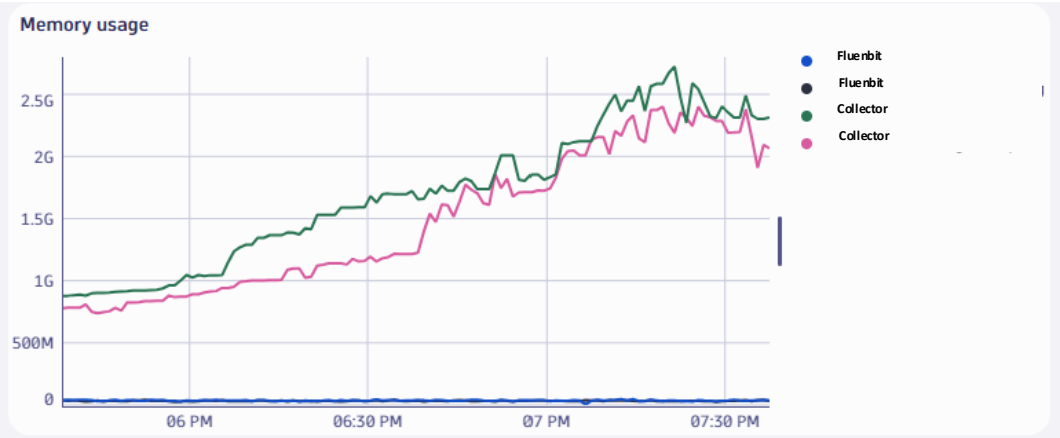


Rampup Tests (2hours) – Logs, Traces & Metrics





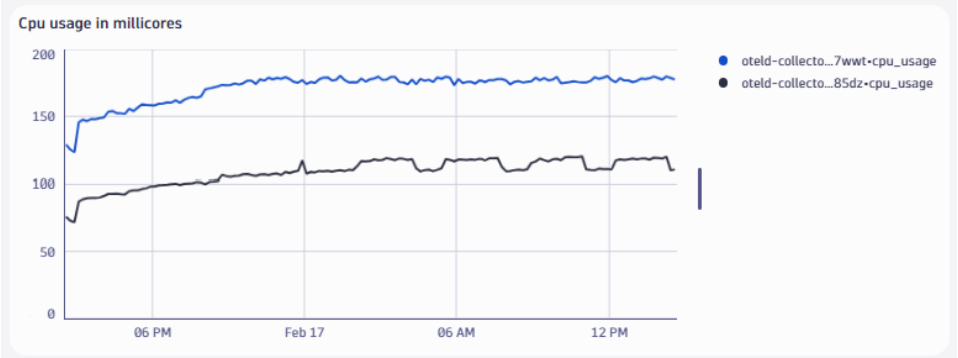


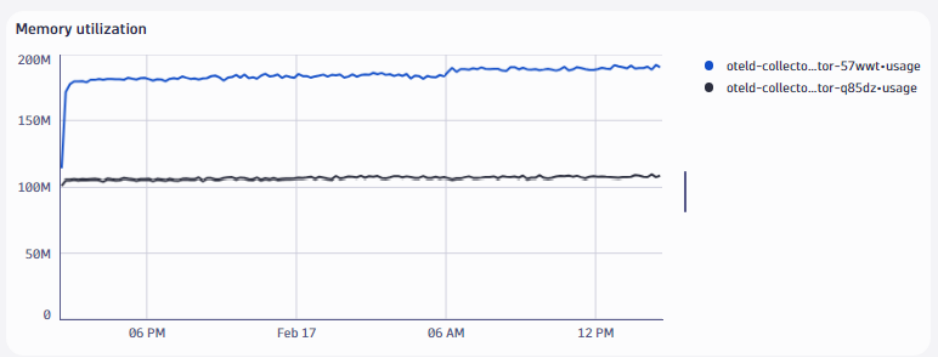


Soak Test (24h) – Logs, Metrics & Traces

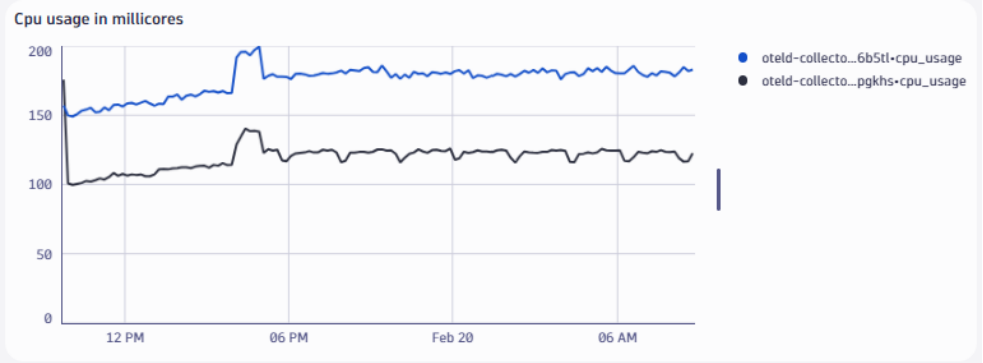
What signal is causing this memory leek

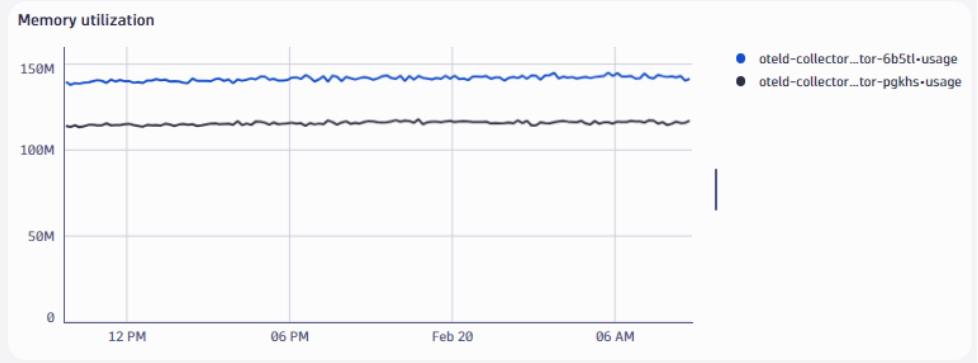
- Collector Soak test with only logs



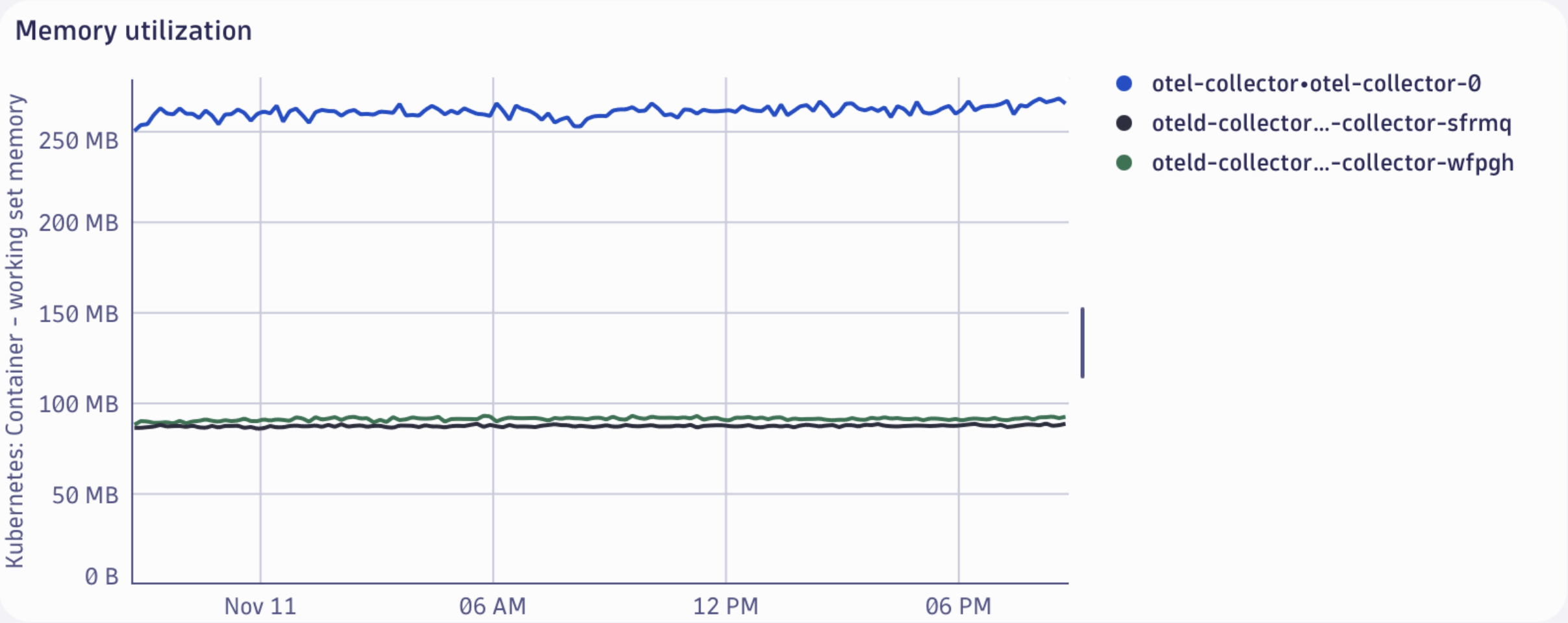


- Collector soak test with logs and traces:





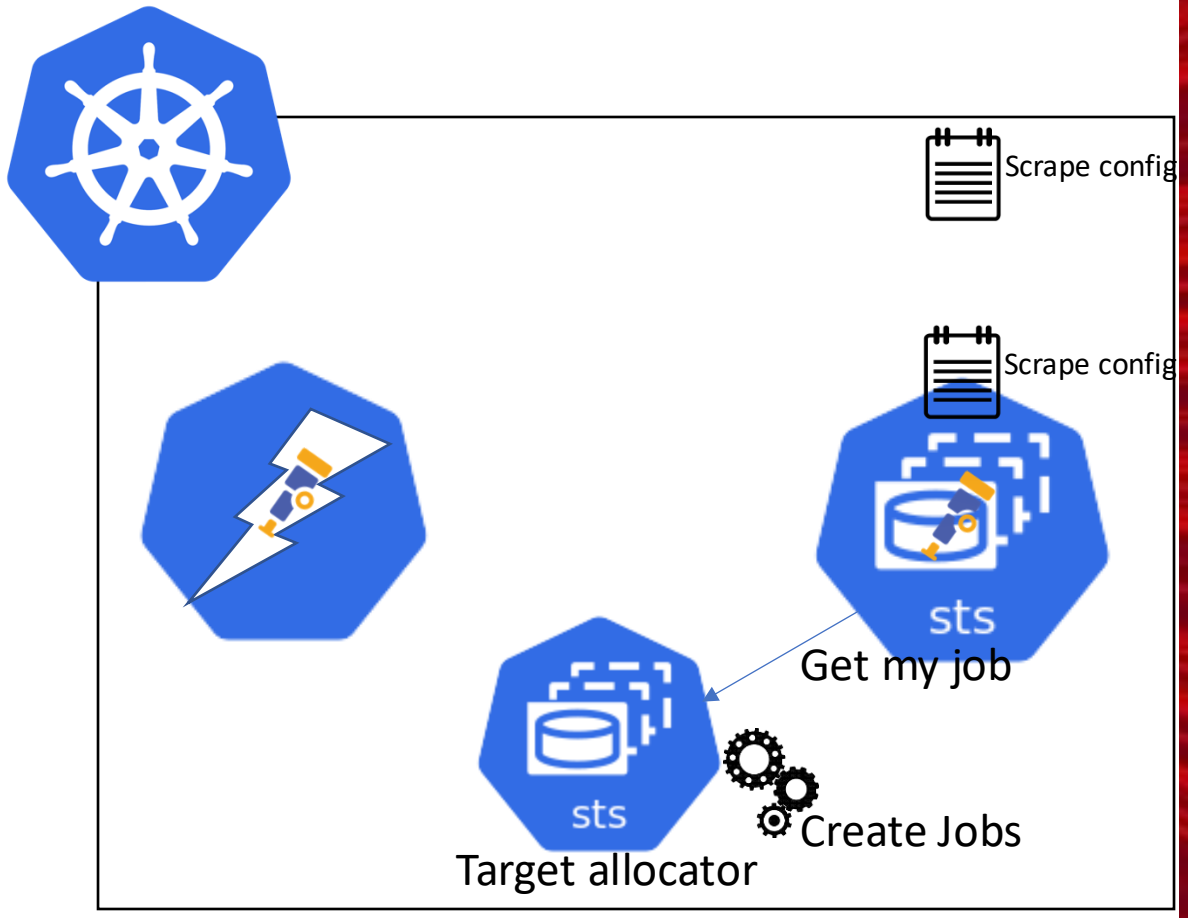
But what is the actual reason?



How can we delegate the metric collection?

The Target Allocator

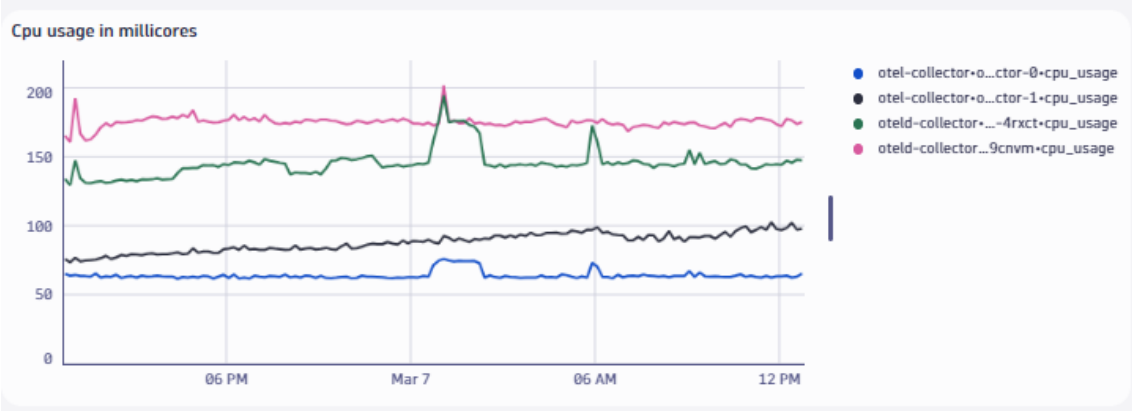
- The OpenTelemtry Operator provides a Target Allocator
- The Target Allocator take the scrape config defined in the pipeline and run it in a separate workload.
- The TargetAllocator create Prometheus Jobs for each collector

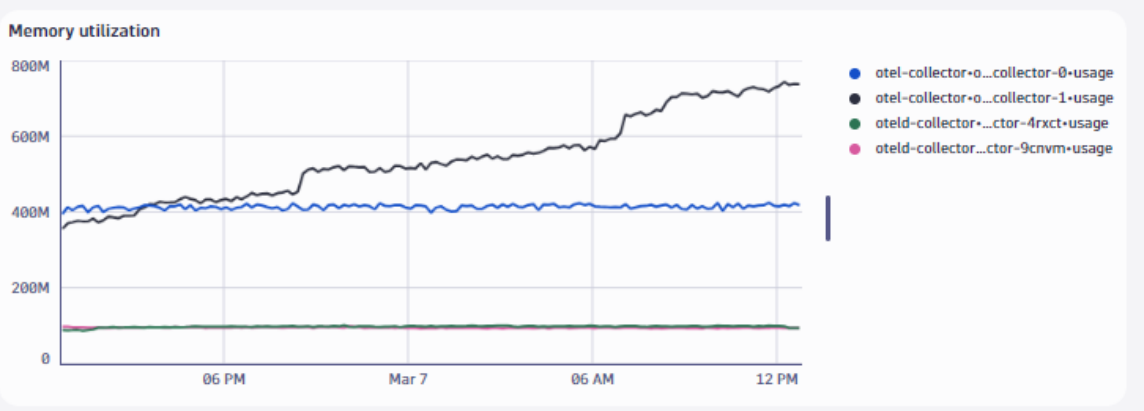


Collector Test Architecture

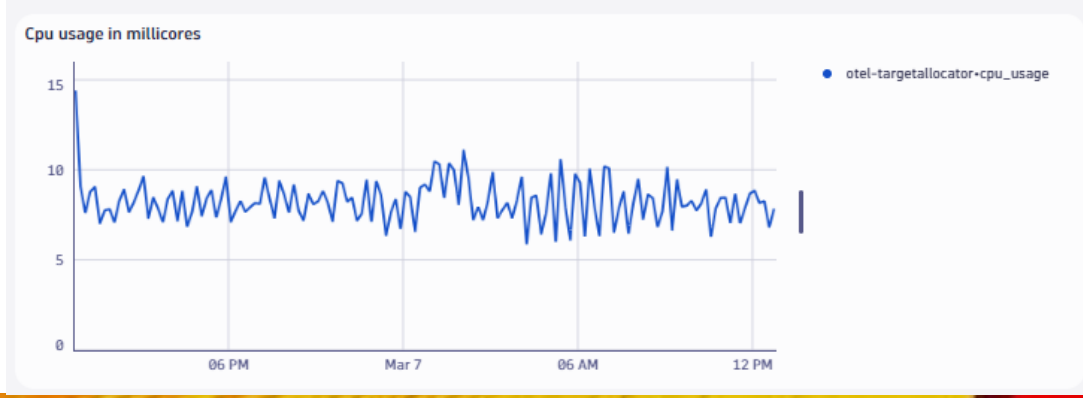
Would the target Allocator reduce the memory usage?

- Collector usage





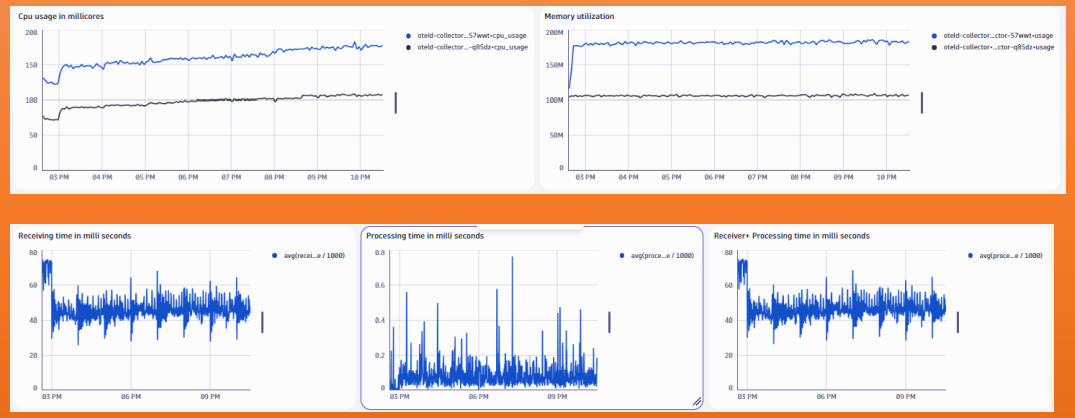
- Target Allocator usage



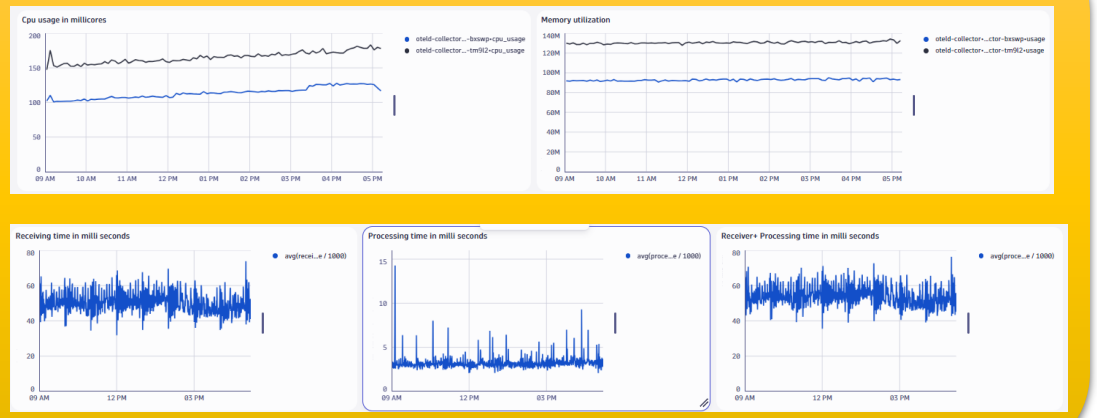
Where should we process our data in the collector?

- When collecting data , it is always recommended to:
  - Filter at the source
  - Transform local details at source
- Let’s compare the behavior of processing at the receiver vs doing it with the transform processor:

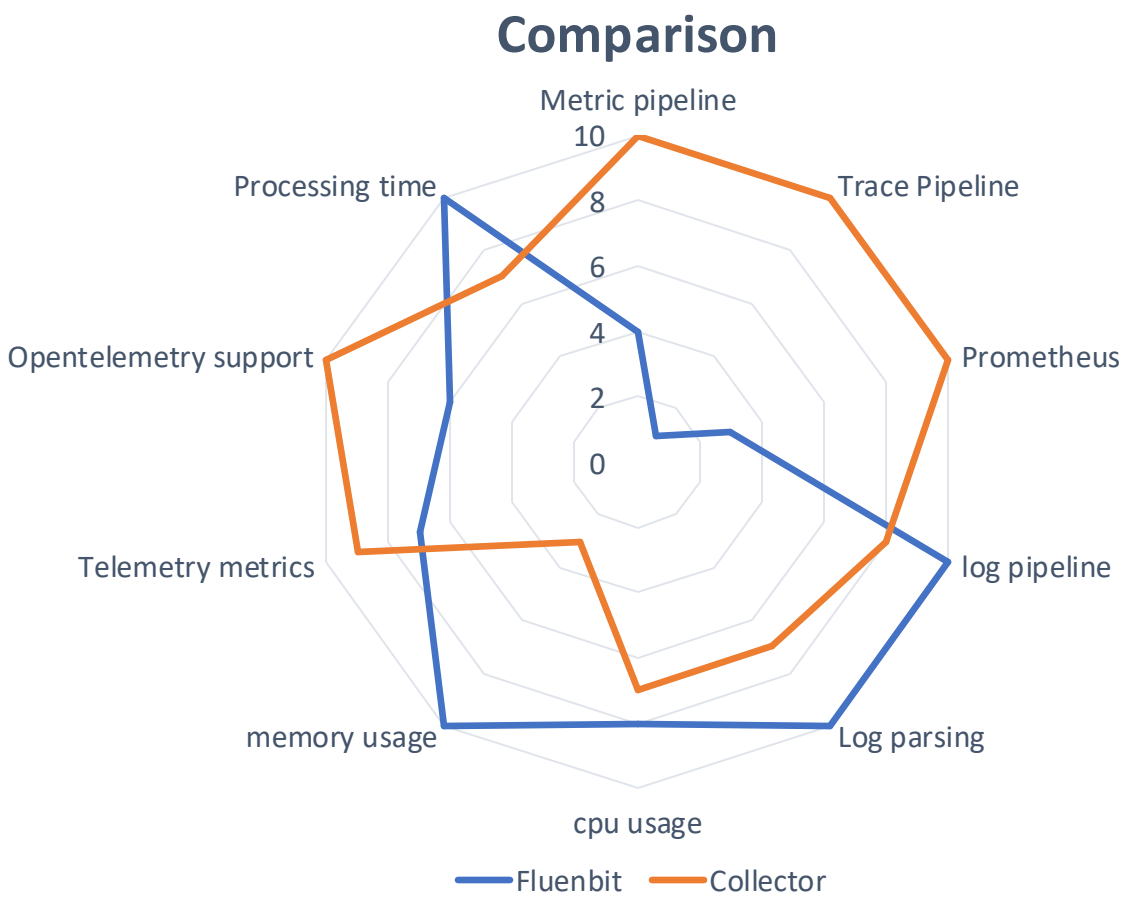
Filelog



Transform



Conclusion


Comparison
Metric pipeline
Trace Pipeline
Prometheus
log pipeline
Log parsing
cpu usage
memory usage
Telemetry metrics
Opentelemetry support
Processing time
10
8
6
4
2
0
Fluenbit
Collector

Is it observable

- Looking for educational content on Observability , Checkout the YouTube Channel :

Is It Observable

Is it Observable ?

dynatrace