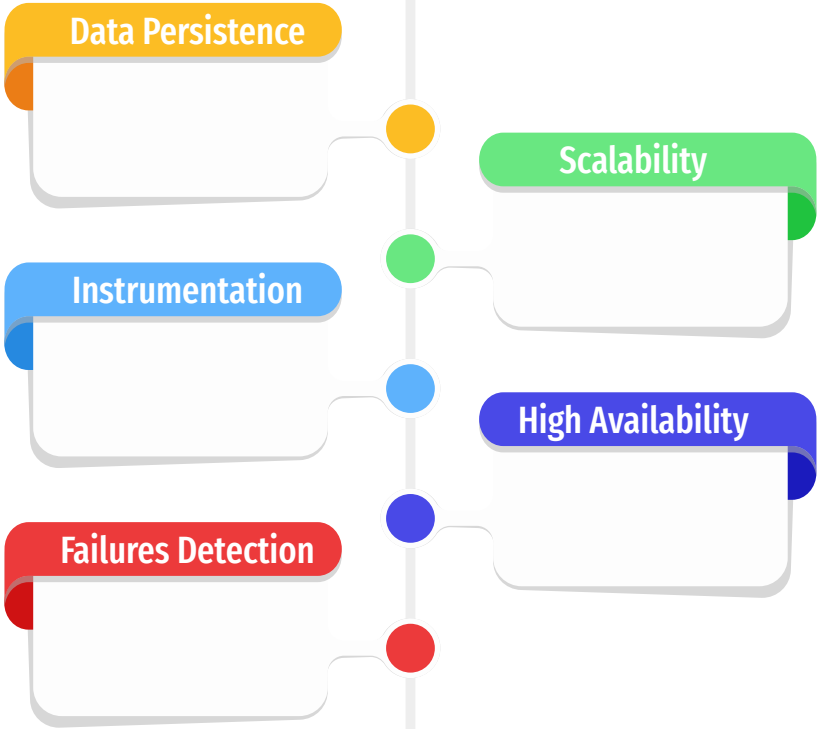
Building Resilient Observability Pipelines

with OpenTelemetry Collector



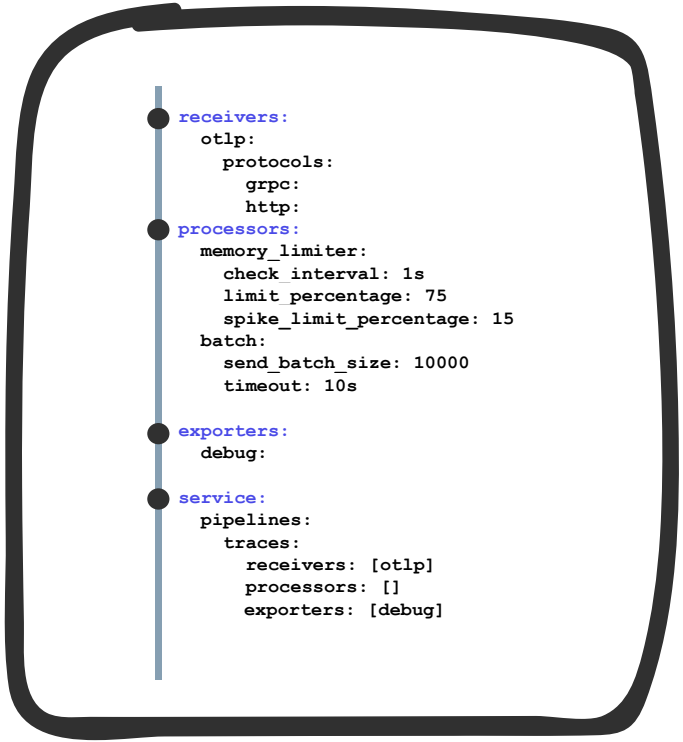
Yuri Oliveira Sá, Red Hat
Juraci Paixão Kröhling, Grafana Labs

What's Resilience in Observability?

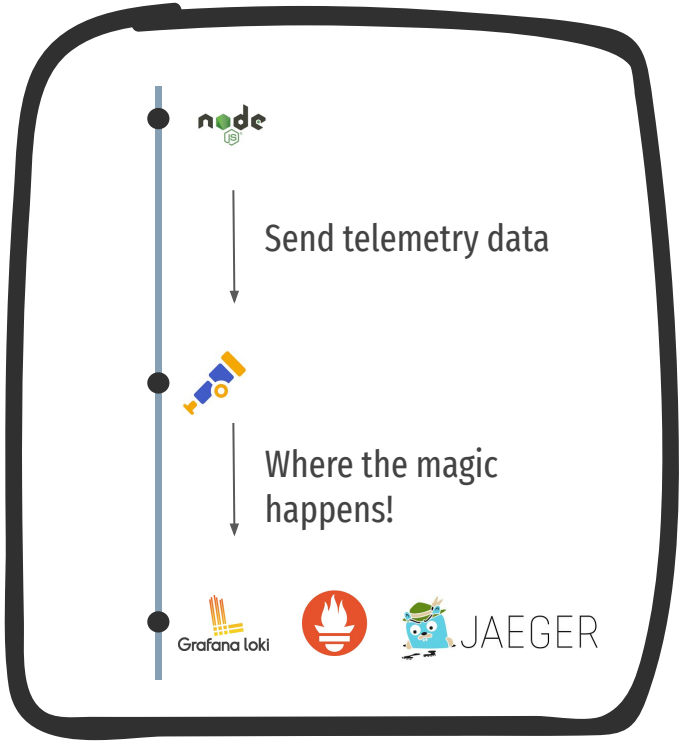
" ...is the ability of an application to resist or recover from certain types of faults or load spikes, and remain functional from the customer perspective"

Source: https://community.aws/concepts/building-an-observability-strategy-for-resilience

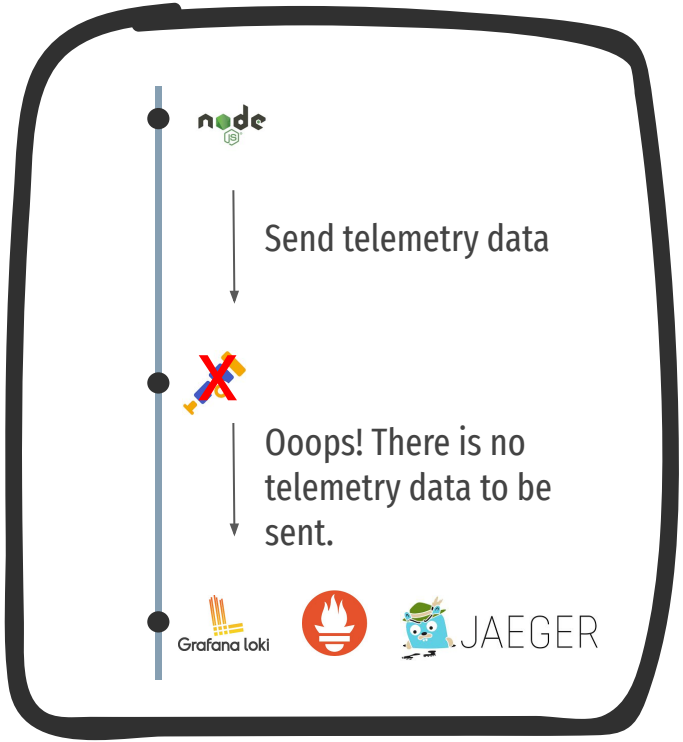
What does an OpenTelemetry Pipeline look like?



Which means...


node
Send telemetry data
Where the magic happens!
Grafana loki
JAEGER

What if OpenTelemetry Collector goes down?

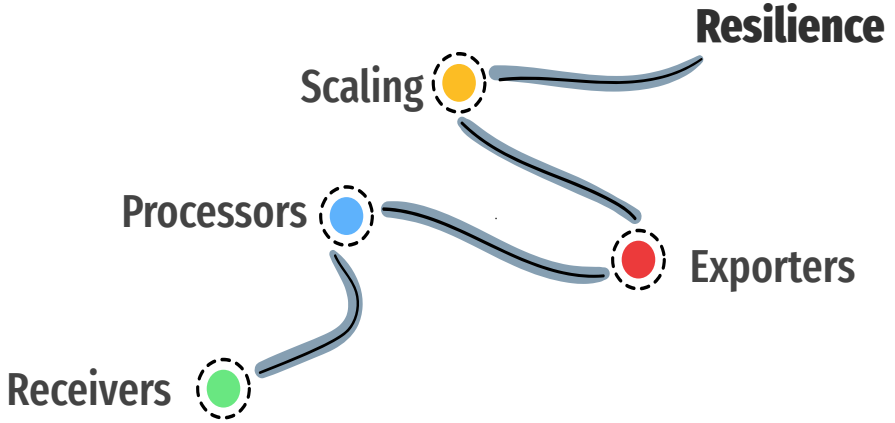



TH—
THIS IS NOT FINE!!

Inflection Point

How long can OTel Collector be unavailable?

OpenTelemetry resources to help us on that journey.



The Big Bang was not the beginning, but Receivers!

- **Channels:** gRPC and HTTP.
- **gRPC - max_connection_idle** : is a duration for the maximum amount of time a connection may exist before it will be closed by sending a Go Away. A random jitter of +/-10% will be added to MaxConnectionAge to spread out connection storms.

#1

The Big Bang was not the beginning, but Receivers!

- **Compression type:** use among gzip, snappy, zstd, and none.
- **read_buffer_size:** lets you set the size of read buffer, this determines how much data can be read at most for one read syscall.
- **write_buffer_size:** determines how much data can be batched before doing a write on the wire.

#2

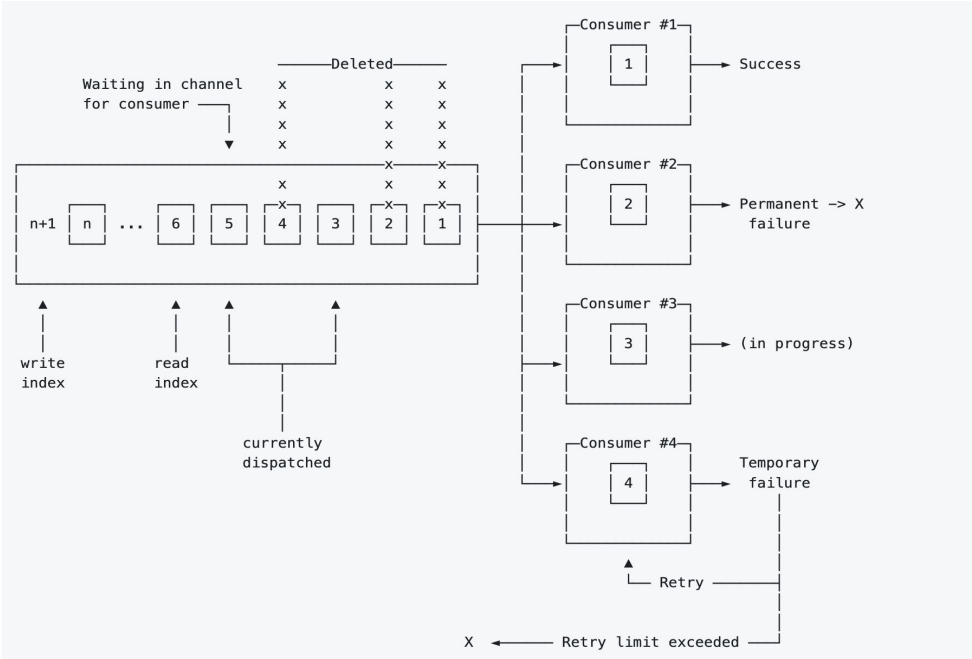
Processors in the pipeline. What's that?

- **Memory Limiter**: The memory limiter processor is used to prevent out of memory situations on the collector, performing periodic checks of memory usage and starting to refuse data and forcing GC to reduce memory consumption when defined limits have been exceeded.
- **Batch processor:** The batch processor helps better compress the data and reduce the number of outgoing connections required to transmit the data, and should be defined in the pipeline after the `memory_limiter` as well as any sampling processors.

Exporter Ensures Data Persistency.

- **Persistent Queue:** If the collector instance is killed while having some items in the persistent queue, on restart the items will be picked and the exporting is continued.
- **Queue Size:** Maximum number of batches kept in memory before dropping; User should calculate this as:
  `seconds * batches_to_survive`

Persistent Queue

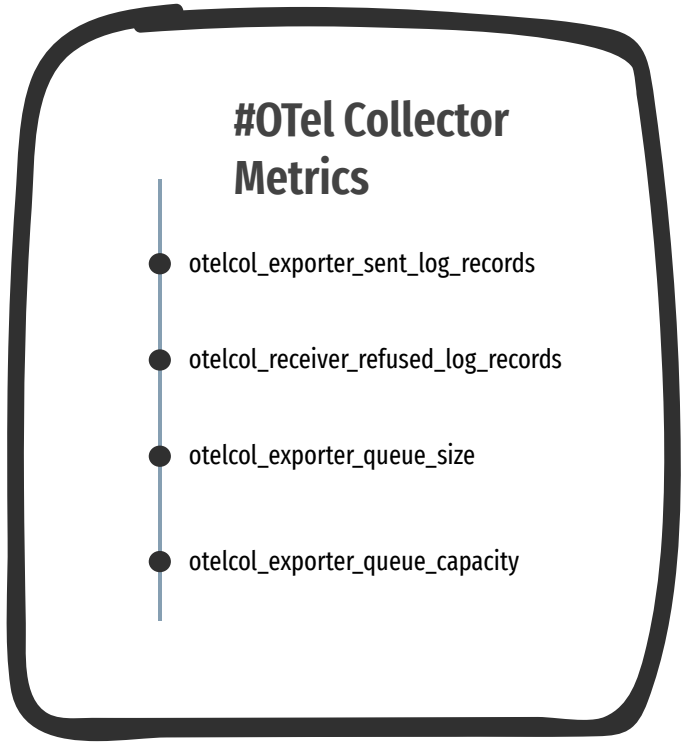


Source:
https://github.com/open-telemetry/opentelemetry-collector/tree/main/exporter/exporterhelper

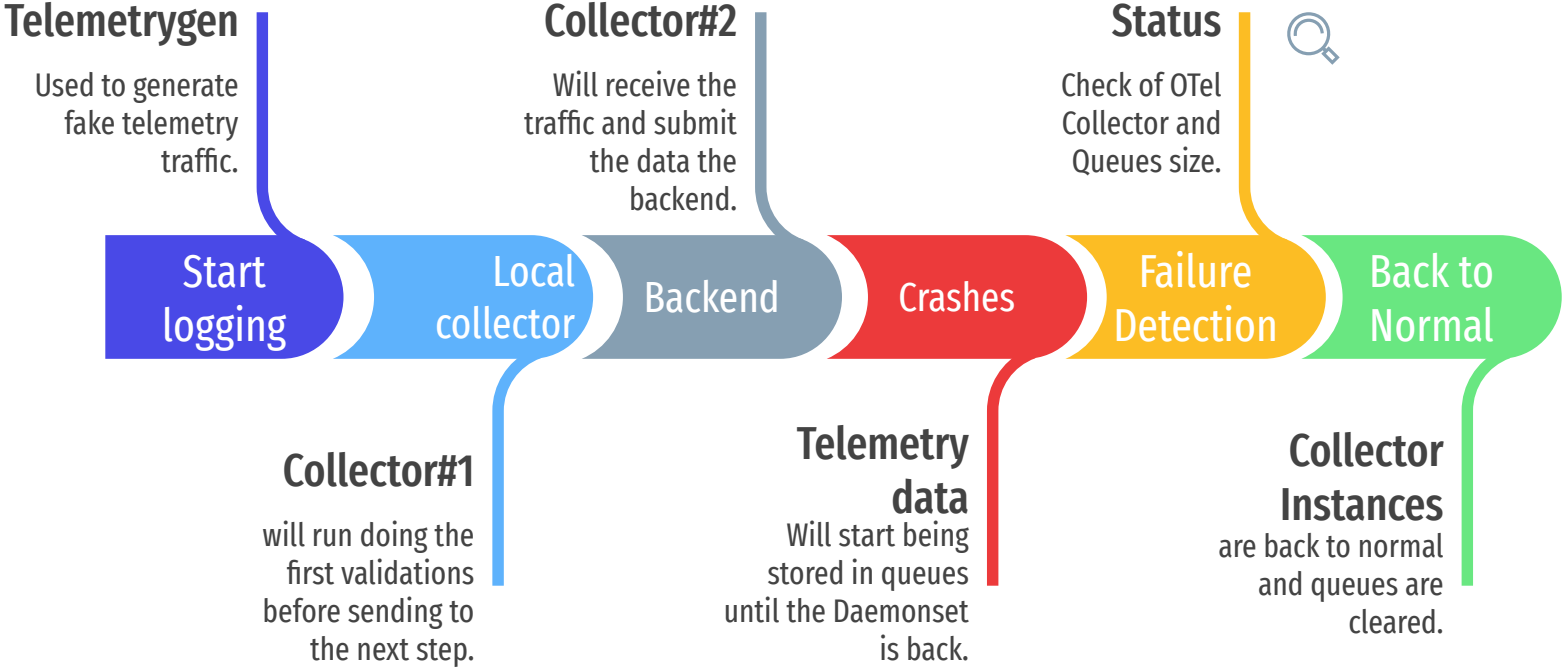
Connection tweaks

- configgrpc tweaks for better client-side load-balancing
- confighttp tweaks for better batching

How to detect failures on OpenTelemetry Collector?



Outage Simulation



It's too good to be true...

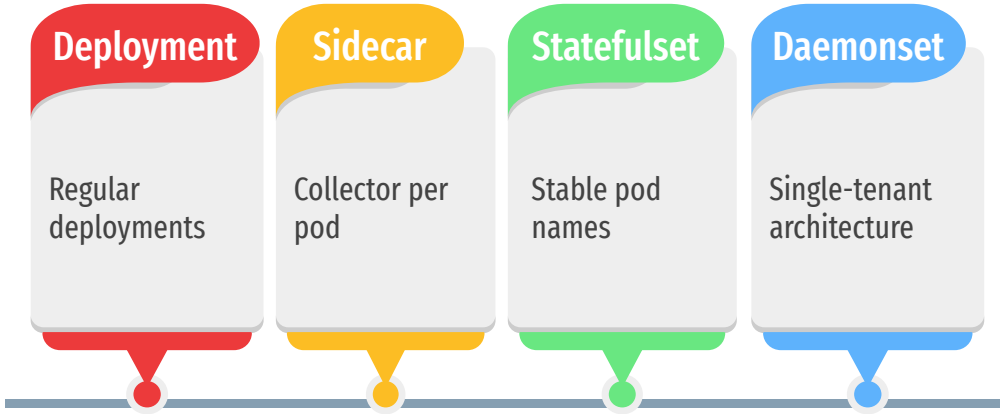
Last, but not least:

Scaling and Balancing

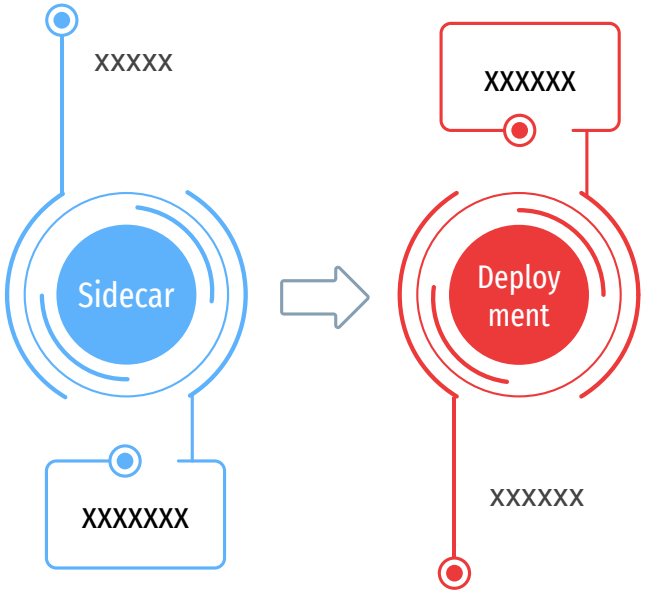
#Strategies

- **Via OpenTelemetry Operator:** HPA.
- **Load-balancing:**
  - Off-the-shelf
  - Exporter
- **Deployment Architecture:**
  - Sidecar to Deployment.

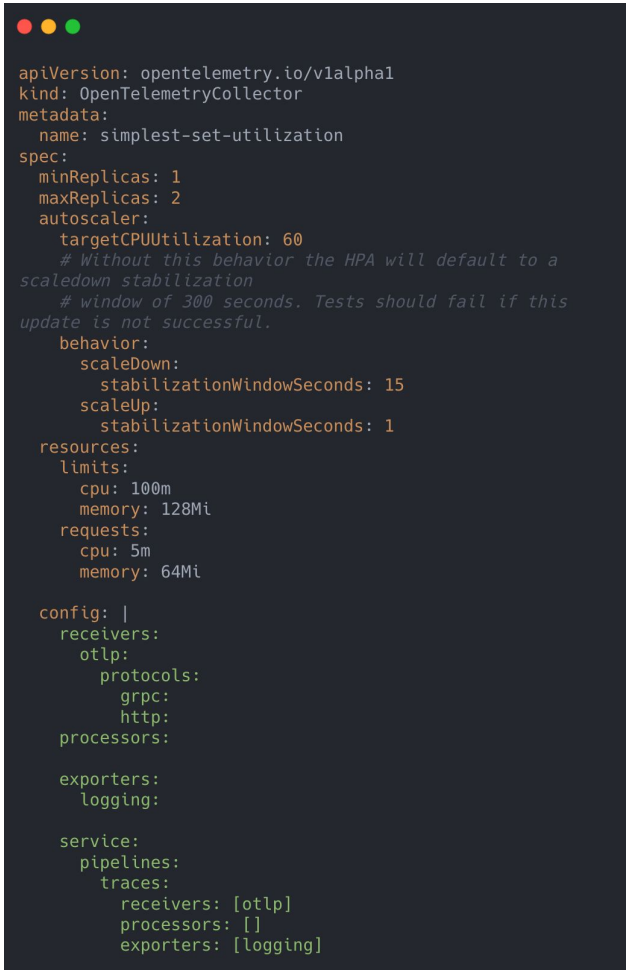
OpenTelemetry Collector Deployment Modes on Kubernetes.



A good approach: Sidecar To Deployment.



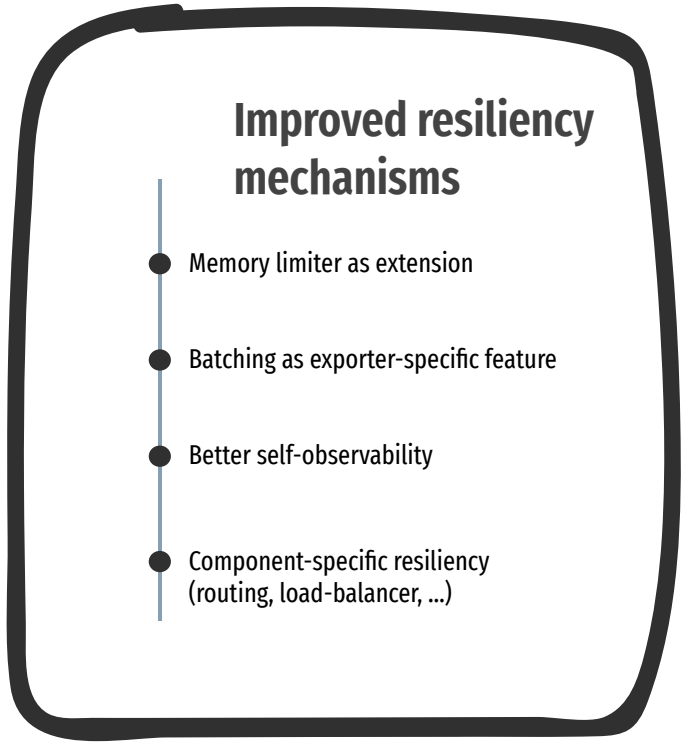
Native HPA config using Operator.



```yaml
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: simplest-set-utilization
spec:
  minReplicas: 1
  maxReplicas: 2
  autoscaler:
    targetCPUUtilization: 60
    # Without this behavior the HPA will default to a
scaledown stabilization
    # window of 300 seconds. Tests should fail if this
update is not successful.
    behavior:
      scaleDown:
        stabilizationWindowSeconds: 15
      scaleUp:
        stabilizationWindowSeconds: 1
  resources:
    limits:
      cpu: 100m
      memory: 128Mi
    requests:
      cpu: 5m
      memory: 64Mi

  config: |
    receivers:
      otlp:
        protocols:
          grpc:
          http:
    processors:

    exporters:
      logging:

    service:
      pipelines:
        traces:
          receivers: [otlp]
          processors: []
          exporters: [logging]
```


Show me
the data!

What's next?


Improved resiliency mechanisms
Memory limiter as extension
Batching as exporter-specific feature
Better self-observability
Component-specific resiliency (routing, load-balancer, ...)

Takeaways

- **Not all** pipelines **HAVE** to be resilient.
- **Resiliency** is a concern for the whole pipeline and for the deployment architecture!
- The **Collector** has internal mechanisms allowing it to survive backend crashes.
- Take a look at the available **metrics**: they tell the story of your pipeline.
- **In-flight** data can also be replayed under specific conditions.
- Different **scaling and load-balancing** techniques are available to achieve HA.

Thank you, Merci, Obrigado!

Yuri Oliveira Sa

 /yurioliveirasa

 /yuriolisa

 /yurioliveirasa

Juraci Paixão Kröhling

 /jpkroehling

 /jpkrohling

 /jpkrohling