

SE Training – Kubernetes OpenTelemetry



Training Recorded

Warning This training is being recorded and will be available for future on-demand consumption. Please maintain professionalism at all times. Thank you.

Kubernetes OpenTelemetry Agenda

- **Goal:** Capture logs, traces, and metrics from Kubernetes using OpenTelemetry and ship them to Dynatrace for analysis.
Kubernetes Observability without any Dynatrace native components installed on the Kubernetes cluster
- **What is OpenTelemetry?**
- **OpenTelemetry with Dynatrace**
 - Dynatrace Distro of OpenTelemetry Collector
- **Hands On**
 - OpenTelemetry for Logs
 - OpenTelemetry for Traces
 - OpenTelemetry for Metrics
 - OpenTelemetry Capstone – putting it all together
- **Knowledge Assessment (Quiz)**

Codespaces Cluster Set Up

But first, let's spin up our Codespaces Kubernetes (Kind) Cluster!

Guide exercises



Prerequisites

5 exercises · Not started

- Prerequisites
- Codespaces Cluster Set Up
- Generate Dynatrace Access Token
- Environment Prep
- Deploy OpenTelemetry Operator

Prerequisites

During this hands-on training, we'll learn how to capture logs, traces, and metrics from Kubernetes using OpenTelemetry and ship them to Dynatrace for analysis. This will demonstrate how to use Dynatrace with OpenTelemetry; without any Dynatrace native components installed on the Kubernetes cluster (Operator, OneAgent, ActiveGate, etc.).

Training Prerequisites

- Codespaces Cluster Set Up
- Generate Dynatrace Access Token
- Environment Prep
- Deploy OpenTelemetry Operator

OpenTelemetry by the numbers



HISTORY

- Announced at KubeCon in 2019
- Next gen of Open Tracing and Open Census
- Traces ship in 2021, Metrics ship in 2022



POPULARITY

- #2 CNCF Project behind K8s (2k followers)
- 17% enterprise usage *
- #1 open observability standard



MATURITY

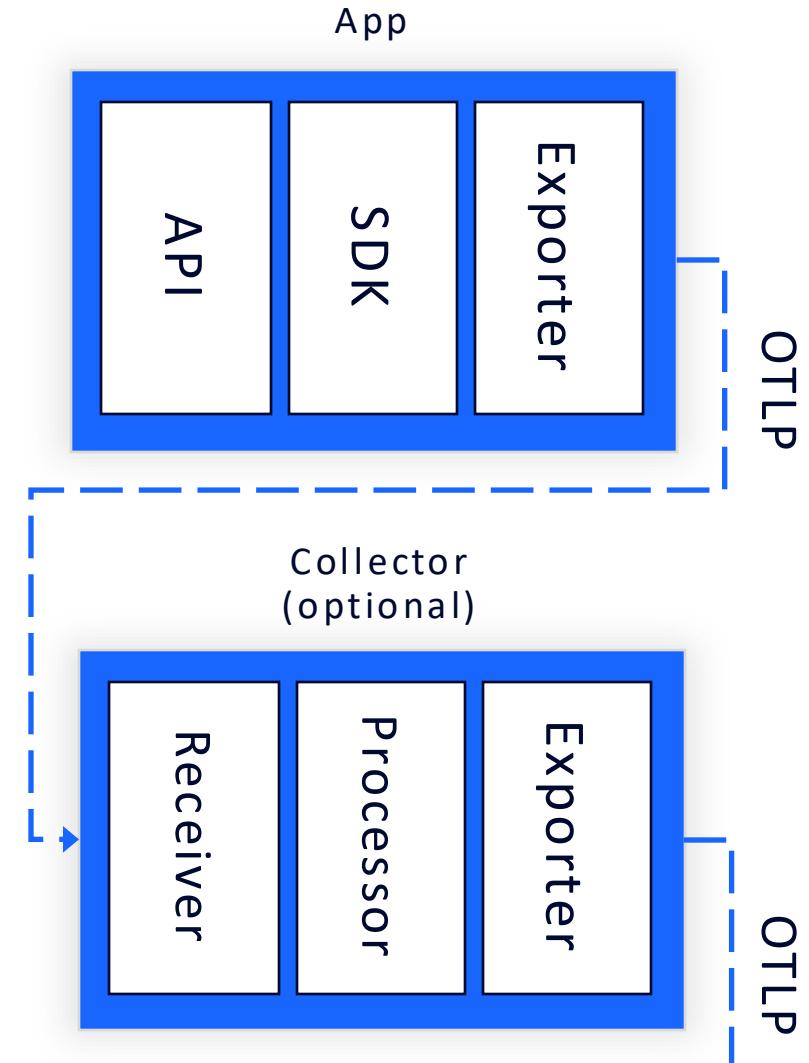
- Stable tracing spec
- Mixed spec for metrics and logs
- Mixed implementation for languages and technologies

* OpenLogic State of Open-Source Survey 2023



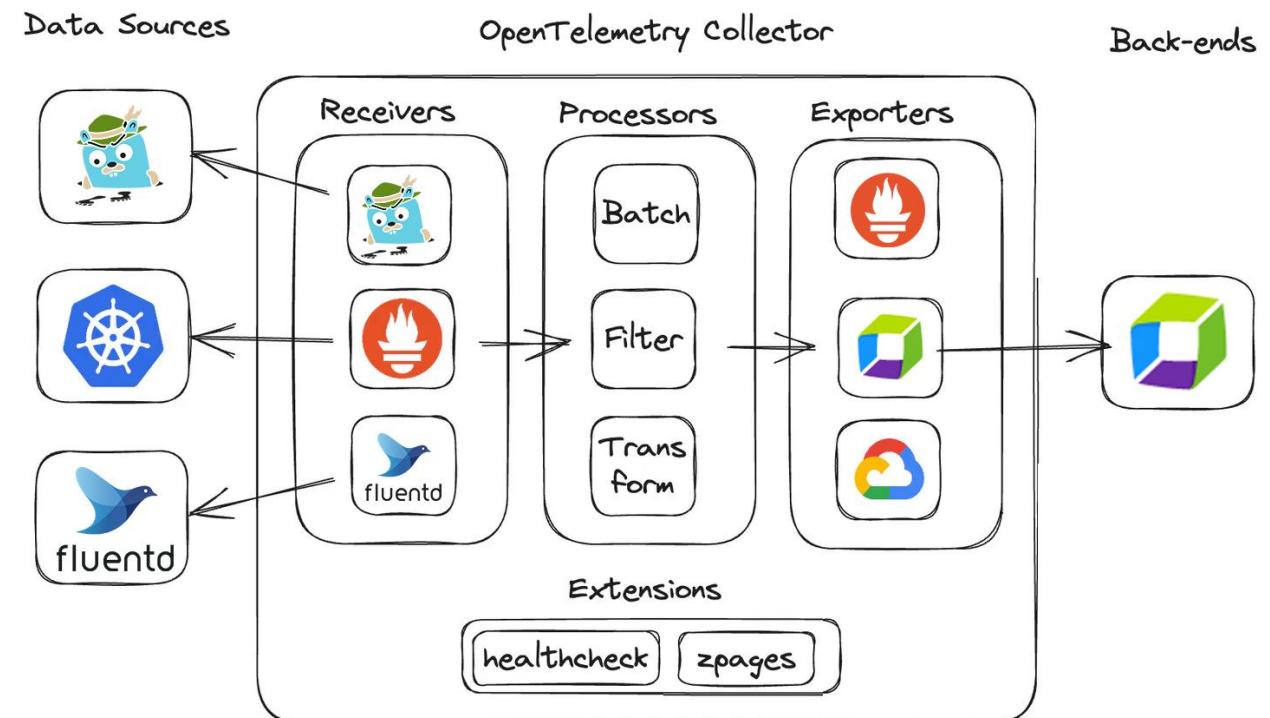
What is OpenTelemetry?

- An observability framework for cloud-native software
- In Practice
 - APIs and SDKs for apps to capture telemetry data
 - Agents to help avoid manual code changes
 - Exporters that send data to different observability platforms
 - Collectors to receive, process, and export
 - Semantic Conventions for standardized naming and tagging
 - Projects and working groups to extend functionality

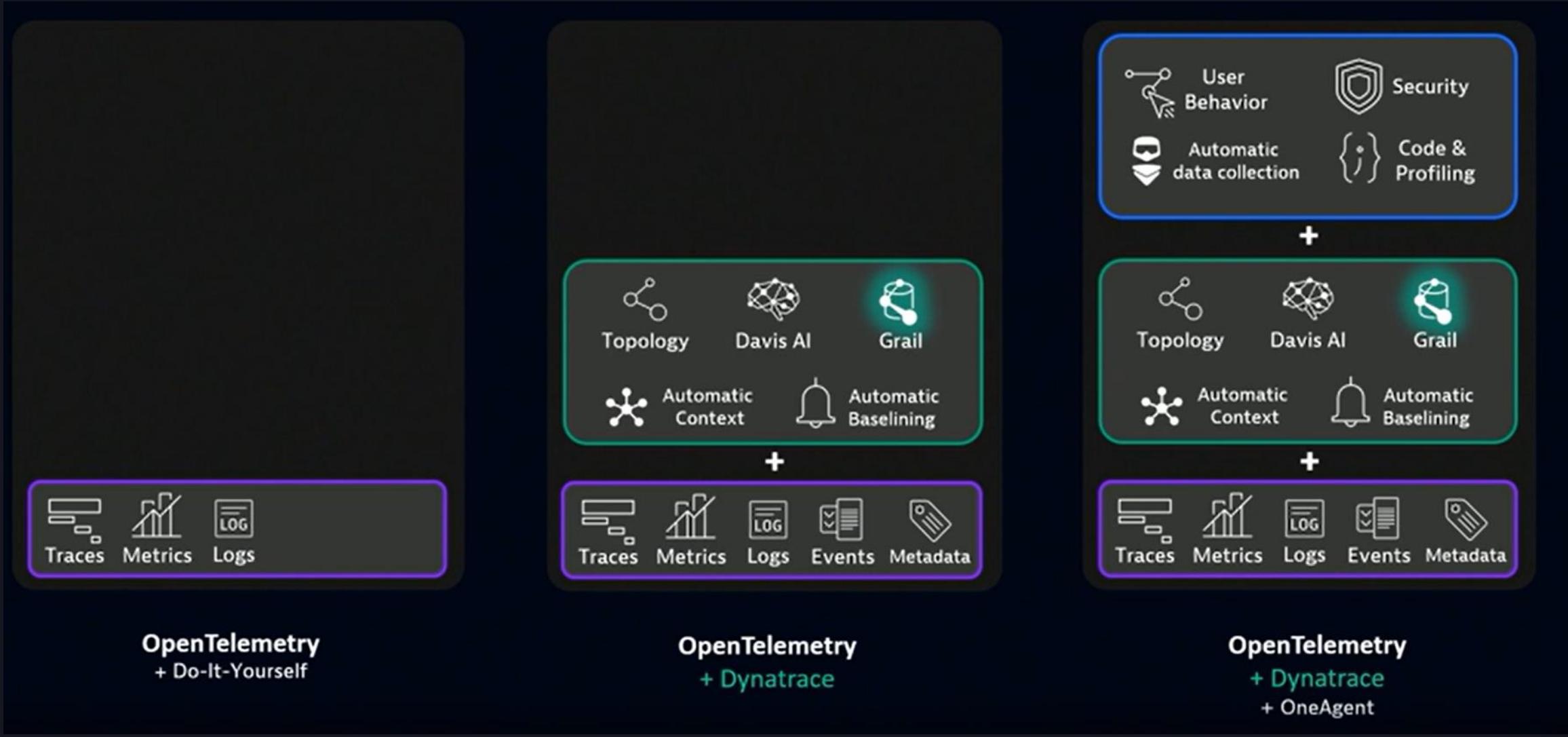


OpenTelemetry: Receivers, Processors, and Exporters

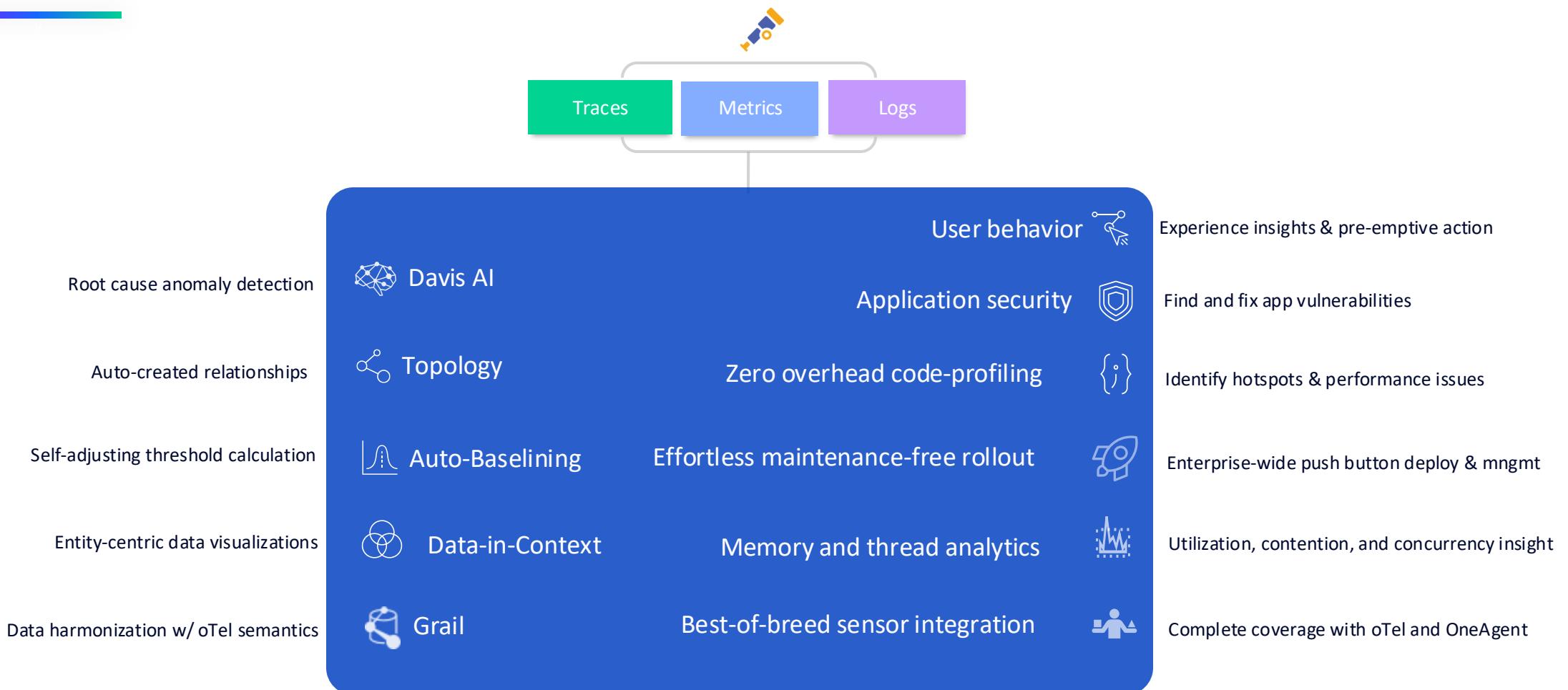
- **Use receivers** to collect telemetry data and manage data streams
- **Use processors** to offload telemetry processing from applications such as batch, filter, transform, etc.
- **Use exporters** to connect and send data to backends



OpenTelemetry + Dynatrace = Faster & Smarter



OpenTelemetry with Dynatrace



Powerful complements



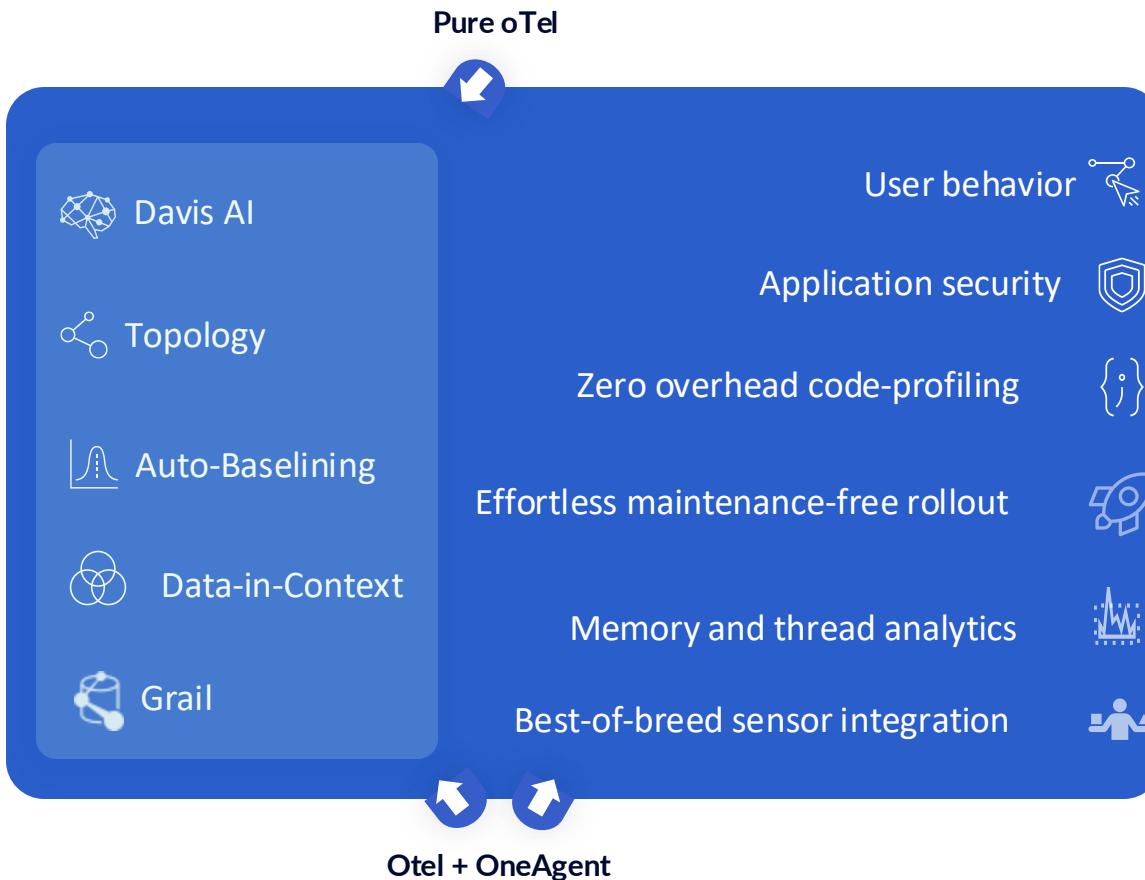
Pure oTel

A fully open implementation of OpenTelemetry combined with Dynatrace's context-aware intelligent observability.

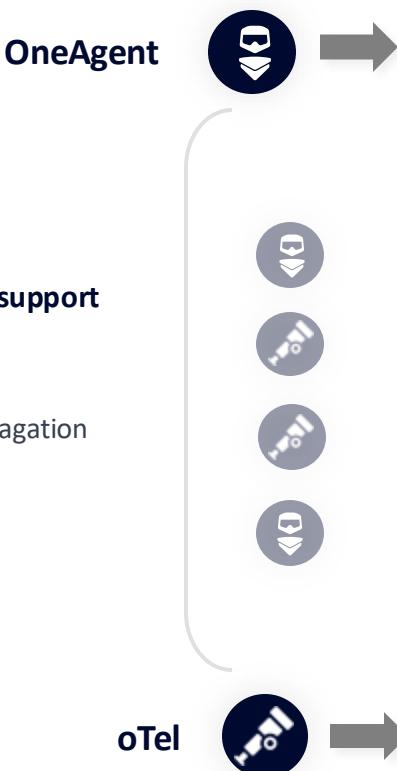
oTel + OneAgent



OpenTelemetry enhanced by 15 years of Dynatrace OneAgent innovation.



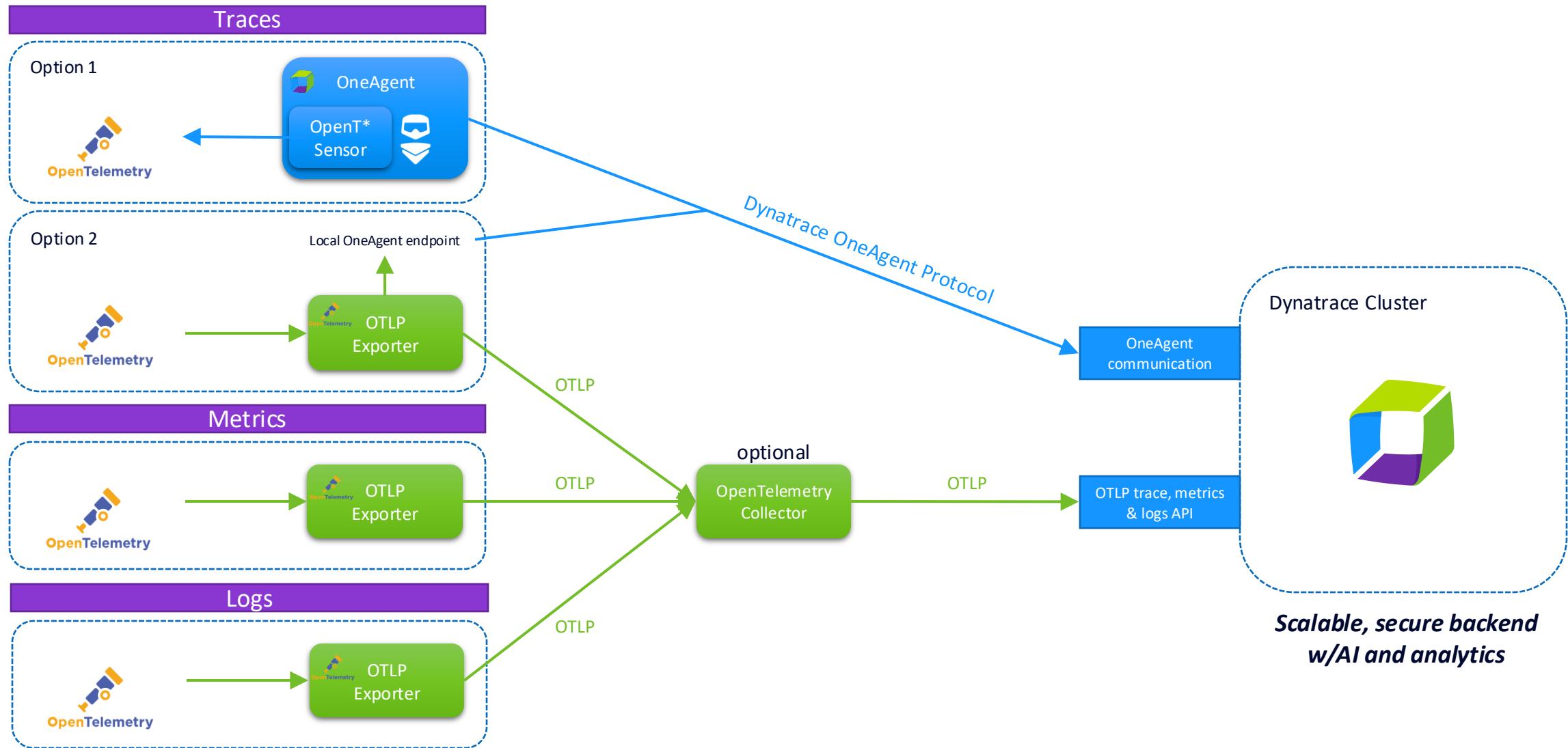
Best of Breed signal integration



Enhanced OpenTelemetry support

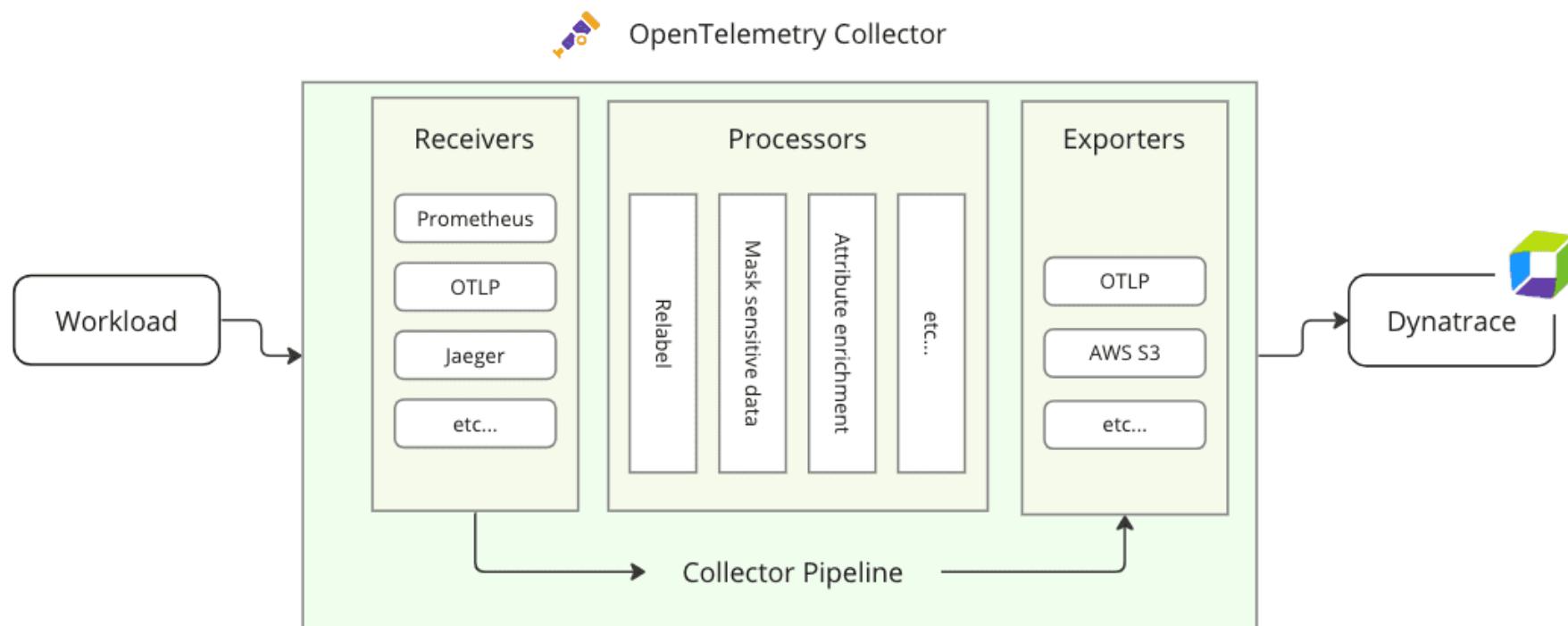
- Trace and Span ID auto-stitching
- Cross-agent OTLP
- Source-independent context propagation

Different ways to send OpenTelemetry to Dynatrace



Dynatrace Distro OpenTelemetry Collector

- <https://docs.dynatrace.com/docs/extend-dynatrace/opentelemetry/collector>
 - <https://github.com/Dynatrace/dynatrace-otel-collector>
- Check documentation or Github repository for latest supported plugins



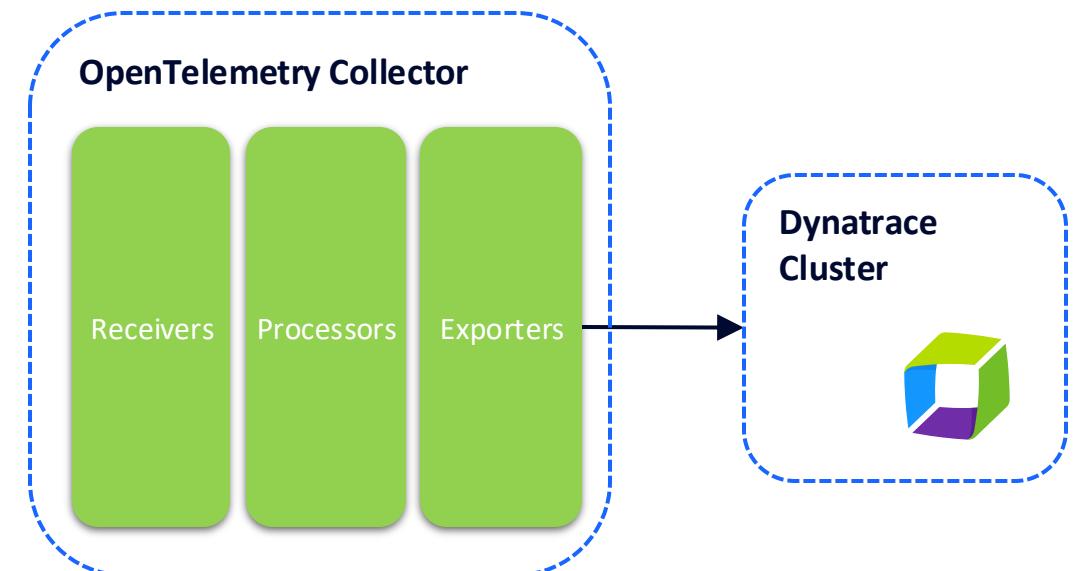
Unleashing the Power of Dynatrace Enterprise Collector for Unrivaled Support, Stability, and Speed

	Dynatrace Collector Distribution	Open Source Collector Distribution
Support	Dynatrace ONE Standard or Premium Support	Limited support
Collector Components	Stable and verified collector components to maximize service reliability	All collector components, including unstable and untested versions
Sample Configurations	Sample configurations, documentation, and support verified by Dynatrace	Sample configurations from the community and Dynatrace documentation
Release Cycles	Support for fast security patching, no dependency on the upstream release cycle	Release Cycles depending on open source milestones



Configuration - Export to Dynatrace

```
receivers:  
  otlp:  
    protocols:  
      http:  
      grpc:  
  
processors:  
  batch:  
  cumulativetodelta  
  
connectors:  
  spanmetrics:  
  
exporters:  
  otlphttp/dynatrace:  
    endpoint: "${DT_OTLP_ENDPOINT}"  
    headers:  
      Authorization: "Api-Token ${DT_API_TOKEN}"  
  
service:  
  pipelines:  
    traces/dynatrace:  
      receivers: [otlp]  
      processors: [batch]  
      exporters: [otlphttp/dynatrace, spanmetrics]  
    metrics/dynatrace:  
      receivers: [otlp, spanmetrics]  
      processors: [batch, cumulativetodelta]  
      exporters: [otlphttp/dynatrace]  
    logs/dynatrace:  
      receivers: [otlp]  
      processors: [batch]  
      exporters: [otlphttp/dynatrace]
```



OpenTelemetry Resources

The screenshot shows a YouTube search results page with the query "isitobservable" in the search bar. The top result is a channel named "Is it Observable" with a profile picture of a man with a beard and a hat. The channel has 9.05K subscribers and 115 videos. The description reads: "Welcome to Is it Observable, your go-to channel for in-depth tutorials on observability. ...more". Below the channel info are links to "github.com/isItObservable" and "4 more links". A "Subscribe" button is present. The navigation bar below the channel includes "Home", "Videos", "Shorts", "Live", "Playlists", "Community", and a search icon.

Cloud done right.

Dynatrace
@dynatrace • 15.7K subscribers • 539 videos
Dynatrace exists to make software work perfectly. The Dynatrace platform unifies obser ...more
dynatrace.com and 7 more links

OpenTelemetry
@otel-official • 1.35K subscribers • 27 videos
The official channel for the OpenTelemetry project on YouTube! ...more
opentelemetry.io and 4 more links

Questions?

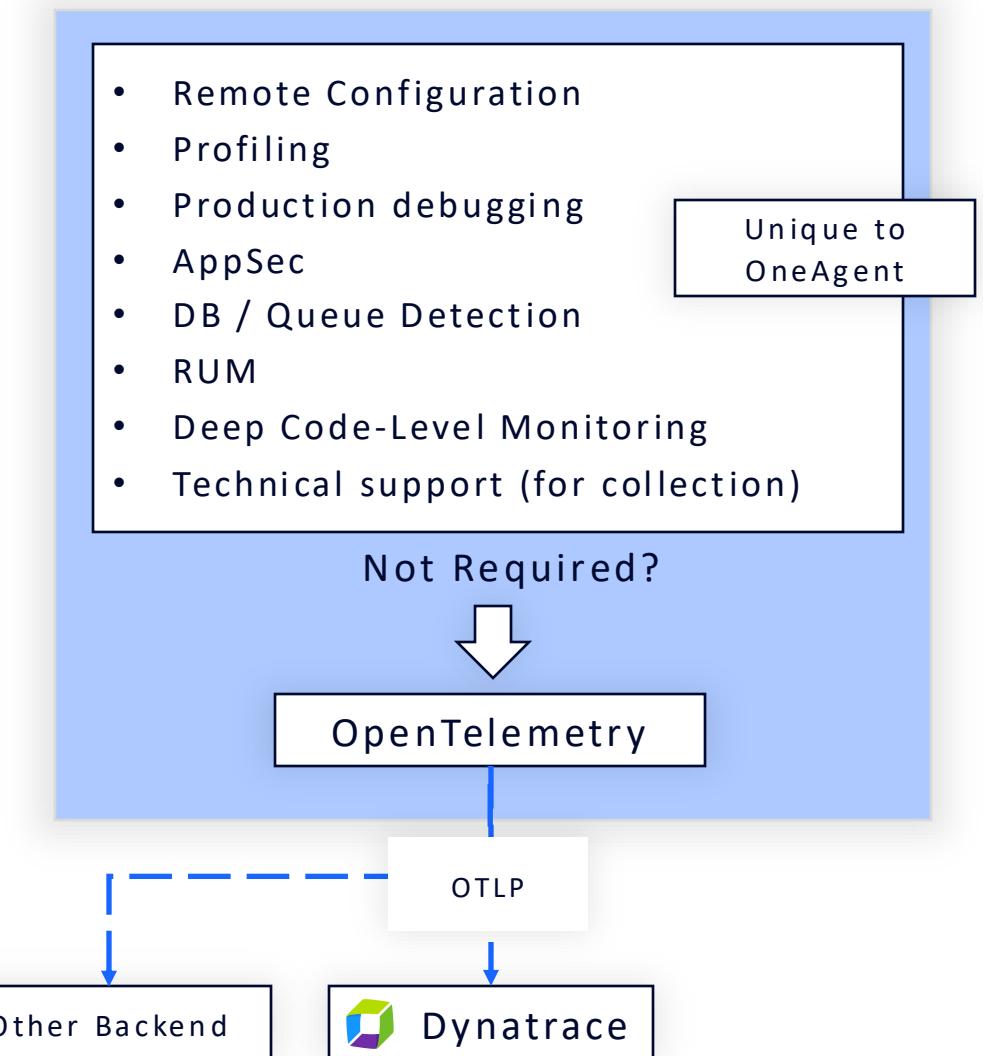


Appendix

Additional OpenTelemetry + Dynatrace Information

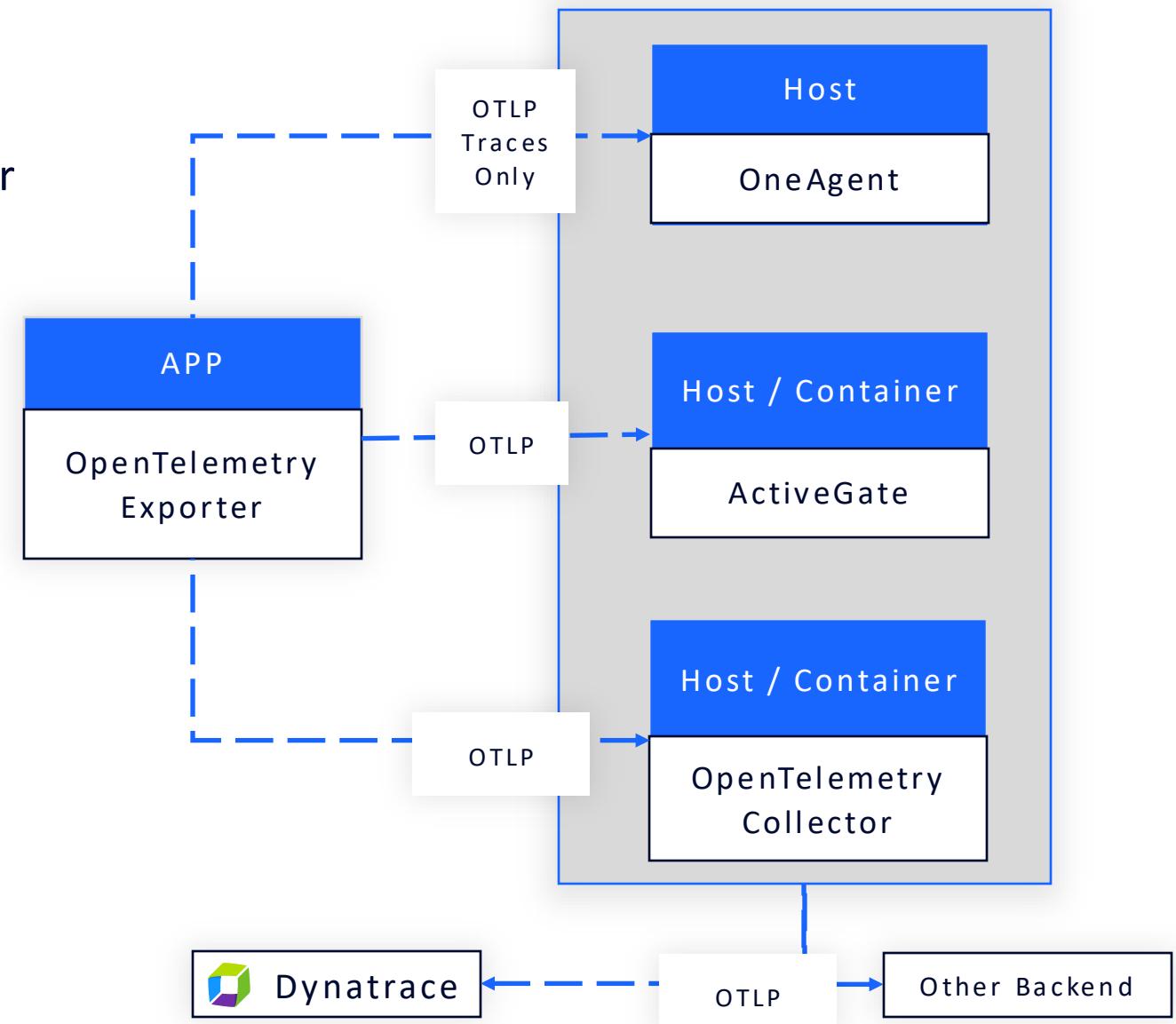
Instrument Services with OneAgent or OpenTelemetry

- Choose OneAgent or OpenTelemetry service-by-service
 - OneAgent offers Real User Monitoring (RUM), Profiling, Deep Code-Level Insights, Support, SLA's, etc.
 - OpenTelemetry offers a standard format supported by many backends.
- For OpenTelemetry start with auto-instrumentation
 - HTTP requests, database and API calls
 - Varies by language and framework
- Use OpenTelemetry manual instrumentation if needed
 - Mandatory for custom metrics and logs
 - Helpful for tracing specific flows
 - Must propagate context end-to-end



OpenTelemetry Collection Options

- Host with OneAgents as OpenTelemetry Collector
 - When OneAgent is deployed to host
 - Collects traces with licensing benefits
- Containerized or routable ActiveGates
 - When OneAgent is not available on host
 - Supports all OTLP signals at /v2/otlp/
- OpenTelemetry Collector
 - Fully supported Dynatrace OTEL Collector available
 - Use in place of (or in addition to) ActiveGate
 - Multiplex to different backends
 - Various processors available for different use-cases



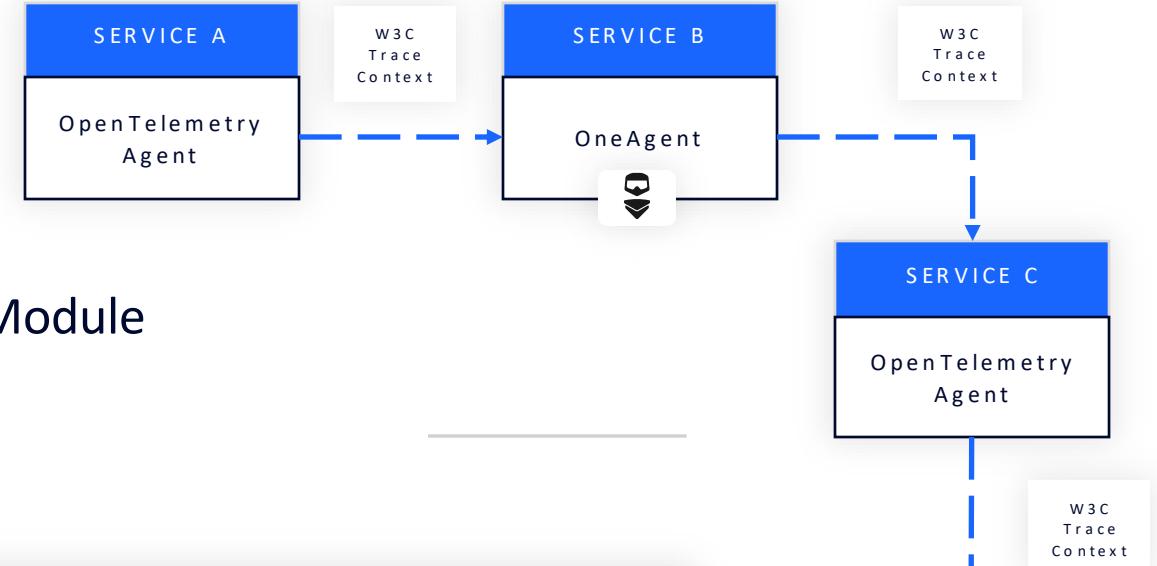
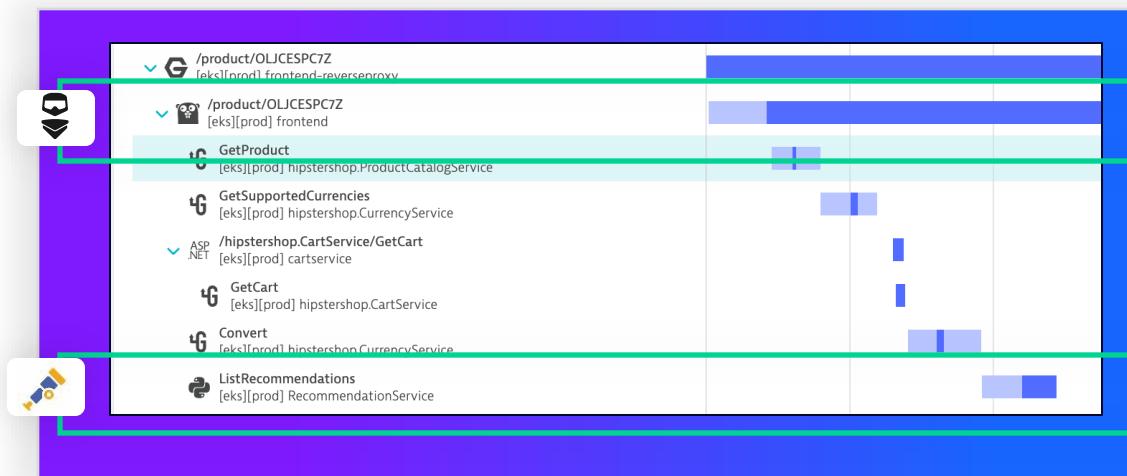
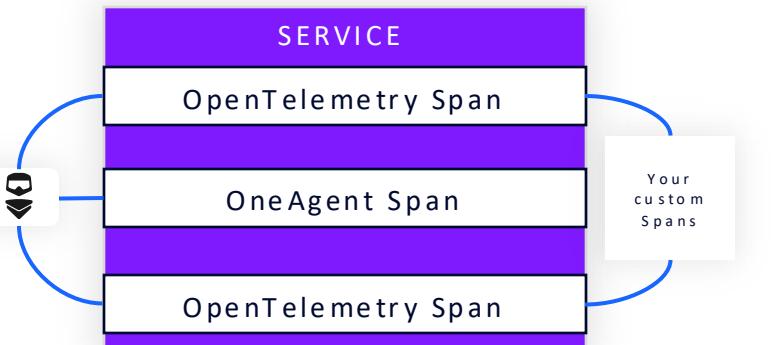
OpenTelemetry Agents and OneAgent Code Module Integration

- OneAgent OpenTelemetry Context Propagation

- OneAgents read/write W3C Context Headers
- Allows for a seamless blend across services
- OpenTelemetry for some apps and OneAgents for others

- OpenTelemetry Instrumentation with OneAgent Code Module

- An alternative way of creating "custom" spans
- Distributed traces with both OpenTelemetry and PurePaths
- Like OneAgent SDK but using OpenTelemetry conventions

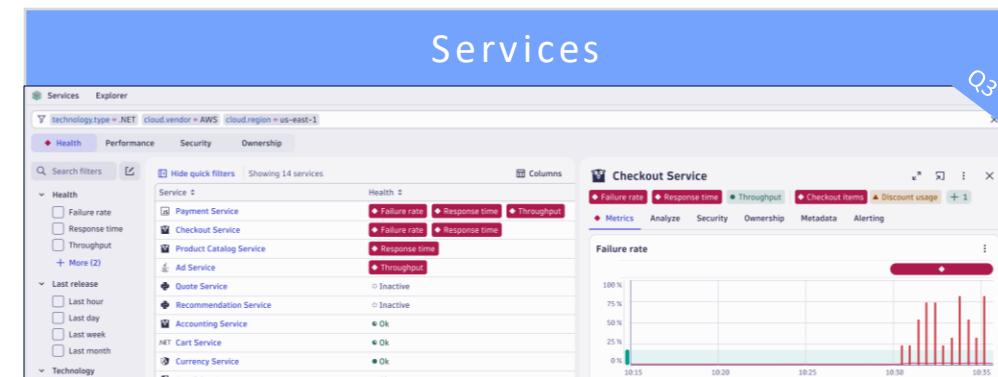
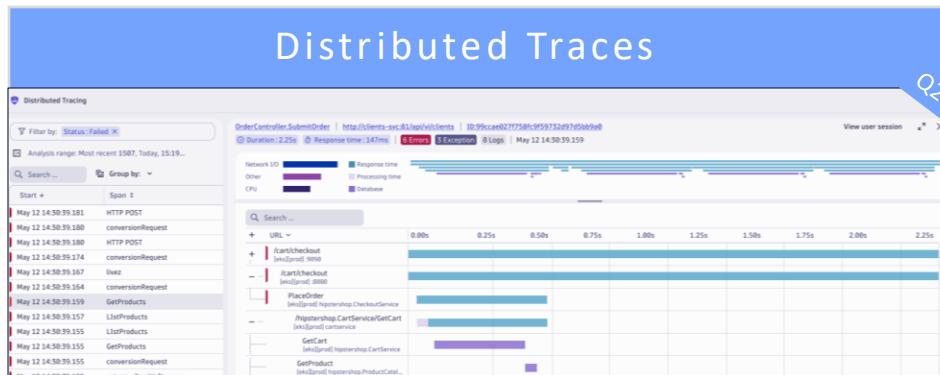


Native OpenTelemetry Across the Dynatrace Platform

- Grail Data Model aligns OpenTelemetry Semantic Conventions
 - Traces: Exceptions, HTTP, Messaging, Databases
 - Metrics: System, Hardware, Process
 - Logs: General attributes
- Unified with Dynatrace Query Language (DQL)
 - Semantic conventions in DQL used across Grail applications
 - Ensures seamless integration of OpenTelemetry signals

DQL Traces Powered by Grail

```
fetch spans
| filter request.is_root_span == true
| makeTimeseries
{ failed_requests=countIf(request.is_failed == true) },
time:start_time
```



Unified Observability in a Single Platform

- Everything in context (example: Kubernetes)
 - Workload info opens OTEL Services and Distributed Tracing experiences
 - High-level and detailed OTEL log viewer
 - Additional Telemetry with associated (OTEL/Prometheus) metrics
 - 3rd party and code-level security vulnerabilities
- DQL Across all signals
 - Pinpoint issues and understand system interactions quickly
 - Example: 500 errors (spans) joined with a specific log string

DQL Across all signals

Traces with 500's from decreased memory

```
fetch spans
| filter http.response.status_code >= 500
| filter trace.id in [
  fetch logs | filter isNotNull(trace_id)
  | filter matchesPhrase(content, "Memory limit decreased")
  | fields @id(trace_id)
]
| summarize count(), by: { service.name }
| sort 'count()' desc
```

Example: spans joined with logs

bookeeping
orders

● bookeeping
● orders
● portal
● deliveries

Service Info

Vulnerabilities

3 third-party vulnerabilities No code-level vulnerabilities

Denial of Service (DoS) S-197: io.netty.netty-codec-http2 Risk assessment:

Denial of Service (DoS) S-196: io.netty.netty-handler Risk assessment:

Allocation of Resources Without Limits or Throttling S-273: io.netty.netty-codec-http Risk assessment:

Everything in context

Deployments 27 records queried

One cluster with monitoring issues

Davis AI Workloads 27

Health Utilization Metadata

Workload	Age	Problems
antrea-controller	17 w 4 d	0
antrea-controller-horizontal-auto...	17 w 4 d	0
dynatrace-operator	15 w	0
dynatrace-webhook	15 w	0
egress-nat-controller	17 w 4 d	0
event-exporter-gke	17 w 4 d	0
open-telemetry-demos	17 w 4 d	0

opentelemetry-demo-adservice Workload

Davis AI Health status

Overview Info Utilization Logs Events Problems SLOs Ownership Vulnerabilities Additional Telemetry

Workload utilization

CPU Usage 14 millicore Requests 500 millicore Limits 500 millicore

Memory Usage 438 MB Requests 2 GiB Limits 2 GiB

Pods Running 1 Desired 1

Logs

Show the last 100 errors

```
fetch logs
| filter dt.entity.cloud.application == "CLOUD_APPLICATION-5CABBE45A387E81"
| filter status == "ERROR" OR status == "WARN"
| sort timestamp desc
```

Additional Telemetry

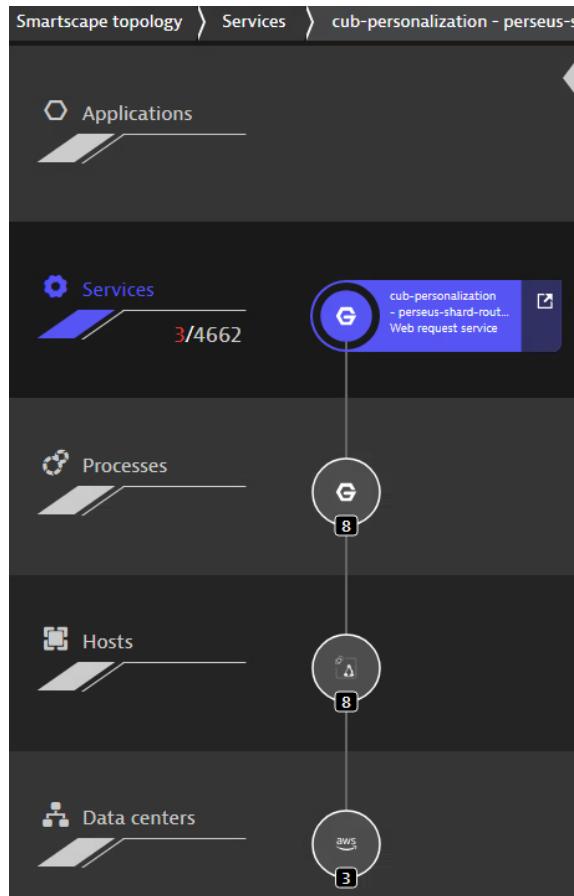
OpenTelemetry Metric

http.server.active_requests

orders_fulfilled

Topology – Entity Model

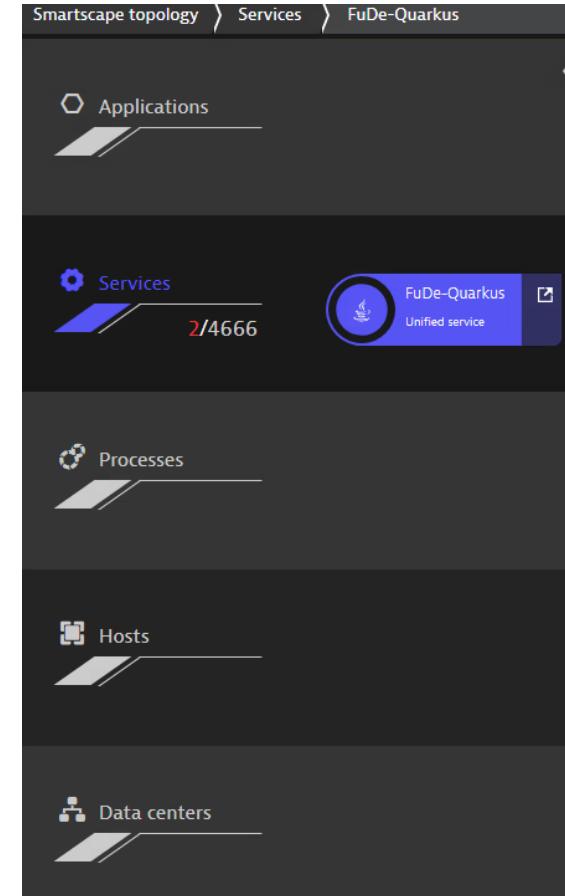
OTEL data
collected via OneAgent



Service detection for
OneAgent + OTEL traces

Entities for OneAgent-
collected infrastructure
metrics

OTEL data
ingested via API

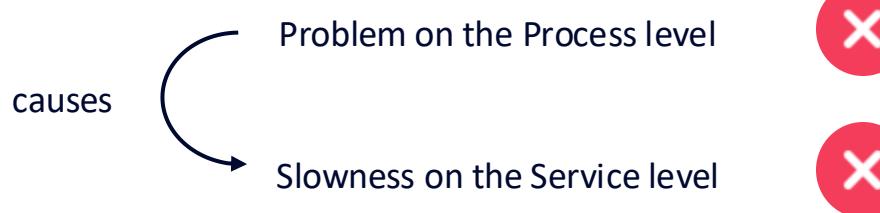


Service detection
for OTEL traces

No entities
for process, host,
data center

Topology – No connection to underlying process / hosts

- OpenTelemetry does not have a concept for an entity model and its relationships
- There is no link between Services and Processes / Hosts
- Problems detected on Service (span) level cannot be ascribed to problems detected on the Process / Host levels
- DAVIS AI cannot correlate issues on one level with issues on other levels without relationships



The screenshot shows the DAVIS AI interface. At the top right, it says "DAVIS® analyzed dependencies across 140,843 monitored entities". Below that, a section titled "Root cause" states "Based on our dependency analysis all incidents have the same root cause". It lists several entities: "CASA AbstractTimerBean - amcasa - prod" (Custom service), "Kubernetes cluster", "Kubernetes namespace", "Kubernetes service", and "Kubernetes workload". To the right of these, labels "amlcsm", "amcasa", "am-casa-frontend", and "am-casa-frontend" are aligned vertically. Below this, two incident cards are shown: "13 Long garbage-collection time events" (Garbage collection is suspending process amcasa (am-casa-frontend-759c66cdd6-857mn) on host ip-10-4-228-227.eu-ce...) and "Service response time degradation event" (The current response time (~6.11 s) exceeds the auto-detected baseline (~2.07 ms) by 294976.72 %. Service CASA Abstrac...). Both cards mention the service "CASA AbstractTimerBean - amcasa - prod".

Metadata Enrichment via OneAgent

- Infra-only OneAgent can enable metadata enrichment
- Enrich spans with process/host ID to enhance vertical topology
- Manual OTEL instrumentation needed

Services > ruby-manual-quickstart

ruby-manual-quickstart Inactive for 8 h 30 min
Overview of the Service

Properties and tags | No problems | 0 SLOs | Owners

Endpoints

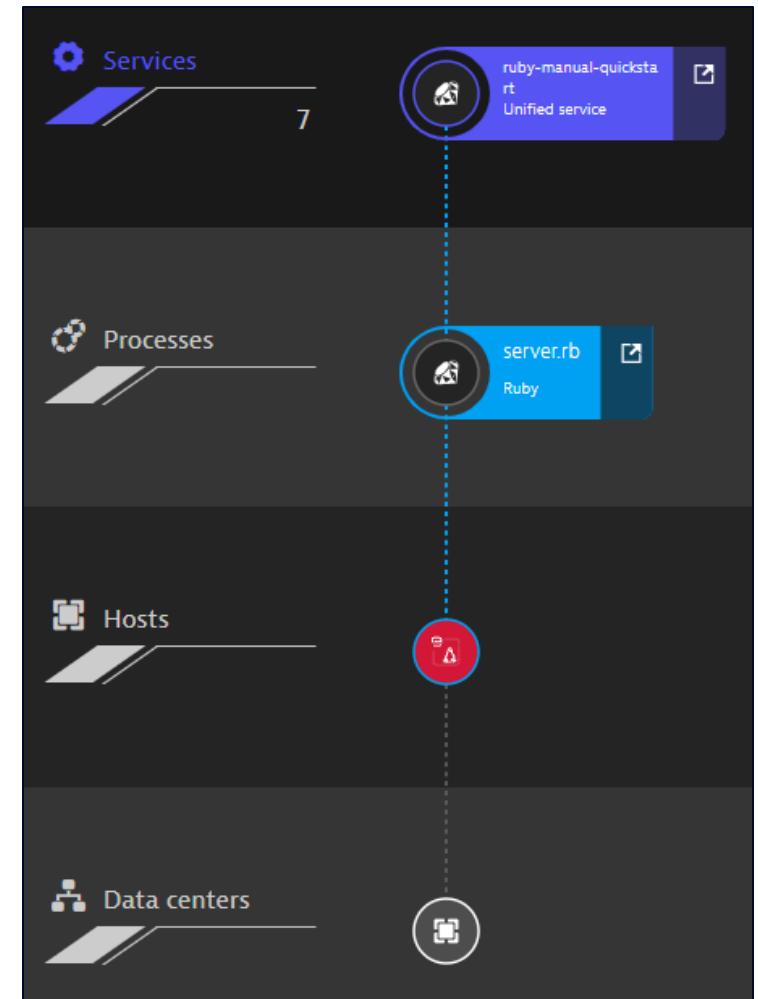
Endpoint	Response time [Median]	Failures	Failure rate	Throughput	Actions	Details
/ctx	1.17 ms	0	0 %	1.3 /min		
Call to /myendpoint	25.5 µs	0	0 %	1.3 /min		

Topology

Related services (0) | Processes and hosts (1)

Processes and hosts
Contains 1 Process.

Name	State	Technology	Host	Process CPU usage [Average]	Process memory [Average]	Details
server.rb	Inactive for 8 h 29 min	Docker, Ruby	qemu-ubuntu-24-its	0 %	0 B	



Observability with OpenTelemetry – Organizational Considerations

- OpenTelemetry experts/team needed
- Enablement for dev teams
- Company-specific best practices for instrumentation & guidance
- Planning of OTEL instrumentations in dev sprints
- Planning of agent and collector deployments

DPS Licensing for OTEL data

Data type	Ingest type	Rate card item	Unit of measure
Metrics	API	Metrics – Ingest & Process	Metric data points
	-	Metrics – Retain	GiB-days (15 months included)
Traces	API	Custom Traces Classic	Spans
	OneAgent	Full-Stack Monitoring (OTEL spans included)	Memory-GiB-hours
Logs	API	Logs – Ingest & Process	GiB
	-	Logs – Retain	GiB-days
	-	Logs – Query	GiB scanned

Exporting Signals with OTLP

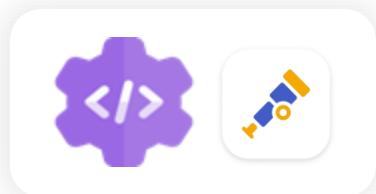
Sending metrics, traces, and logs to Dynatrace

Export with OTLP

- Dynatrace API – OTLP endpoints
 - `https://{{environment-id}}.live.dynatrace.com/api/v2/otlp/v1/metrics`
 - `https://{{environment-id}}.live.dynatrace.com/api/v2/otlp/v1/traces`
 - `https://{{environment-id}}.live.dynatrace.com/api/v2/otlp/v1/logs`
- Protocols
 - OTLP HTTP supported
 - OTLP gRPC not supported (can be converted via OTEL collector)
- Protobuf payload encoding
 - Binary format supported
 - JSON format is not supported

Metrics

- Metrics sent directly to Dynatrace SaaS cluster



OTEL Manual
Instrumentation

OTLP

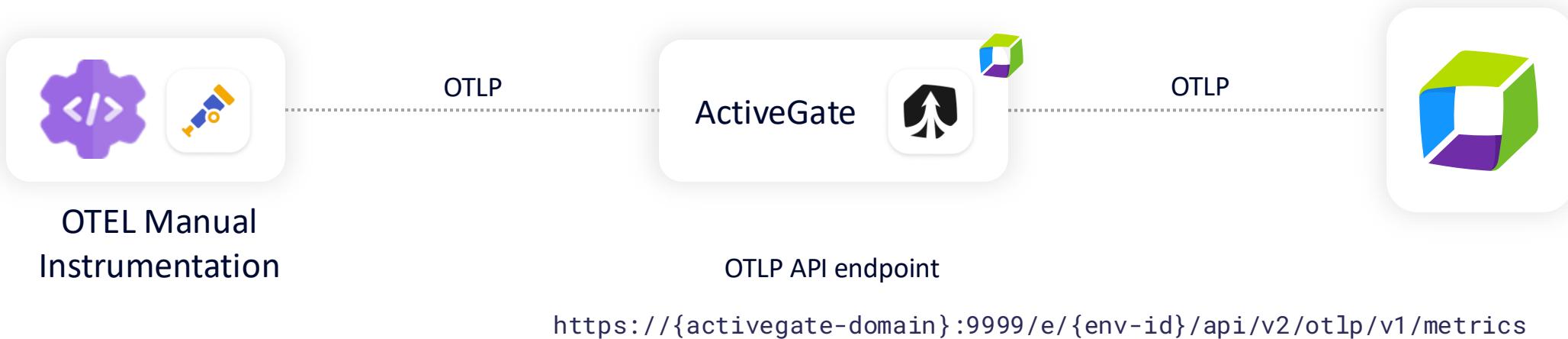


OTLP API endpoint

<https://{{env-id}}.live.dynatrace.com/api/v2/otlp/v1/metrics>

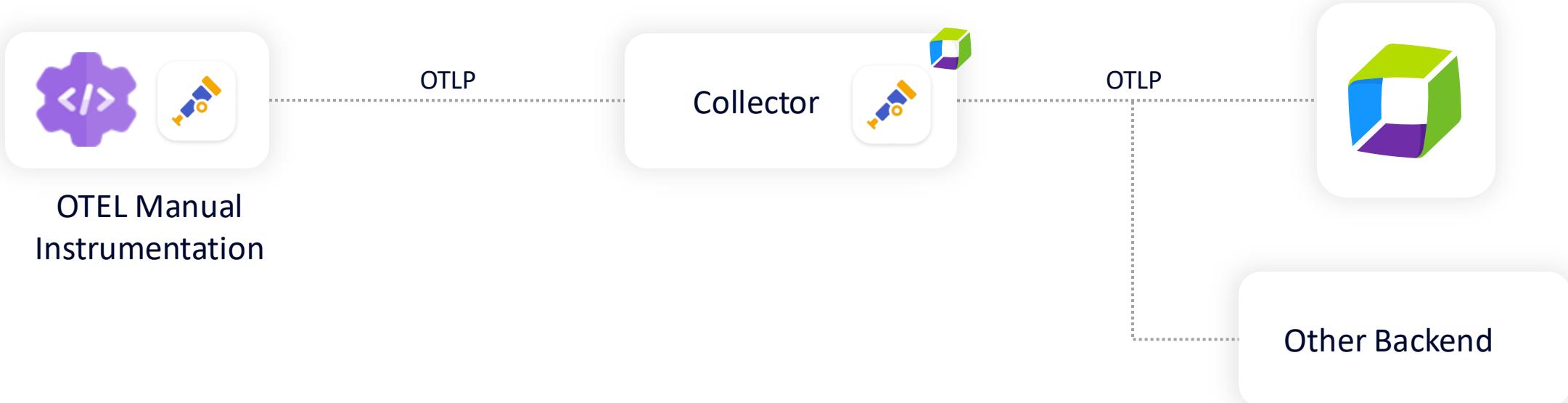
Metrics

- Metrics sent via ActiveGate



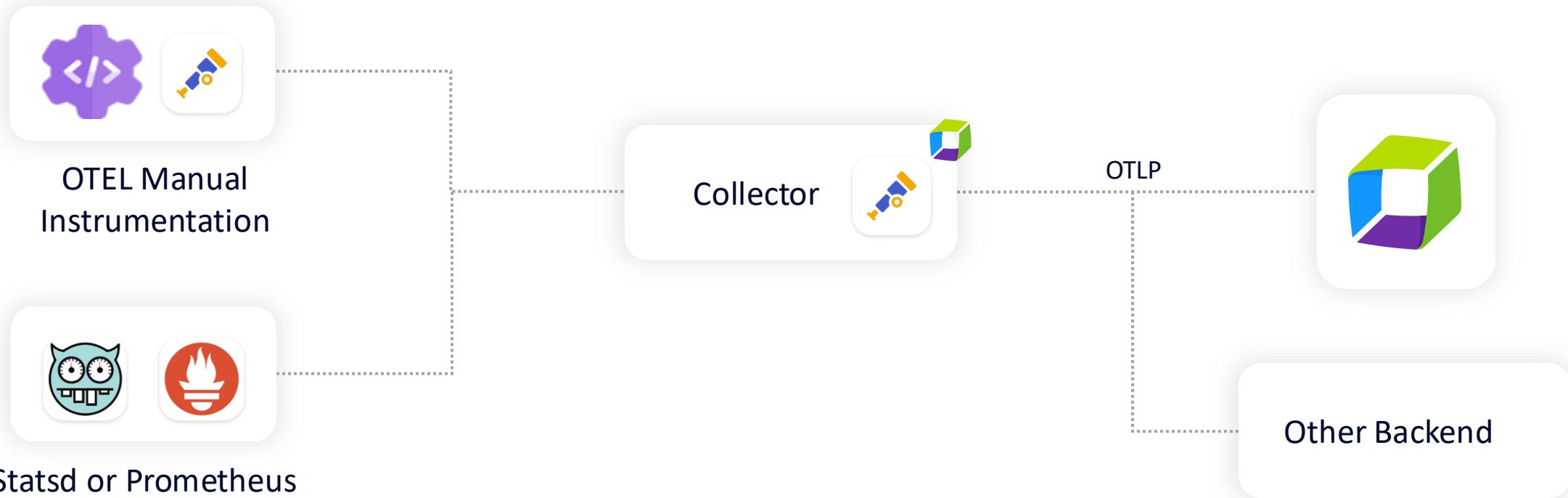
Metrics

- Metrics sent via OTEL Collector



Metrics

- Collector can receive metrics also from other sources



Metrics – Manual Instrumentation

- <https://docs.dynatrace.com/docs/extend-dynatrace/opentelemetry/getting-started/metrics/samples>

```
1 DynatraceMetricExporter dynatraceMetricExporter = DynatraceMetricExporter.builder()
2     .setUrl("<URL>")
3     .setApiToken("<TOKEN>")
4     .build();
5
6 SdkMeterProvider meterProvider = SdkMeterProvider.builder()
7     .registerMetricReader(PeriodicMetricReader.builder(dynatraceMetricExporter).build())
8     .build();
9
10 OpenTelemetry openTelemetry = OpenTelemetrySdk
11     .builder()
12     .setMeterProvider(meterProvider)
13     .buildAndRegisterGlobal();
14
15 Meter meter = openTelemetry.meterBuilder("instrumentation-library-name")
16     .setInstrumentationVersion("1.0.0")
17     .build();
18
19 Attributes attributes = Attributes
20     .of(AttributeKey.stringKey("my-key-1"), "my-value-1",
21         AttributeKey.stringKey("my-key-2"), Long.valueOf(new Random().nextInt(3)).toString());
```

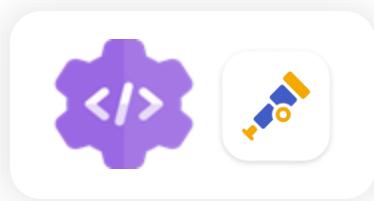
```
1 private static LongCounter counter;
2
3 counter = meter
4     .counterBuilder("my-counter")
5     .setDescription("This is my cool counter.")
6     .build();
7
8 counter.add((long) new Random().nextInt(11), attributes)
```

- Configuration & limits:

- Histograms are not supported (only bucket summaries possible)
- <https://docs.dynatrace.com/docs/extend-dynatrace/opentelemetry/getting-started/metrics/configuration>
- <https://docs.dynatrace.com/docs/extend-dynatrace/opentelemetry/getting-started/metrics/limitations>

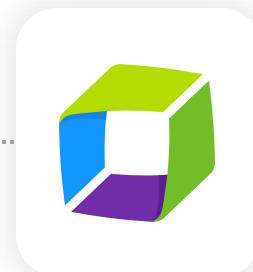
Traces

- Traces sent directly to Dynatrace SaaS cluster



OTEL Manual
Instrumentation
and/or OTEL Agent

OTLP

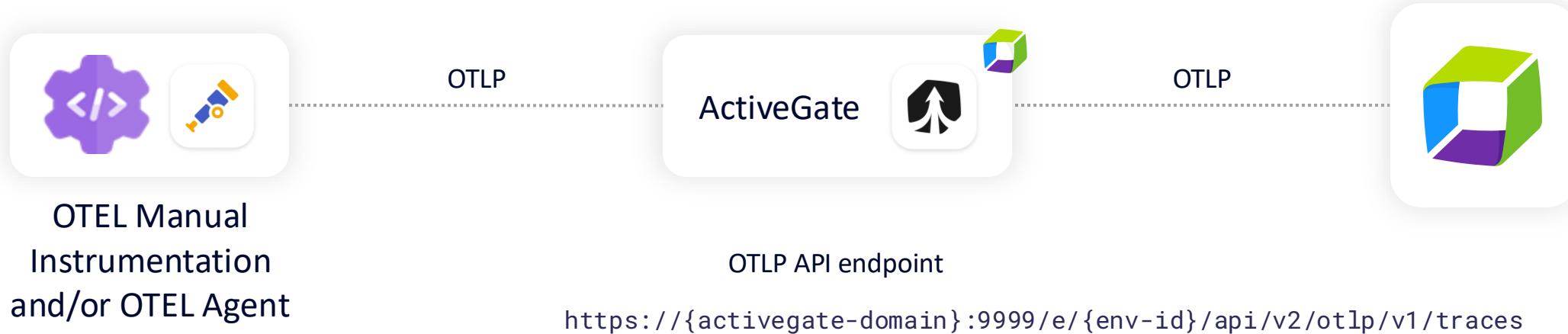


OTLP API endpoint

<https://{{env-id}}.live.dynatrace.com/api/v2/otlp/v1/traces>

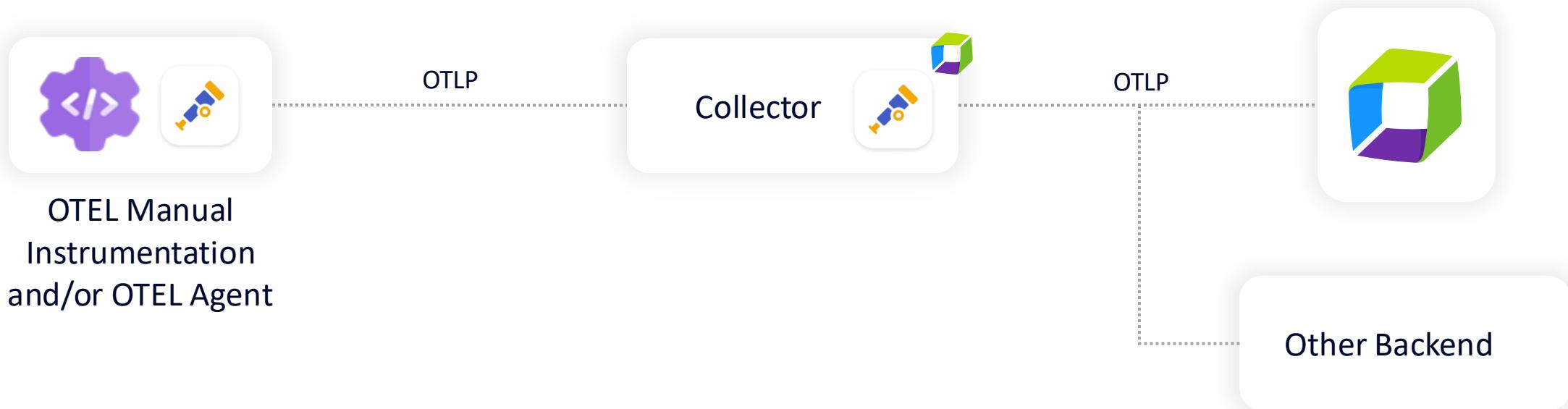
Traces

- Traces sent via ActiveGate



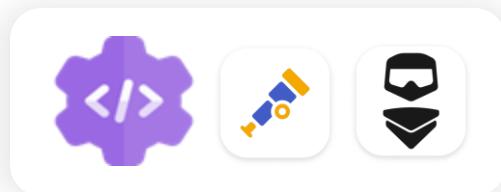
Traces

- Traces sent via OTEL Collector



Traces

- Traces sent via OneAgent
- OTEL spans + OneAgent spans will be collected
- It is not recommended to use an OTEL agent and OneAgent for the same service



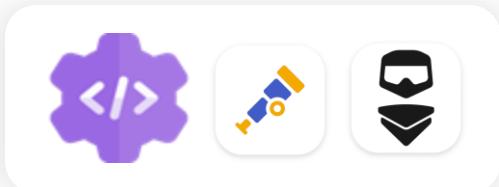
OneAgent + OTEL Manual
Instrumentation

OneAgent Protocol



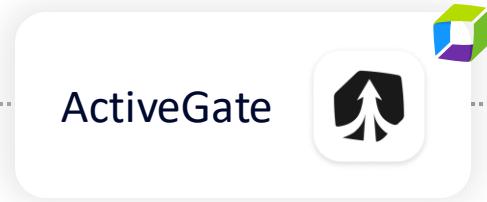
Traces

- Traces sent via OneAgent & ActiveGate
- OTEL spans + OneAgent spans will be collected
- It is not recommended to use an OTEL agent and OneAgent for the same service



OneAgent + OTEL Manual
Instrumentation

OneAgent Protocol

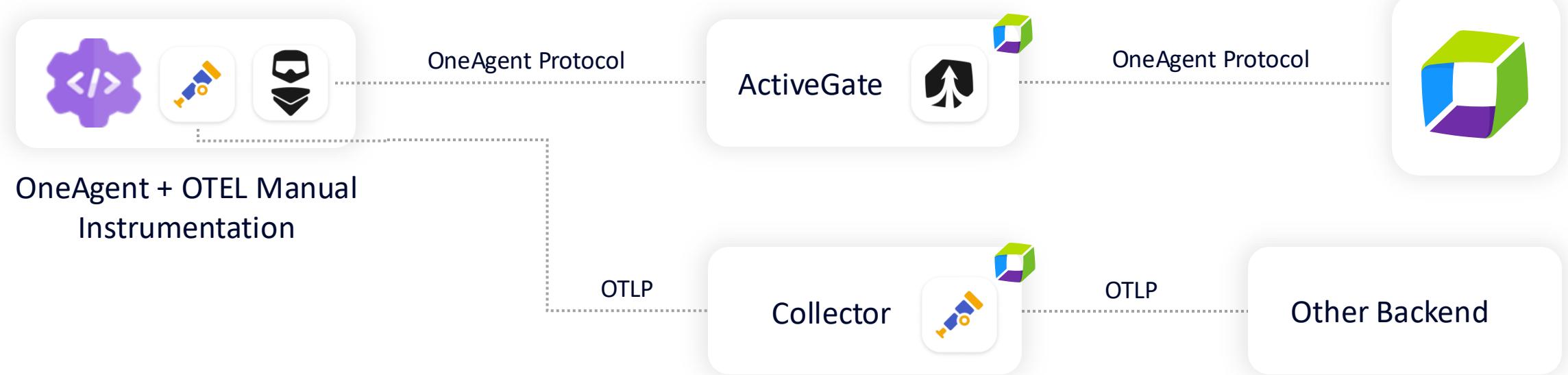


OneAgent Protocol



Traces

- Traces sent via OTEL Collector & OneAgent
- OTEL spans + OneAgent spans will be sent via OneAgent/ActiveGate
- OTEL spans will be sent via OTEL Collector



Traces – Instrumentation

- Manual Instrumentation:

```
1  Tracer tracer = GlobalOpenTelemetry
2      .getTracerProvider()
3      .tracerBuilder("my-tracer") //TODO Replace with the name of your application
4      .build();
```

- Auto-Instrumentation Agents:

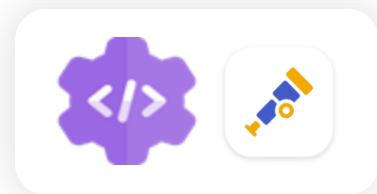
- Java
- PHP
- Python
- .NET
- Go
- Javascript (Node.js)
- Ruby

```
1  // Obtain and name new span from tracer
2  Span span = tracer.spanBuilder("Call to /myendpoint")
3      .setSpanKind(SpanKind.CLIENT)
4      .startSpan();
5
6  // Set demo span attributes using semantic naming
7  span.setAttribute("http.method", "GET");
8  span.setAttribute("net.protocol.version", "1.1");
9
10 // Set the span as current span and parent for future child spans
11 try (Scope scope = span.makeCurrent())
12 {
13     // TODO your code goes here
14 }
15 finally
16 {
17     // Completing the span
18     span.end();
19 }
```

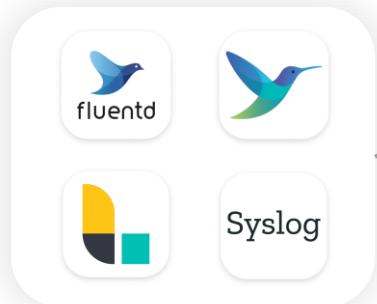
Logs



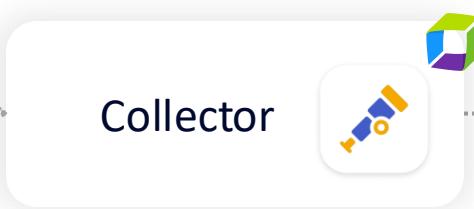
Filelog receiver



OTEL Log SDK



Fluentd, FluentBit,
Logstash, Syslog

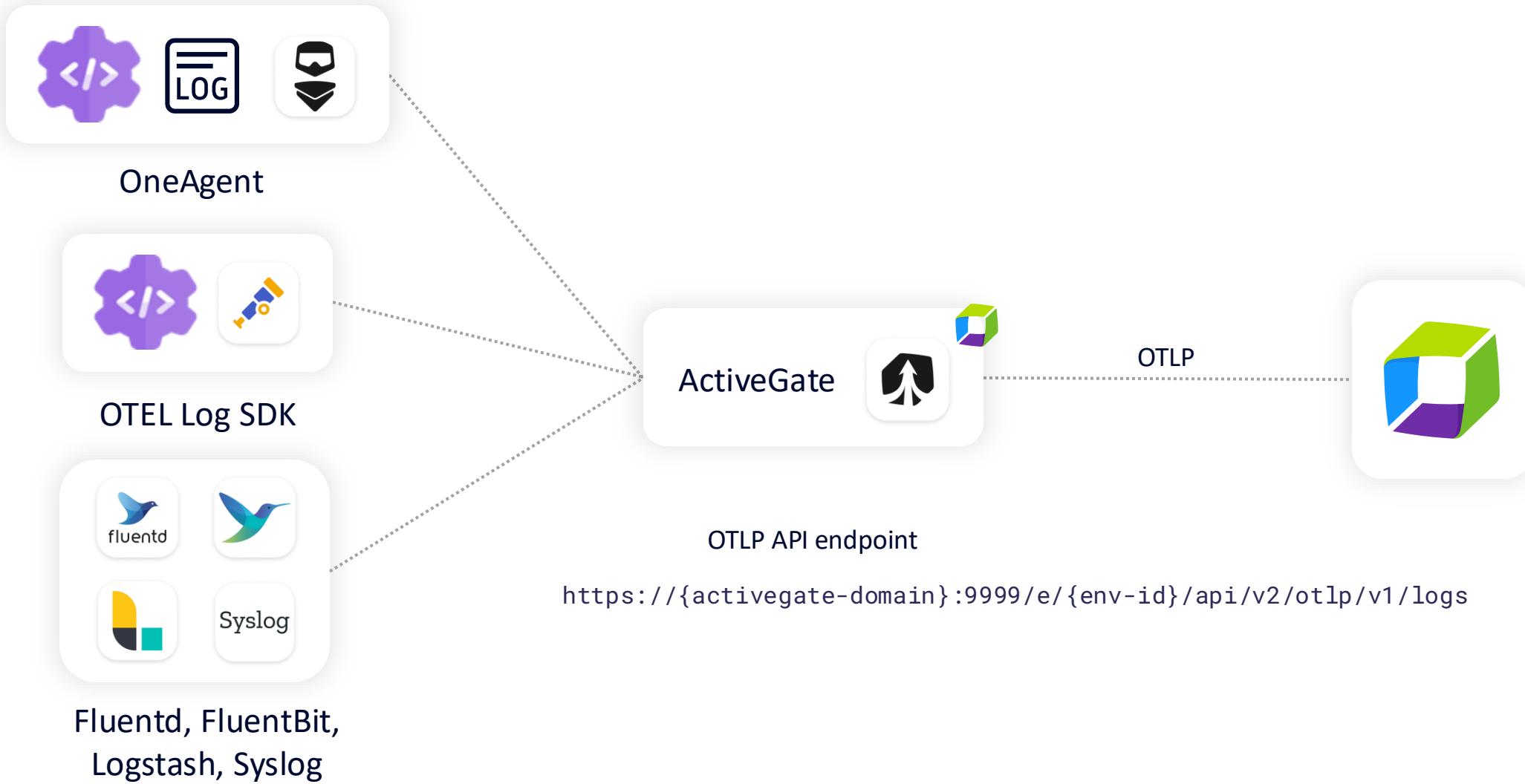


OTLP API endpoint
<https://{{env-id}}.live.dynatrace.com/api/v2/otlp/v1/logs>

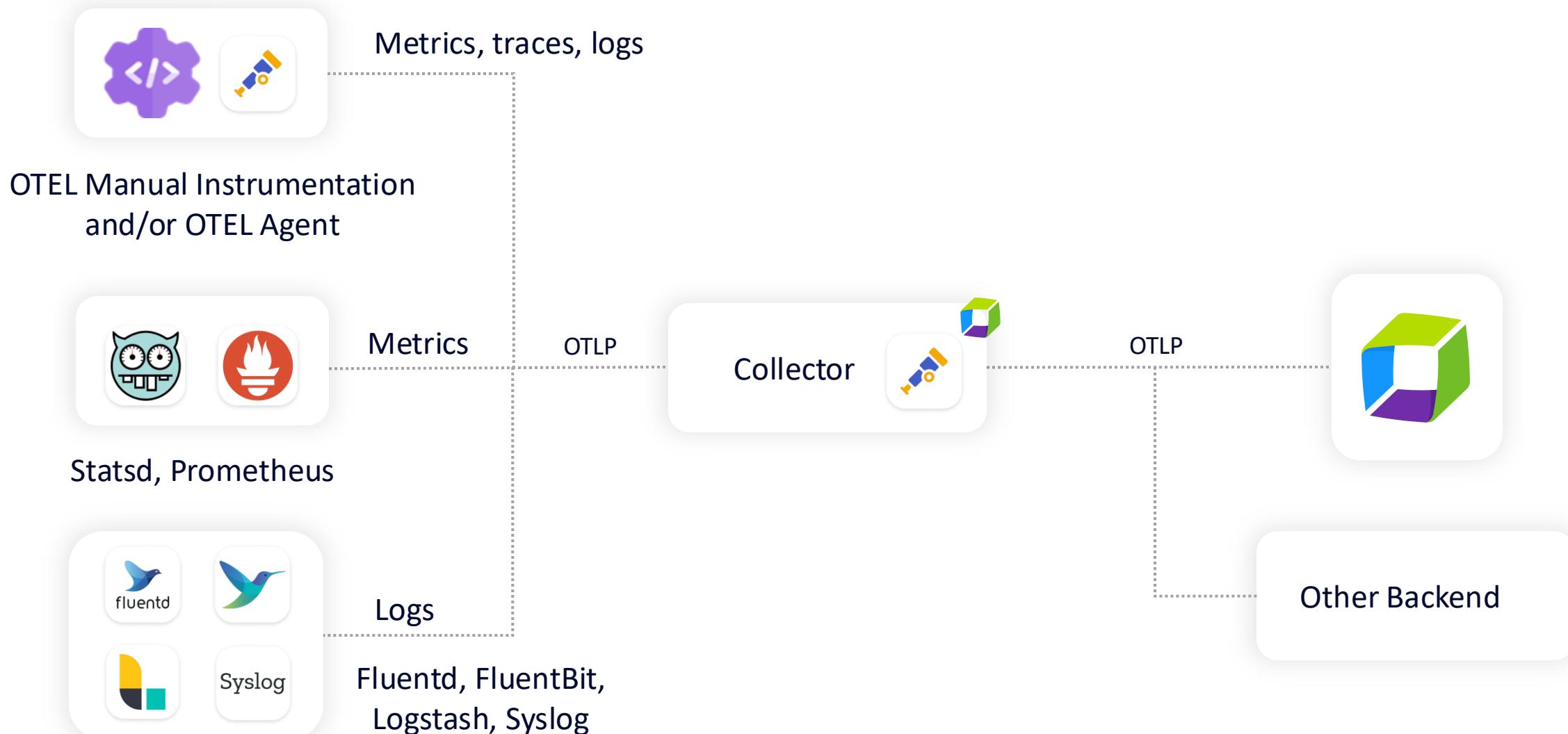


Other Backend

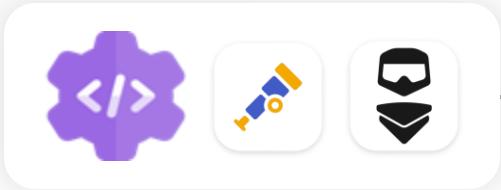
Logs



Metrics, Traces and Logs

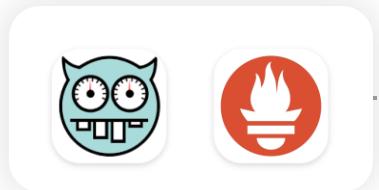


OTEL + OneAgent



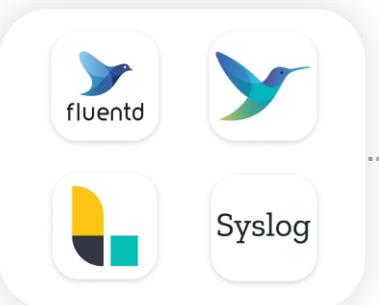
Metrics, traces, logs

OneAgent + OTEL Manual
Instrumentation



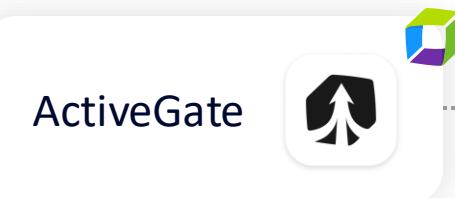
Metrics

Statsd, Prometheus



Logs

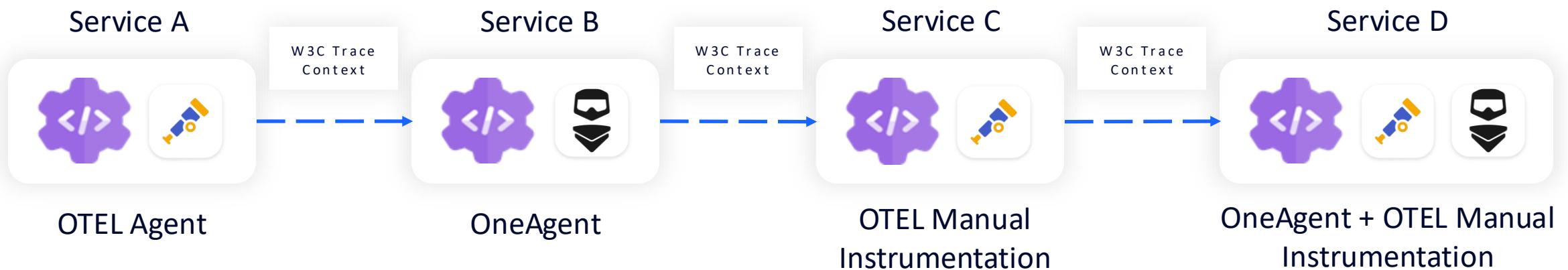
Fluentd, FluentBit,
Logstash, Syslog



- OneAgent for traces
- Custom OTEL spans are captured via OneAgent

Context Propagation

- W3C Trace Context is required to connect spans to a trace across services (HTTP/HTTPS only)
- OneAgents read/write W3C Context Headers
- Manual instrumentation must propagate context to avoid broken traces

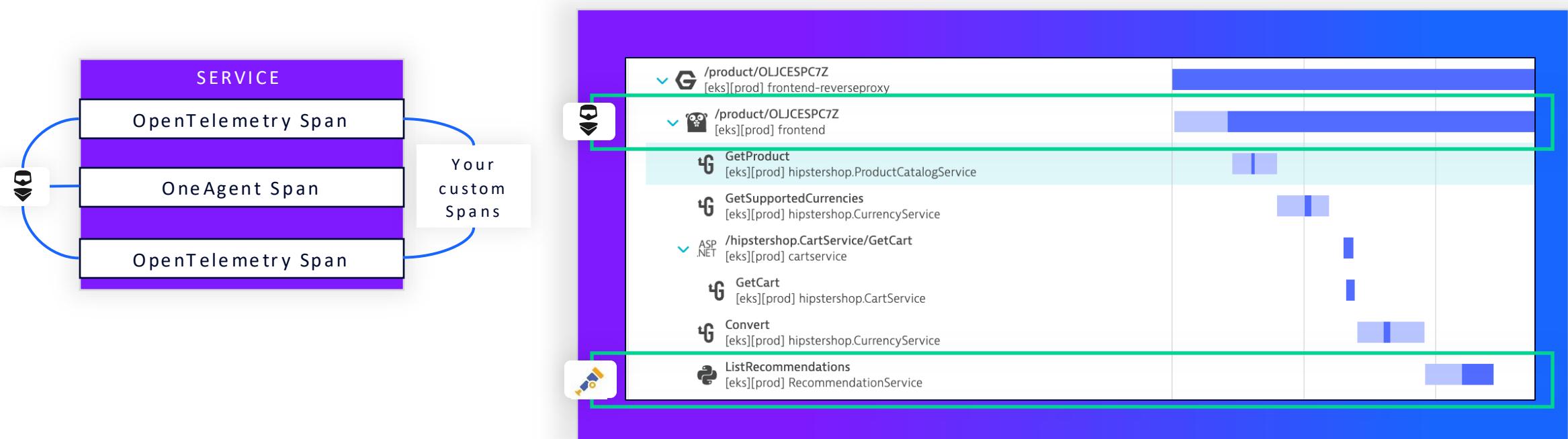


Use any collection option depending on use case

OpenTelemetry Agents & OneAgent Code Module Integration

- OpenTelemetry Instrumentation with OneAgent Code Module

- An alternative way of creating "custom" spans
- Distributed traces with both OpenTelemetry spans and PurePaths
- OneAgent will ignore spans from common OTEL auto-instrumentation libraries to avoid span duplication



Context Propagation – Manual Instrumentation

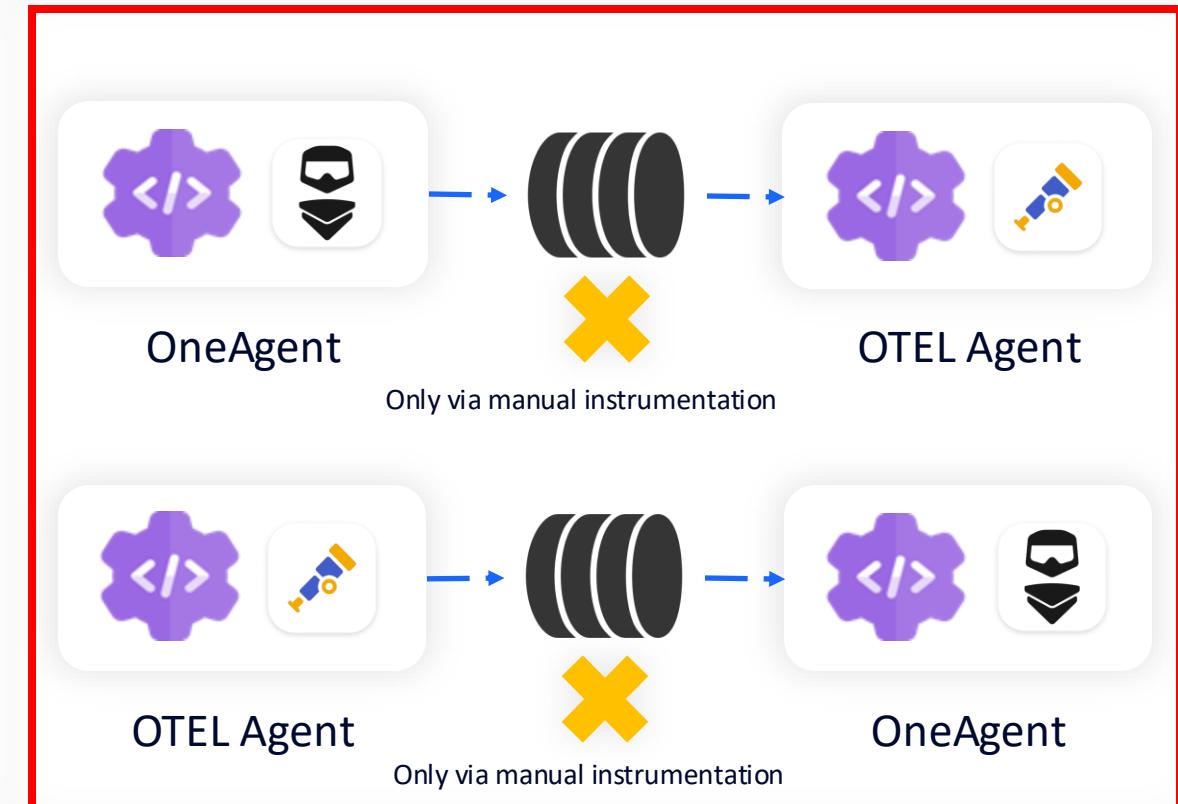
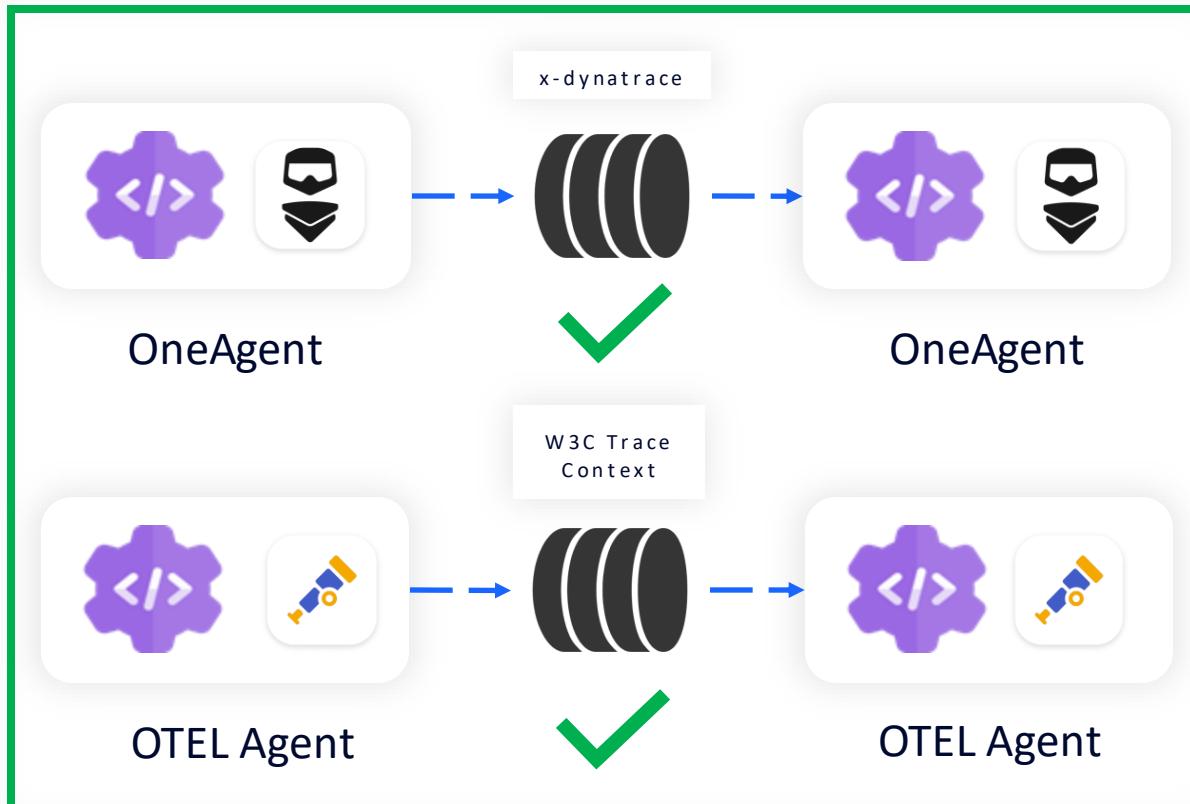
- Extract context when receiving requests
- Inject context when sending requests

```
1 //The getter will be used for incoming requests
2 TextMapGetter<HttpExchange> getter =
3     new TextMapGetter<>() {
4         @Override
5         public String get(HttpExchange carrier, String key) {
6             if (carrier.getRequestHeaders().containsKey(key)) {
7                 return carrier.getRequestHeaders().get(key).get(0);
8             }
9             return null;
10        }
11
12        @Override
13        public Iterable<String> keys(HttpExchange carrier) {
14            return carrier.getRequestHeaders().keySet();
15        }
16    };
17
18    @Override
19    public void handle(HttpExchange httpExchange) {
20        //Extract the SpanContext and other elements from the request
21        Context extractedContext = GlobalOpenTelemetry.getPropagators().getTextMapPropagator()
22            .extract(Context.current(), httpExchange, getter);
23        try (Scope scope = extractedContext.makeCurrent()) {
24            //This will automatically propagate context by creating child spans within the extracted context
25            Span serverSpan = tracer.spanBuilder("my-server-span") //TODO Replace with the name of your span
26                .setSpanKind(SpanKind.SERVER) //TODO Set the kind of your span
27                .startSpan();
28            serverSpan.setAttribute(SemanticAttributes.HTTP_METHOD, "GET"); //TODO Add attributes
29            serverSpan.end();
30        }
31    }
```

```
1 //The setter will be used for outgoing requests
2 TextMapSetter<HttpURLConnection> setter =
3     (carrier, key, value) -> {
4         assert carrier != null;
5         // Insert the context as Header
6         carrier.setRequestProperty(key, value);
7     };
8
9 URL url = new URL("<URL>"); //TODO Replace with the URL of the service to be called
10 Span outGoing = tracer.spanBuilder("my-client-span") //TODO Replace with the name of your span
11     .setSpanKind(SpanKind.CLIENT) //TODO Set the kind of your span
12     .startSpan();
13 try (Scope scope = outGoing.makeCurrent()) {
14     outGoing.setAttribute(SemanticAttributes.HTTP_METHOD, "GET"); //TODO Add attributes
15     HttpURLConnection transportLayer = (HttpURLConnection) url.openConnection();
16     // Inject the request with the *current* Context, which contains our current span
17     GlobalOpenTelemetry.getPropagators().getTextMapPropagator().inject(Context.current(), transportLayer, setter);
18     // Make outgoing call
19 } finally {
20     outGoing.end();
21 }
```

Context Propagation – Queues

- Tracing through queues and topics (Kafka, IBM MQ, etc.) either via DT or W3C trace context
- OneAgent on one end and OTEL on the other – currently no auto-propagation support
- <https://docs.dynatrace.com/docs/platform-modules/applications-and-microservices/queues/queue-concepts>



OpenTelemetry Data Pipeline

Scalability, Resiliency, and Security for Observability Signals

Message Routing

- OTEL SDK
 - Endpoints have to be configured for exporters
 - Retry mechanism has to be configured and might vary by SDK
- OTEL collector
 - Endpoints have to be configured for exporters
 - Retry and buffer functionality is enabled by default via otlphttpexporter and can be configured
 - <https://github.com/open-telemetry/opentelemetry-collector/blob/main/exporter/exporterhelper/README.md>
- OneAgent
 - Automatically connects to known endpoints (ActiveGates or directly to the Dynatrace cluster)
 - Checks for updates about new endpoints
 - Performs connection retries and checks known endpoints
 - Uses backup buffer for messages while communication is lost
- ActiveGate
 - Can only route to the Dynatrace cluster
 - Performs connection retries
 - Uses backup buffer for messages while communication is lost

Sampling of transactions

- OpenTelemetry
 - Head Sampling
 - Sampling within application via OTEL SDK
 - Random sample of transactions
 - Tail Sampling
 - Sampling within OTEL collector
 - Define policies for sampling (e.g. keep errors, traces longer than 500ms, and 20% of all remaining traces)
 - Risk of broken traces
 - Head sampling might create broken traces across services, especially with multiple services in the transaction chain
 - For tail sampling all spans from related services must be sent to the same collector instance to get end-to-end traces
- OneAgent
 - Adaptive traffic management
 - Sampling is only active if peak trace volume for Dynatrace environment has been exceeded
 - Intelligent sampling to maintain a statistically valid sample of all transactions and to capture end-to-end traces
 - Benefit of cost control and predictability
 - <https://docs.dynatrace.com/docs/observe-and-explore/distributed-traces/adaptive-traffic-management/adaptive-traffic-management-saas>

Scaling & Load Balancing

- ActiveGate
 - More ActiveGates can simply be added
 - Automatic load balancing in a round-robin fashion
 - Network zones to control monitoring traffic flow
- OTEL collector
 - A load balancer is needed in front of OTEL collectors
 - Scaling should be carefully planned (expected traffic per data type, elastic workloads and peaks, etc.):
<https://opentelemetry.io/docs/collector/scaling/>
 - Example:
 - Scaling Prometheus receivers might require coordination among scrapers
 - Cannot simply add more instances, endpoint configuration for scraping has to be adapted
 - Risk of broken traces when tail sampling is used

Network Bandwidth

- OTLP exporters
 - Default compression for OTLP exporters is not set and has to be configured
 - Gzip compression is available, some exporters might offer zstd compression
 - Gzip compression is recommended by Dynatrace
<https://docs.dynatrace.com/docs/extend-dynatrace/opentelemetry/best-practices/performance>
- OTEL collector
 - The collector can queue and batch OTLP requests to improve throughput performance
 - Batching and batch size has to be configured via batch processor
 - <https://docs.dynatrace.com/docs/extend-dynatrace/opentelemetry/collector/use-cases/batch>
- OneAgent
 - Uses snappy compression for payloads above 2kB by default
 - Could be configured to use zstd compression
 - Adaptive traffic management saves network bandwidth on peak loads
- ActiveGate
 - Takes care of buffering and compression
 - Uses zstd compression
 - Considerably reduces the network overhead

Encryption

- OTEL SDK / Collector
 - TLS 1.2 encryption has to be used
 - Configured by default for SDK/Collector but not all gRPC exporters
- OneAgent / ActiveGate
 - All data exchanged between OneAgent, ActiveGate, and Dynatrace Cluster is encrypted in transit
 - Data is serialized and deserialized using Google Protocol Buffers
 - Dynatrace SaaS supports TLS 1.2 and TLS 1.3 (SSL Labs Grade A+)

Maintenance

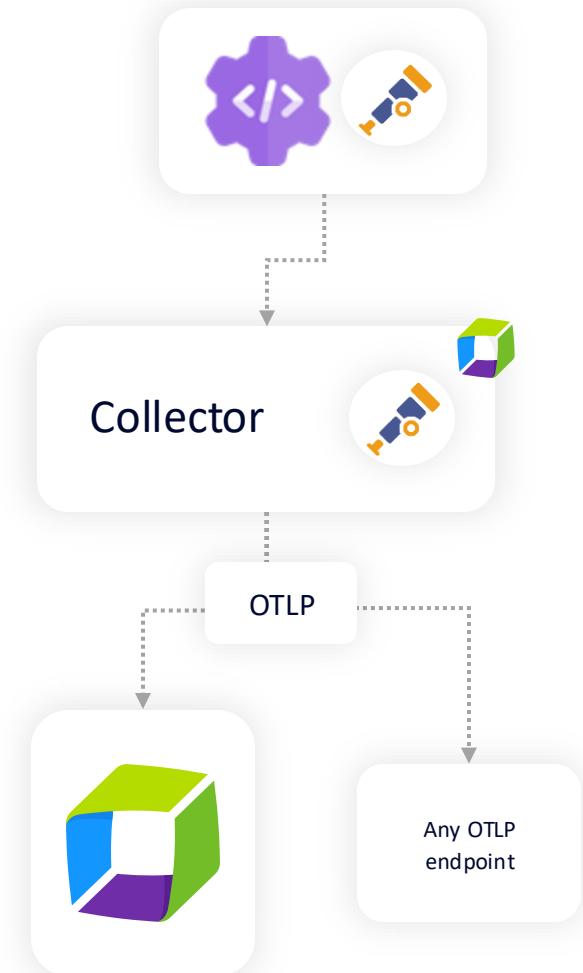
- OpenTelemetry
 - Manual updates required for collectors
 - Manual updates required for agents
 - In Kubernetes environments updates for agents/collectors can be done via Operator
- OneAgent
 - Automatic or manual updates possible
- ActiveGate
 - Automatic or manual updates possible

OpenTelemetry Collector

Additional Information

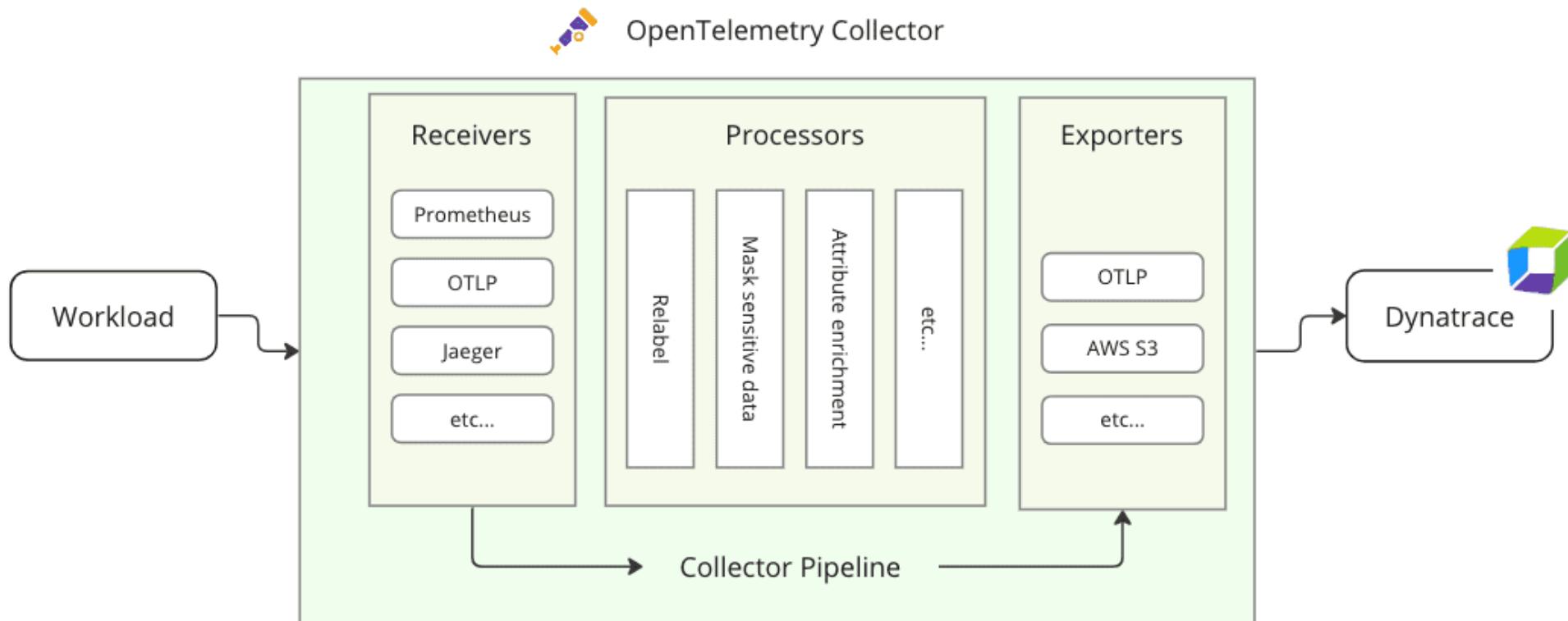
Dynatrace OpenTelemetry Collector Distribution

	Dynatrace Collector Distribution	Open Source Collector Distribution
Support	Dynatrace ONE Standard or Premium Support	Limited support
Collector Components	Stable and verified collector components to maximize service reliability	All collector components, including experimental and unstable versions
Sample Configurations	Sample configurations, documentation, and support verified by Dynatrace	Sample configurations from the community and Dynatrace documentation
Release Cycles	Support for fast security patching, no dependency on the upstream release cycle	Release Cycles depending on open source milestones



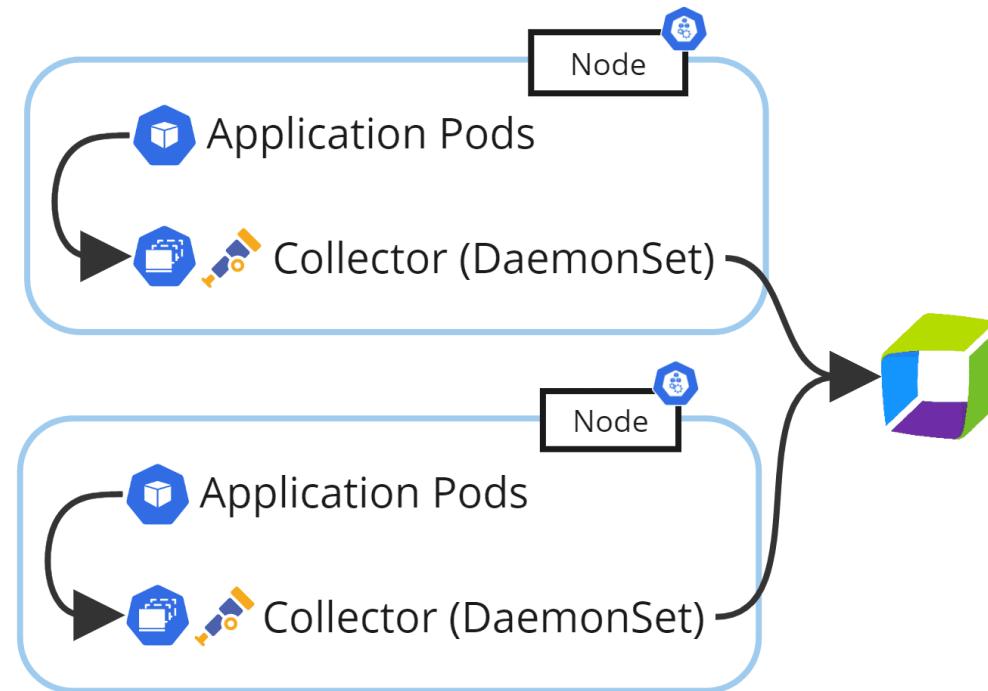
OTEL Collector

- Collector deployment modes: Agent, Gateway, Sidecar
- Allows to convert various formats
- Enables edge processing (e.g. mask sensitive data)



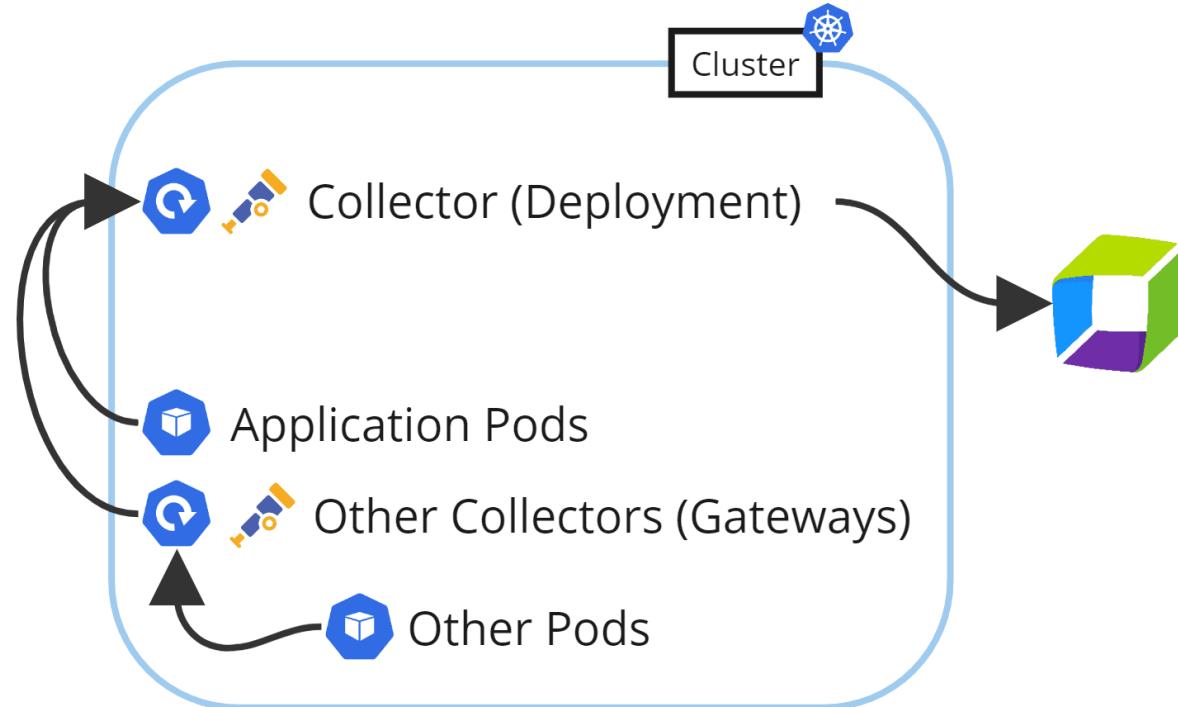
OTEL Collector – Agent

- As an agent, the Collector is deployed either with the application or (recommended) on the same host as the application. This Collector can receive telemetry data and enhance it with, for example, tags or infrastructure information.



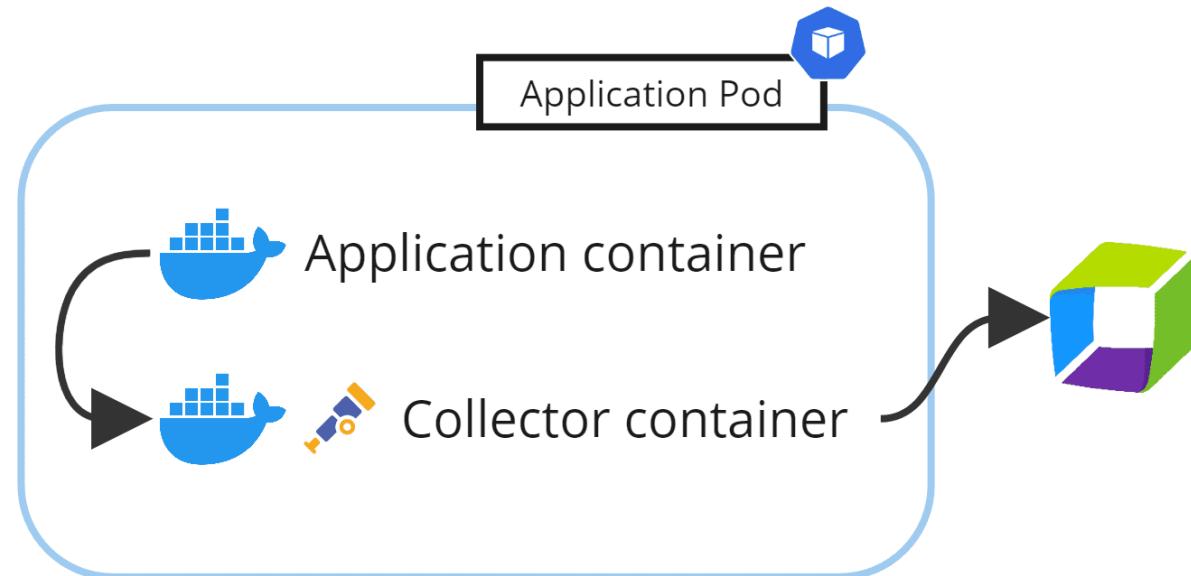
OTEL Collector – Gateway

- As a gateway, one or more Collector instances can be deployed as standalone services. This Collector can be deployed additionally, for example, per cluster, region, or data center. A load balancer can help scale the independently operating Collector instances.



OTEL Collector – Sidecar

- In Kubernetes clusters, you can use a variant of the agent mode and deploy the Collector as a sidecar to your application pods.
- You need to use the [OpenTelemetry operator](#) for this deployment option. Be aware that using the Collector as a sidecar might increase resource consumption.



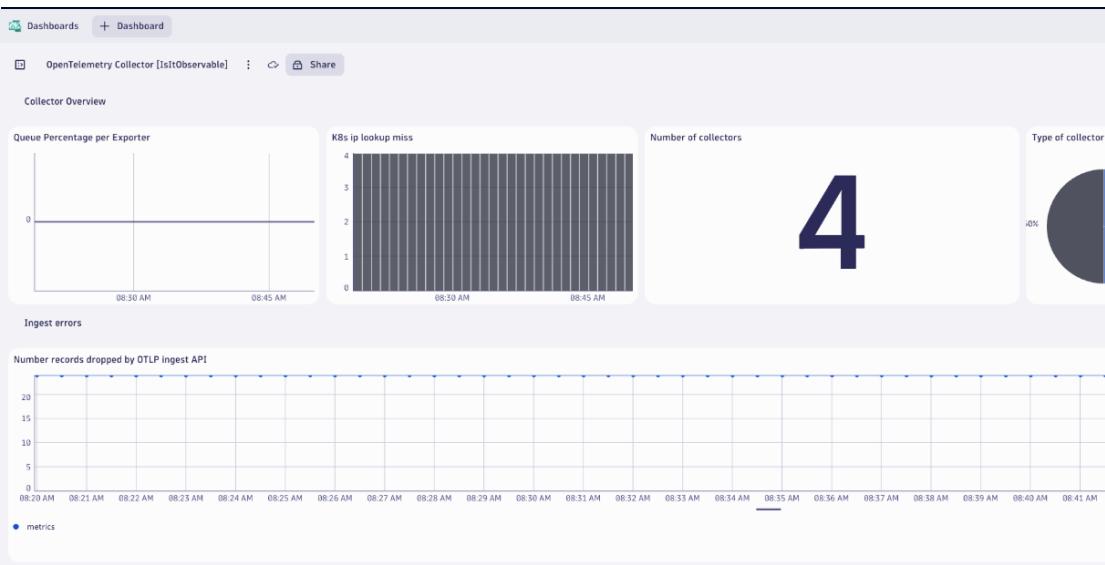
OTEL Collector – Use Cases

- <https://docs.dynatrace.com/docs/extend-dynatrace/opentelemetry/collector/use-cases>

Use cases			
 Batching Configure the OpenTelemetry Collector to send data in batches to the Dynatrace backend.	 FluentD Configure the OpenTelemetry Collector to ingest data from FluentD.	 gRPC to HTTP Configure the OpenTelemetry Collector to transform a gRPC OTLP request to HTTP.	 Histogram summaries Configure the OpenTelemetry Collector to compute bucket summaries for histogram metrics.
 Jaeger Configure the OpenTelemetry Collector to ingest and transform Jaeger data into Dynatrace.	 Kubernetes Enrichment Configure the OpenTelemetry Collector to enrich OTLP requests with Kubernetes data.	 Log files Configure the OpenTelemetry Collector to ingest log files.	 Memory Limitation Configure the OpenTelemetry Collector to respect memory limits and not use excessive system resources.
 Multiple backends Configure the OpenTelemetry Collector to export to multiple backends.	 Prometheus Configure the OpenTelemetry Collector to scrape your Prometheus data.	 Sampling Configure the OpenTelemetry Collector to sample distributed traces.	 StatsD Configure the OpenTelemetry Collector to ingest StatsD data.
 Syslog Configure the OpenTelemetry Collector to ingest syslog data.	 Transforming and filtering Configure the OpenTelemetry Collector to add, transform, and drop OpenTelemetry data.	 Zipkin Configure the OpenTelemetry Collector to ingest and transform Zipkin data into Dynatrace.	

OTEL Collector Monitoring

- OTEL collector provides health metrics and logs
- Scrape metrics via Prometheus exporter and send to Dynatrace



CONFIDENTIAL

Enrichment via OTEL collector – Kubernetes metadata

- **k8sattributes processor provides Kubernetes metadata**

The screenshot shows the Jaeger UI interface. On the left, there's a tree view of service spans, with a specific node for 'oteldemo.CheckoutService/PlaceOrder' highlighted. To the right, the main panel displays the details for this span. It includes tabs for Summary, Timing, Events (5), Links (0), and Code level. The Summary tab is active, showing the following attributes:

Attribute	Value
app.order.amount	1818.0
app.order.id	cb4d827d-3563-11ef-8b6e-c660b65cb5f8
app.order.items.count	1
app.shipping.amount	799.0
app.shipping.tracking.id	f50b62b7-2450-4aad-8208-c923951a00fa
app.user.currency	USD
app.user.id	cb452bda-3563-11ef-9ce5-0aa4f29da450
net.sock.peer.addr	127.0.0.6
net.sock.peer.port	59343
rpc.grpc.status_code	0
rpc.method	PlaceOrder
rpc.service	oteldemo.CheckoutService
rpc.system	grpc

Below the attributes, there's a section for 'Resource attributes' which lists various Kubernetes metadata fields. A red box highlights the 'k8s.' prefix on several of these fields:

Attribute	Value
k8s.name	astronomy-shop-checkoutservice-b6ccc7778-p96s9
k8s.deployment.name	astronomy-shop-otelcol
k8s.namespace.name	astronomy-shop
k8s.node.name	gke-tpc-0626-k8s-otel-col-01-default-pool-00f51967-vx77
k8s.pod.ip	127.0.0.6
k8s.pod.name	astronomy-shop-otelcol-dd657cc45-jsxrm
k8s.pod.uid	ef1877e6-3f53-4b04-8f97-036a7b46e814
os.description	Alpine Linux 3.20.0 (Linux astronomy-shop-checkoutservice-b6ccc7778-p96s9 6.1.75+ #1 SMP PREEMPT_DYNAMIC Sat Mar 30 14:38:17 UTC 2024 x86_64)
os.type	linux
process.command_args	./checkoutservice
process.executable.name	checkoutservice
process.executable.path	/usr/src/app/checkoutservice
process.owner	root
process.pid	1
process.runtime.description	go version go1.22.4 linux/amd64

Enrichment via OTEL collector – Cloud metadata

- **resourcedetection processor provides cloud metadata**

The screenshot shows the CloudWatch Metrics Insights interface. On the left, there's a search bar and a tree view of log groups and metrics. A specific metric named "oteldemo.CheckoutService/PlaceOrder" is selected and expanded. This selection is highlighted with a light blue box. The main pane displays the enriched log data for this metric, organized into sections: Attributes and Resource attributes.

Attributes

app.order.amount	844.0
app.order.id	95ff552-3566-11ef-8b6e-c660b65cb5f8
app.order.items.count	1
app.shipping.amount	144.0
app.shipping.tracking.id	44fc584b-35d5-4a81-8c79-e921913abdc6
app.user.currency	USD
app.user.id	95ea544-3566-11ef-9ce5-0aa4f29da450
net.sock.peer.addr	127.0.0.6
net.sock.peer.port	59343
rpc.grpc.status_code	0
rpc.method	PlaceOrder
rpc.service	oteldemo.CheckoutService
rpc.system	grpc

Resource attributes

cloud.account.id	
cloud.availability_zone	us-central1-c
cloud.platform	gcp_kubernetes_engine
cloud.provider	gcp
host.id	7481437109762512772
host.name	astronomy-shop-checkoutservice-b6ccc778-p96s9
k8s.cluster.name	tpc-0626-k8s-otel-0tly
k8s.deployment.name	astronomy-shop-otelcol
k8s.namespace.name	astronomy-shop
k8s.node.name	gke(tpc-0626-k8s-otel-0tly-default-pool-00f51967-vx77)
k8s.pod.ip	127.0.0.6
k8s.pod.name	astronomy-shop-otelcol-dd657cc45-jsxrm
k8s.pod.uid	ef187e6-3f53-4b04-8f97-036a7b46e814
os.description	Alpine Linux 3.20.0 (Linux astronomy-shop-checkoutservice-b6ccc778-p96s9 6.1.75+ #1 SMP PREEMPT_DYNAMIC Sat Mar 30 14:38:17 UTC 2024 x86_64)
os.type	linux
process.command_args	/checkoutservice
process.executable.name	checkoutservice
process.executable.path	/usr/src/app/checkoutservice
process.owner	root
process.pid	1

Custom Attributes

Search name, URL, SQL, attribute...

POST
loadgenerator

ingress
frontendproxy

router frontend egress
frontendproxy

POST
frontend

executing api route (pages) /api/checkout
frontend

grpc.oteldemo.CheckoutService/PlaceOrder
frontend

oteldemo.CheckoutService/PlaceOrder
checkoutservice

prepareOrderItemsAndShippingQuoteFromCart
checkoutservice

oteldemo.CartService/GetCart
checkoutservice

ASP .NET POST /oteldemo.CartService/GetCart
cartservice

ASP .NET GET
cartservice

oteldemo.ProductCatalogService/GetProduct
checkoutservice

oteldemo.ProductCatalogService/GetProduct
productcatalogservice

oteldemo.CurrencyService/Convert
checkoutservice

oteldemo.ShippingService/GetQuote
checkoutservice

oteldemo.ShippingService/GetQuote
shippingservice

POST
shippingservice

PHP POST /getquote
quotesservice

[closure] PHP [closure] quotesservice

oteldemo.CheckoutService/PlaceOrder

Summary Timing Events (5) Links (0) Code level

Resource attributes

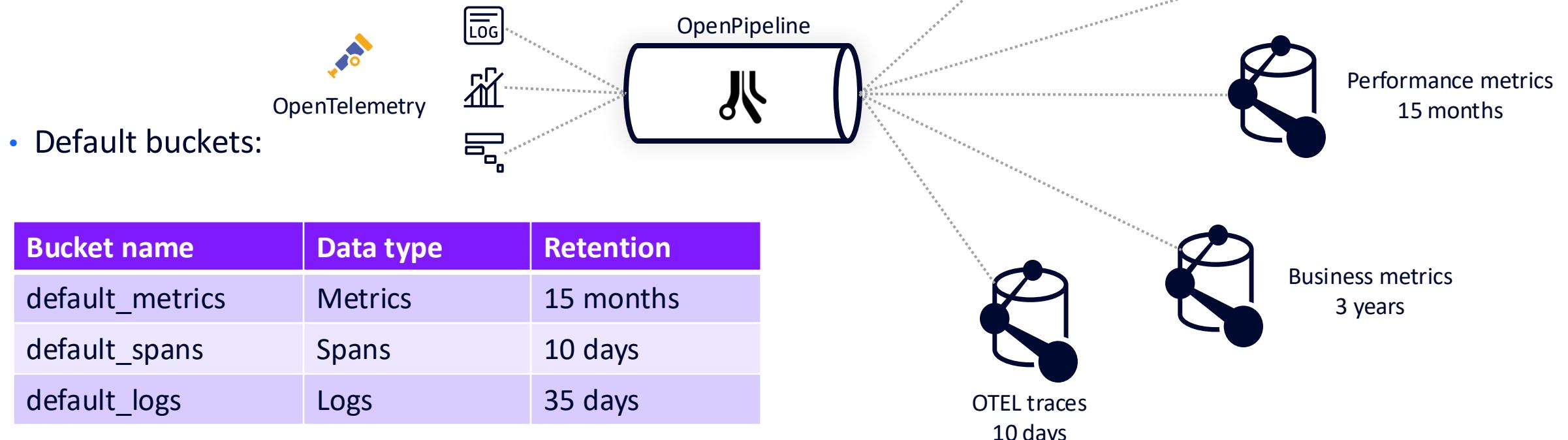
cloud.account.id	us-central1-c
cloud.availability_zone	gcp_kubernetes_engine
cloud.platform	gcp
cloud.provider	
dynatrace.otel.collector	dynatrace-traces
host.id	7481437109762312772
host.name	astronomy-shop-checkoutservice-b6ccc7778-p96s9
k8s.cluster.name	tpc-0626-k8s-otel-01ly
k8s.deployment.name	astronomy-shop-otelcol
k8s.namespace.name	astronomy-shop
k8s.node.name	gke-tpc-0626-k8s-otel-01-default-pool-00f51967-vx77
k8s.pod.name	astronomy-shop-otelcol-dd657cc45-jsxm
k8s.pod.uid	ef1877e6-3f53-4b04-8f97-036a7b46e814
os.description	Alpine Linux 3.20.0 (Linux astronomy-shop-checkoutservice-b6ccc7778-p96s9 6.1.75+ #1 SMP PREEMPT_DYNAMIC Sat Mar 30 14:38:17 UTC 2024 x86_64)
os.type	linux
process.command_args	/checkoutservice
process.executable.name	checkoutservice
process.executable.path	/usr/src/app/checkoutservice
process.owner	root
process.pid	1
process.runtime.description	go version go1.22.4 linux/amd64
process.runtime.name	go
process.runtime.version	go1.22.4
service.name	checkoutservice
service.namespace	tpc-0626-k8s-otel-01ly
service.version	1.10.0
telemetry.sdk.language	go
telemetry.sdk.name	opentelemetry
telemetry.sdk.version	1.27.0

Configure Attributes block-list

Dynatrace internal-use attributes

Data Retention

- OTEL data is stored in Grail buckets
- Each bucket stores one data type and defines retention period
- Custom buckets can be created (max 80 per environment)
- Data can be assigned to buckets via rules



Record Level Security



Bucket: audit

Retention: 3y

"All audit logs"

Bucket: infra

Retention: 3mo

“Business unit X infra logs”

Bucket: debug

Retention: 10d

"All debug logs"

She needs access to her business unit's infra logs, security events and debug data



Makes sure the Dynatrace monitoring components are deployed, updated and working fine.



When developing or troubleshooting services, I need to monitor performance and regression

Access control for OpenTelemetry data

- Policies can be defined on data record level
- Policy statement syntax:

```
ALLOW <permissions> WHERE <conditions>
```

- Conditions can be based on:
 - Specific attributes per data type
 - Security context attribute
- Example policy:
 - Team is only allowed to see data from Kubernetes namespace DEV

```
ALLOW storage:metrics:read WHERE storage:k8s.namespace = "DEV";  
ALLOW storage:spans:read WHERE storage:k8s.namespace = "DEV";  
ALLOW storage:logs:read WHERE storage:k8s.namespace = "DEV";
```

storage:spans:read

Grants permission to read records from the spans-table

conditions:

- `storage:k8s.namespace.name` - The name of the namespace that the pod is running in.
operators: `=`, `IN`, `startsWith`
- `storage:k8s.cluster.name` - The name of the cluster that the pod is running in.
operators: `=`, `IN`, `startsWith`
- `storage:host.name` - Name of the host.
operators: `=`, `IN`, `startsWith`
- `storage:dt.host_group.id` - Id of the host group.
operators: `=`, `IN`, `startsWith`
- `storage:dt.security_context` - Custom field for security context.
operators: `=`, `IN`, `startsWith`
- `storage:gcp.project.id` - Google Cloud Platform Project ID.
operators: `=`, `IN`, `startsWith`
- `storage:aws.account.id` - Amazon Web Services Account ID.
operators: `=`, `IN`, `startsWith`
- `storage:azure.subscription` - Azure subscription.
operators: `=`, `IN`, `startsWith`
- `storage:azure.resource.group` - Azure resource group.
operators: `=`, `IN`, `startsWith`

Security Context

- Dedicated attribute just for permissions
- Can be any string:
e.g. „region.department.team“
- Value has to be added on the source or during ingest

Policy details

Name*
Access self monitoring logs

Description
Optional

Policy statement

```
1 ALLOW storage:logs:read
2 WHERE storage:dt.security_context = "dsfm";
```

Settings > Log Monitoring > Security context

Settings

- Monitoring
- Cloud Automation
- Ownership
- Processes and containers
- Web and mobile monitoring
- Cloud and virtualization
- Server-side service monitoring
- Service Detection
- Log Monitoring
- Ingest pipeline
- Processing
- Security context**
- Custom attributes
- Events extraction
- Metrics extraction
- Bucket assignment

Log Security Context [In development]

Configure rules for security context. The first user-defined rule that matches will be executed. Learn more by visiting our [documentation](#).

Add rule

Filter by

Summary

MyApp logs

Rule name: MyApp logs

Matcher: matchesValue(dt.entity.process_group, "PROCESS_GROUP-8115A939B7280815")

See our documentation

Select value source type:

- Field: Value will be copied from field
- Literal: Constant literal will be used as value

Value source field: dt.entity.process_group

Value: dsfm

Literal value to be set: doc-test3



CLOUD DONE RIGHT