SE Training – Kubernetes OpenTelemetry

Training Recorded

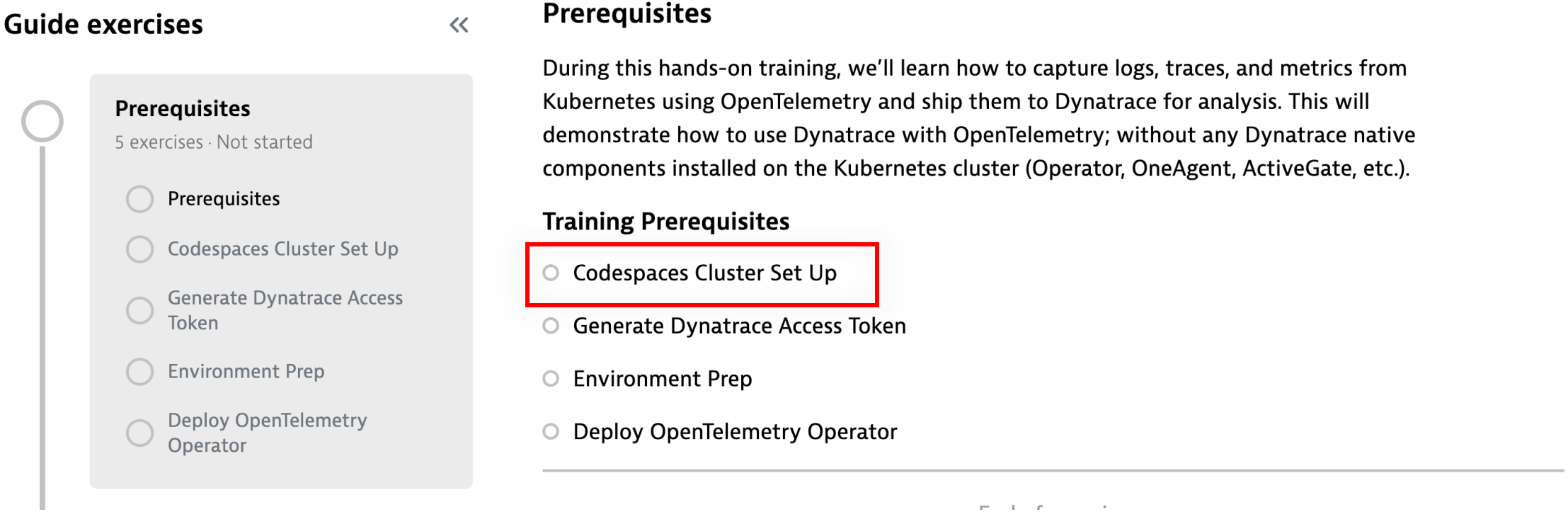
Warning This training is being recorded and will be available for future on-demand consumption. Please maintain professionalism at all times. Thank you.

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!



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

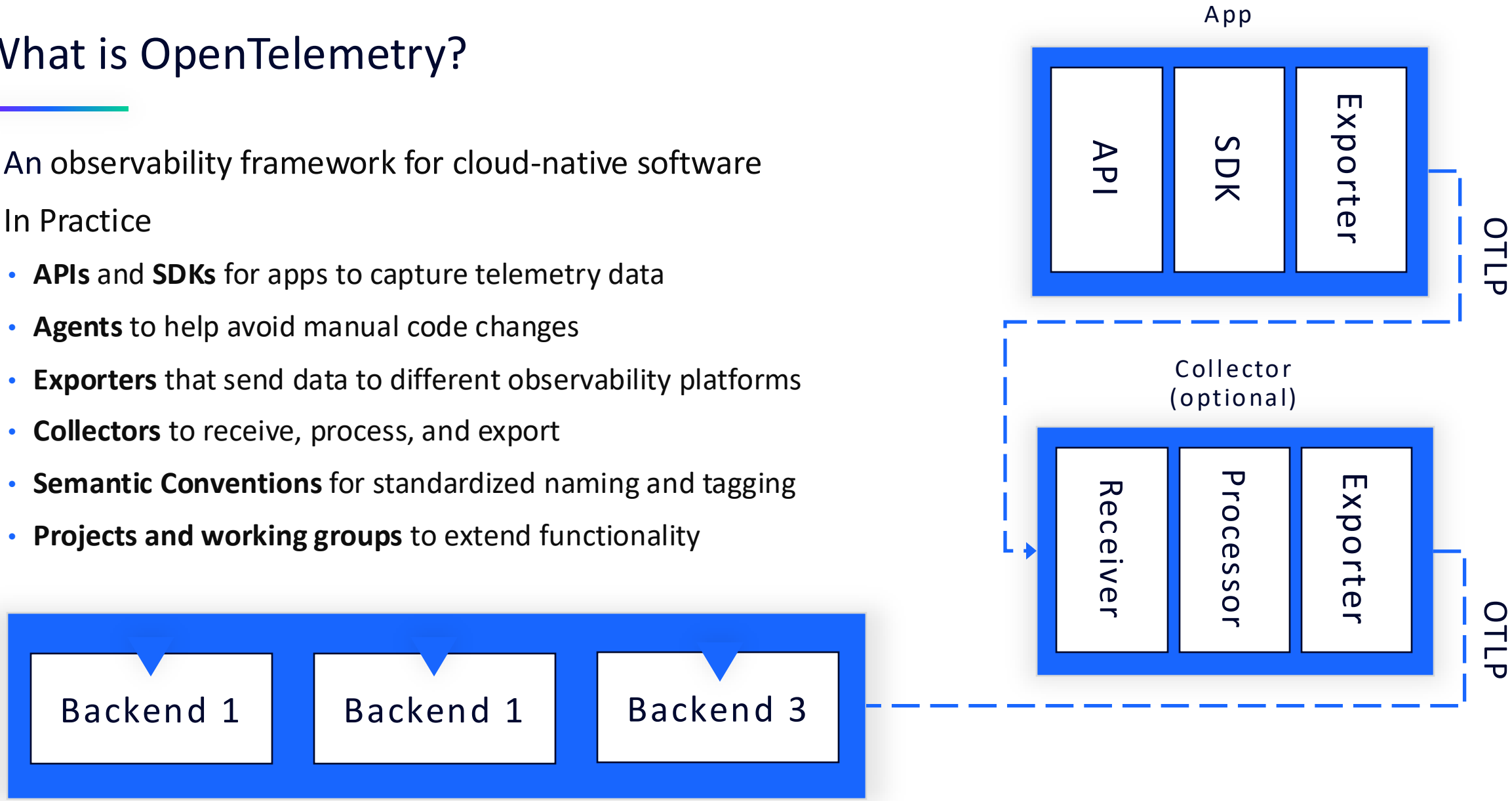
MATURITY

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

* OpenLogic State of Open-Source Survey 2023

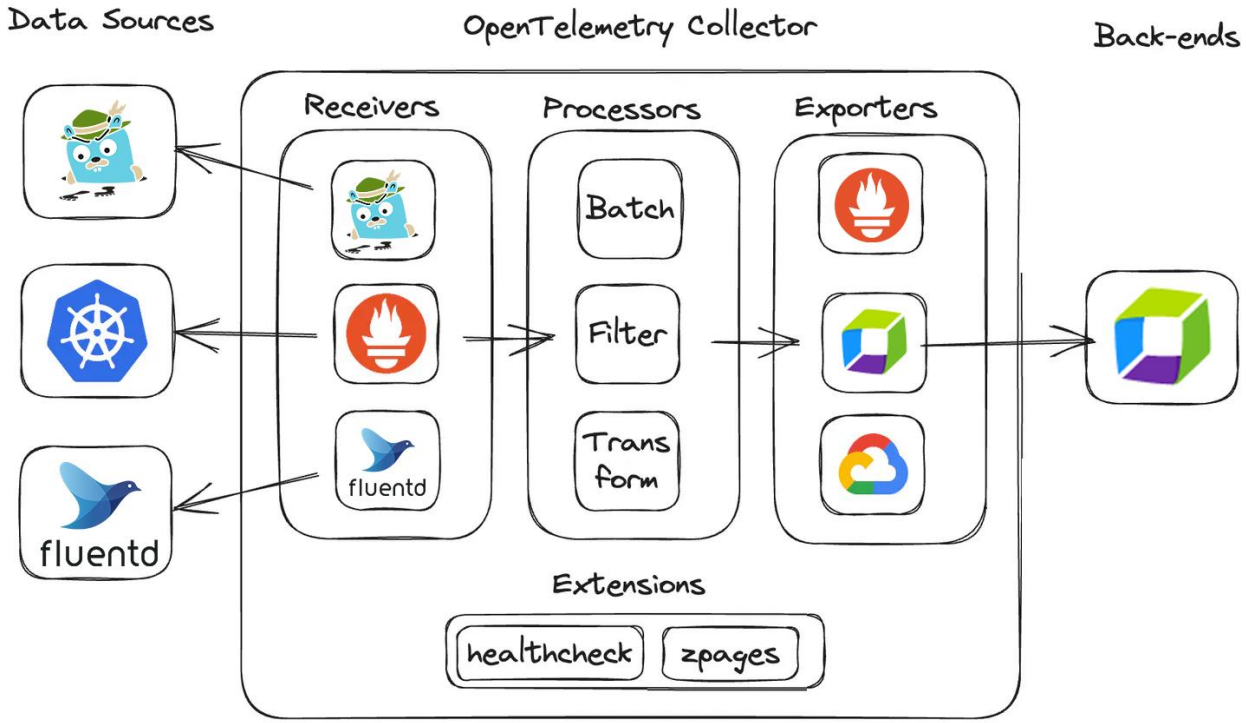
What is OpenTelemetry?

- 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

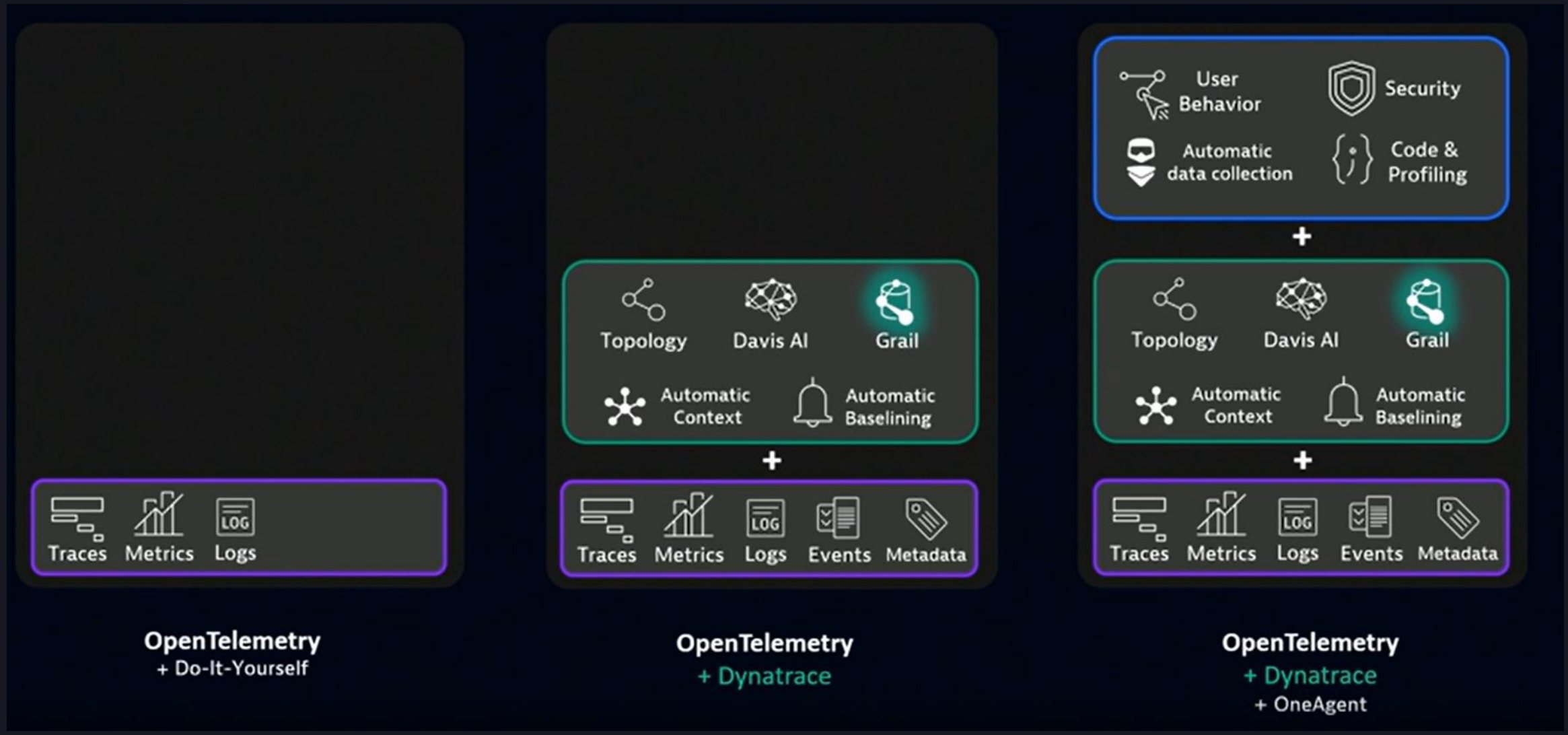


OpenTelemetry: Receivers, Processors, and Exporters

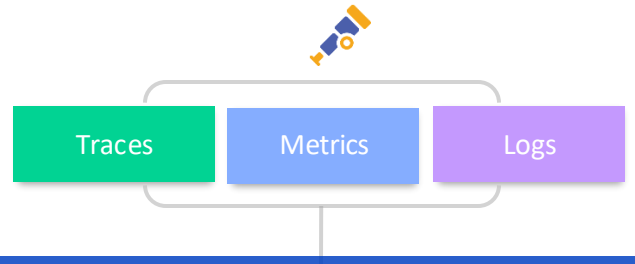
- **Use receivers** to collect telemetry data and manage data streams
- **Use processors** to offload telemetry processing from applications such as batch, filter, transform, etc.
- **Use exporters** to connect and send data to backends



OpenTelemetry + Dynatrace = Faster & Smarter



OpenTelemetry with Dynatrace



Root cause anomaly detection	Davis AI	User behavior	Experience insights & pre-emptive action
Auto-created relationships	Topology	Application security	Find and fix app vulnerabilities
Self-adjusting threshold calculation	Auto-Baselining	Zero overhead code-profiling	Identify hotspots & performance issues
Entity-centric data visualizations	Data-in-Context	Effortless maintenance-free rollout	Enterprise-wide push button deploy & mngmt
Data harmonization w/ oTel semantics	Grail	Memory and thread analytics	Utilization, contention, and concurrency insight
		Best-of-breed sensor integration	Complete coverage with oTel and OneAgent

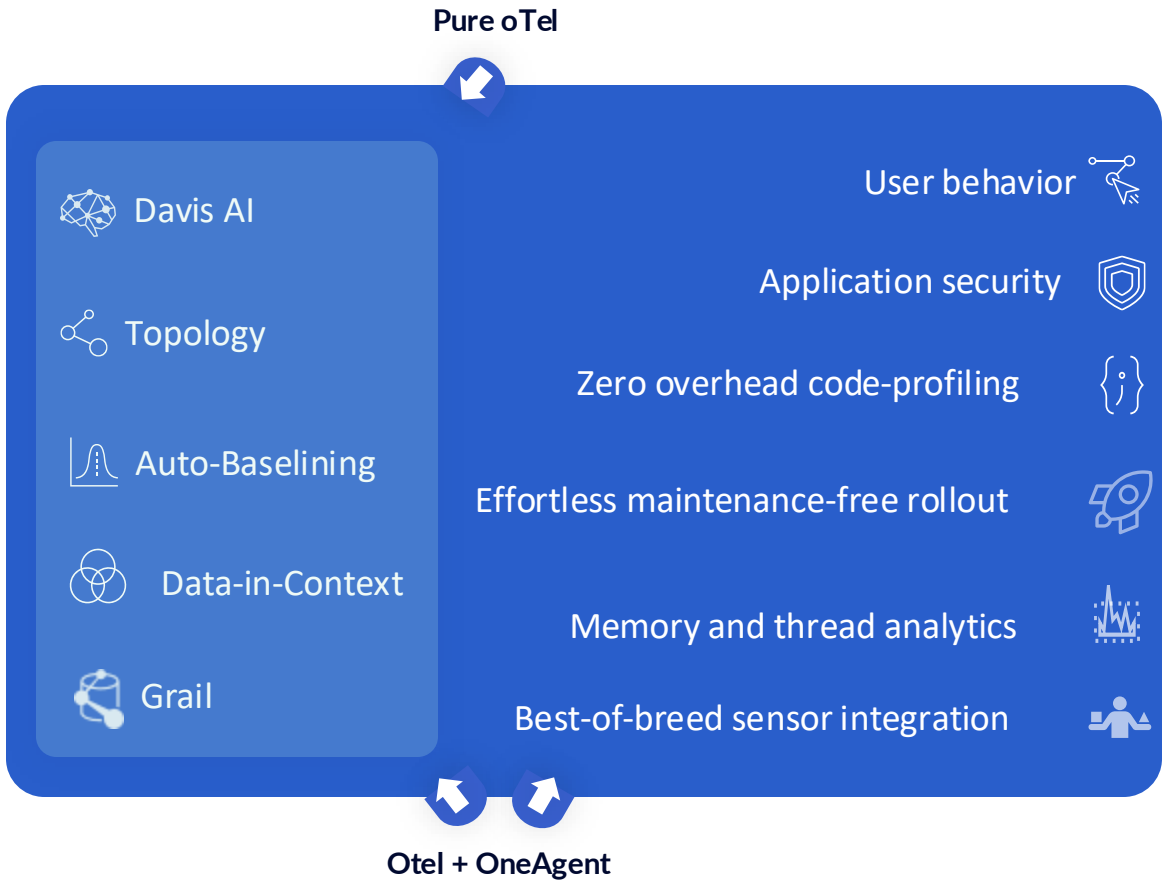
Powerful complements

Pure oTel

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

oTel + OneAgent

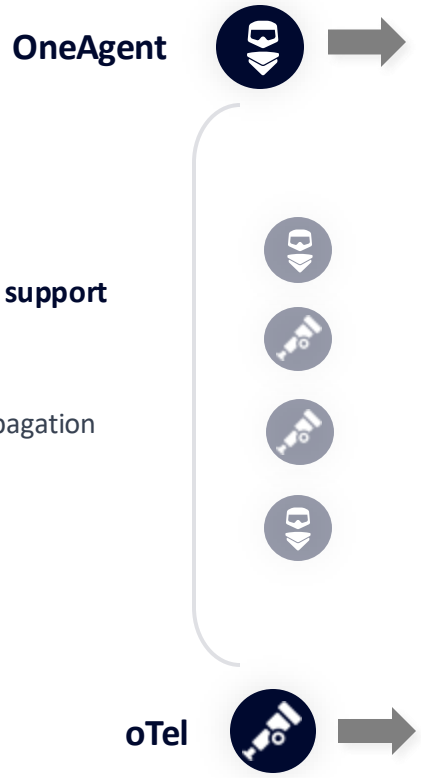
OpenTelemetry enhanced by 15 years of Dynatrace OneAgent innovation.

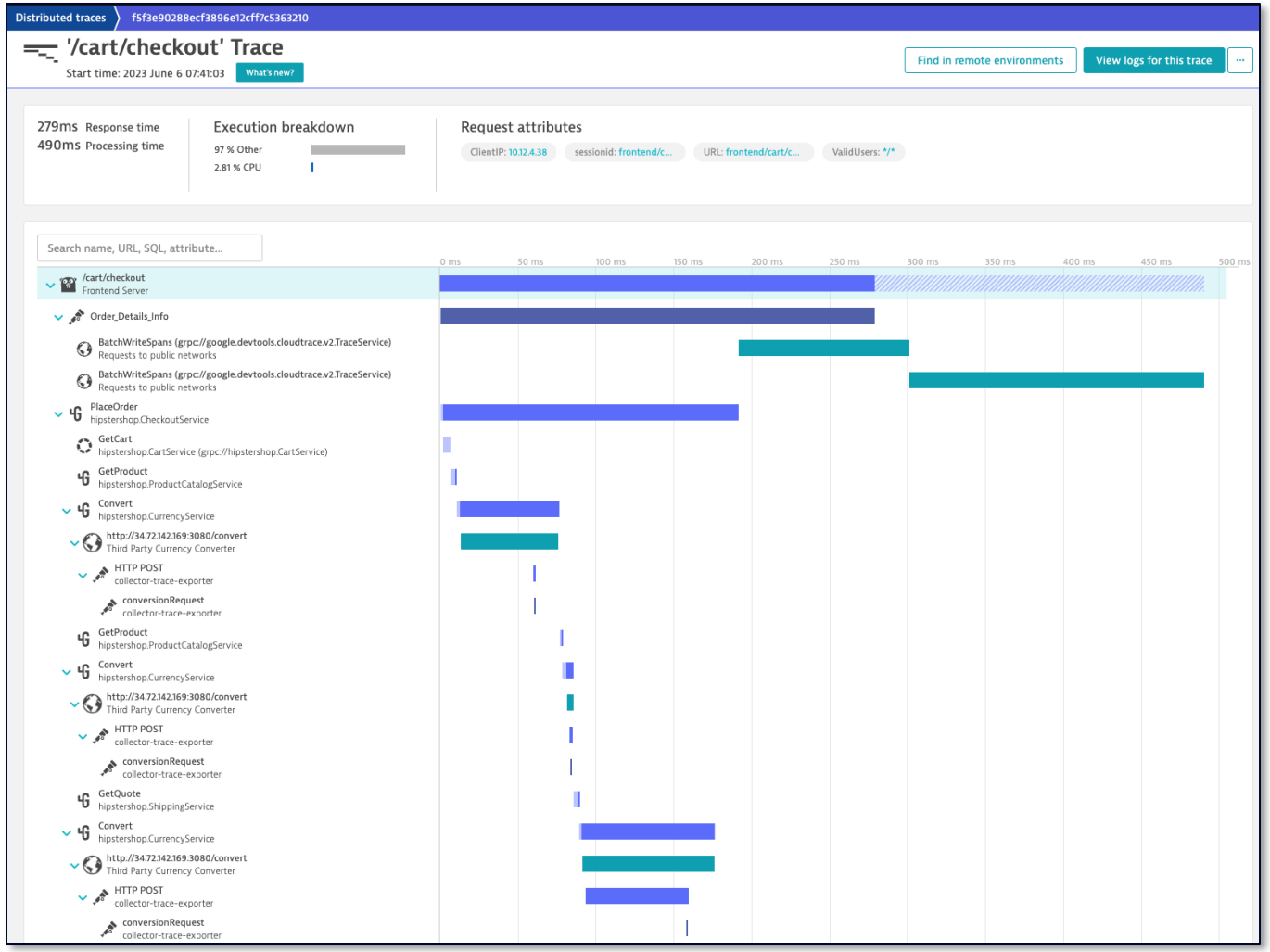


Best of Breed signal integration

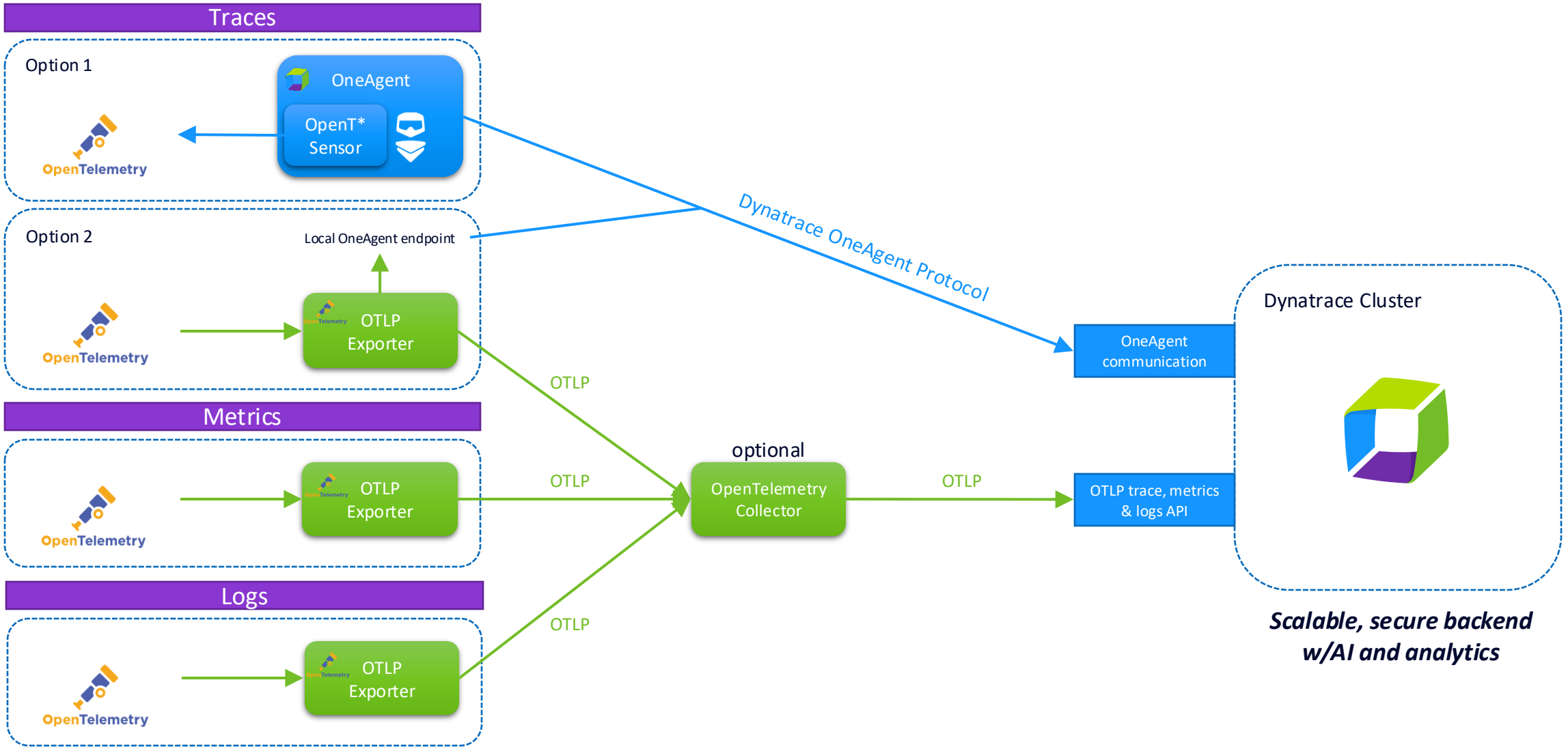
Enhanced OpenTelemetry support

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





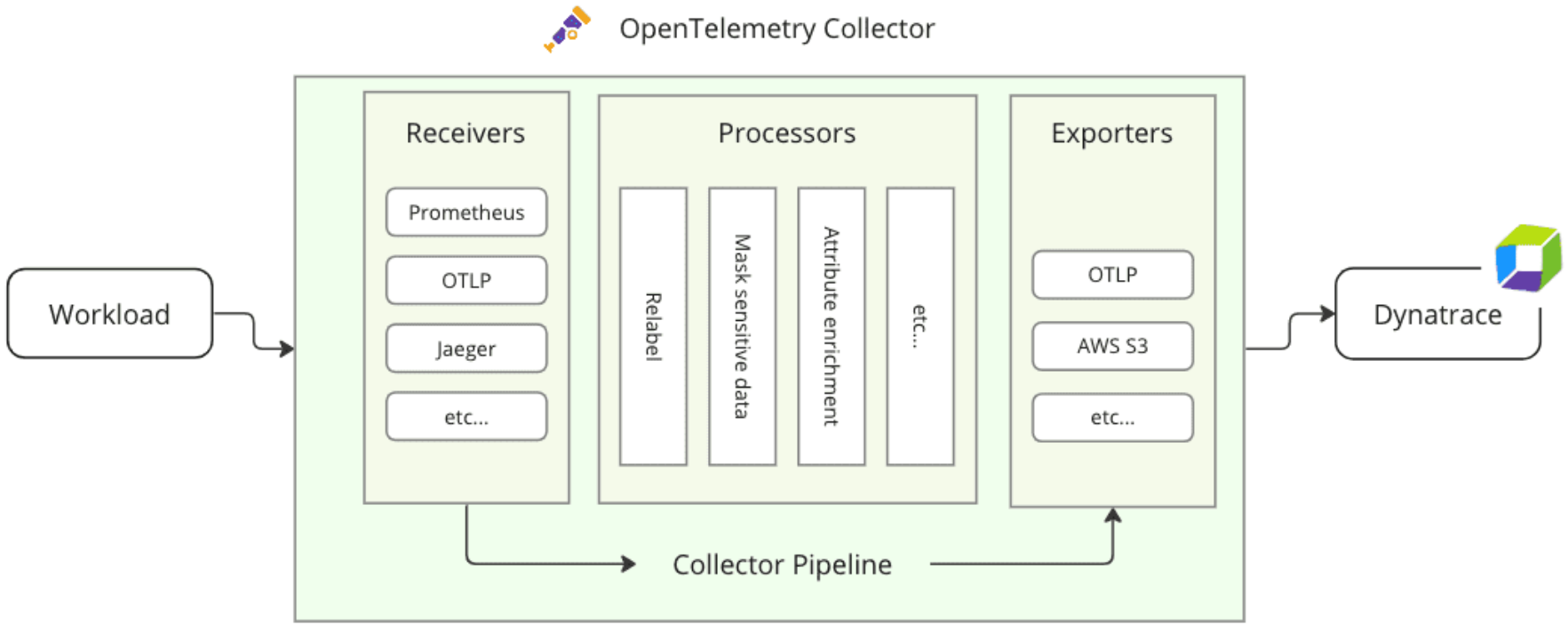
Different ways to send OpenTelemetry to Dynatrace



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

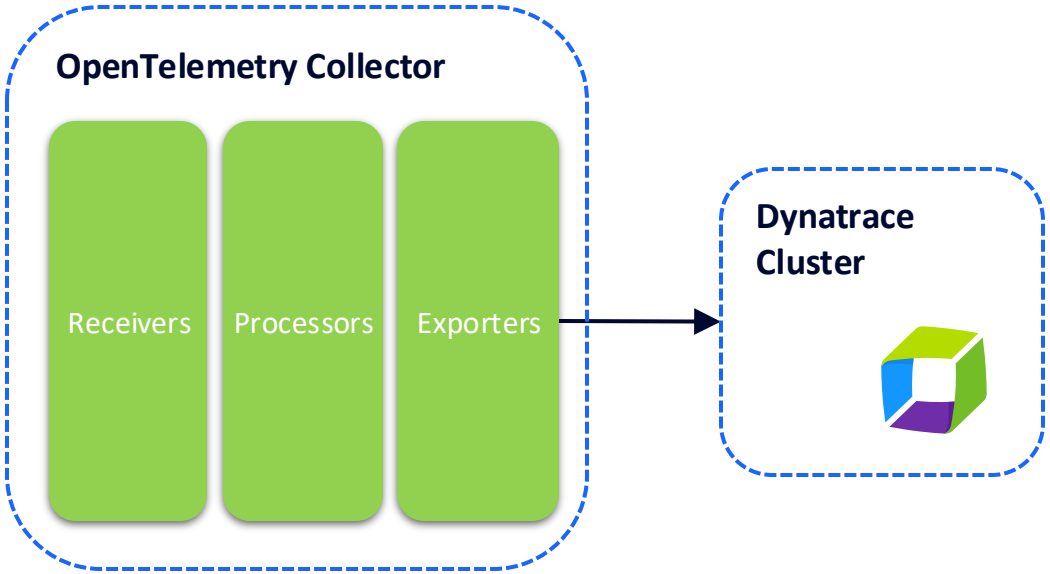


Unleashing the Power of Dynatrace Enterprise Collector for Unrivaled Support, Stability, and Speed

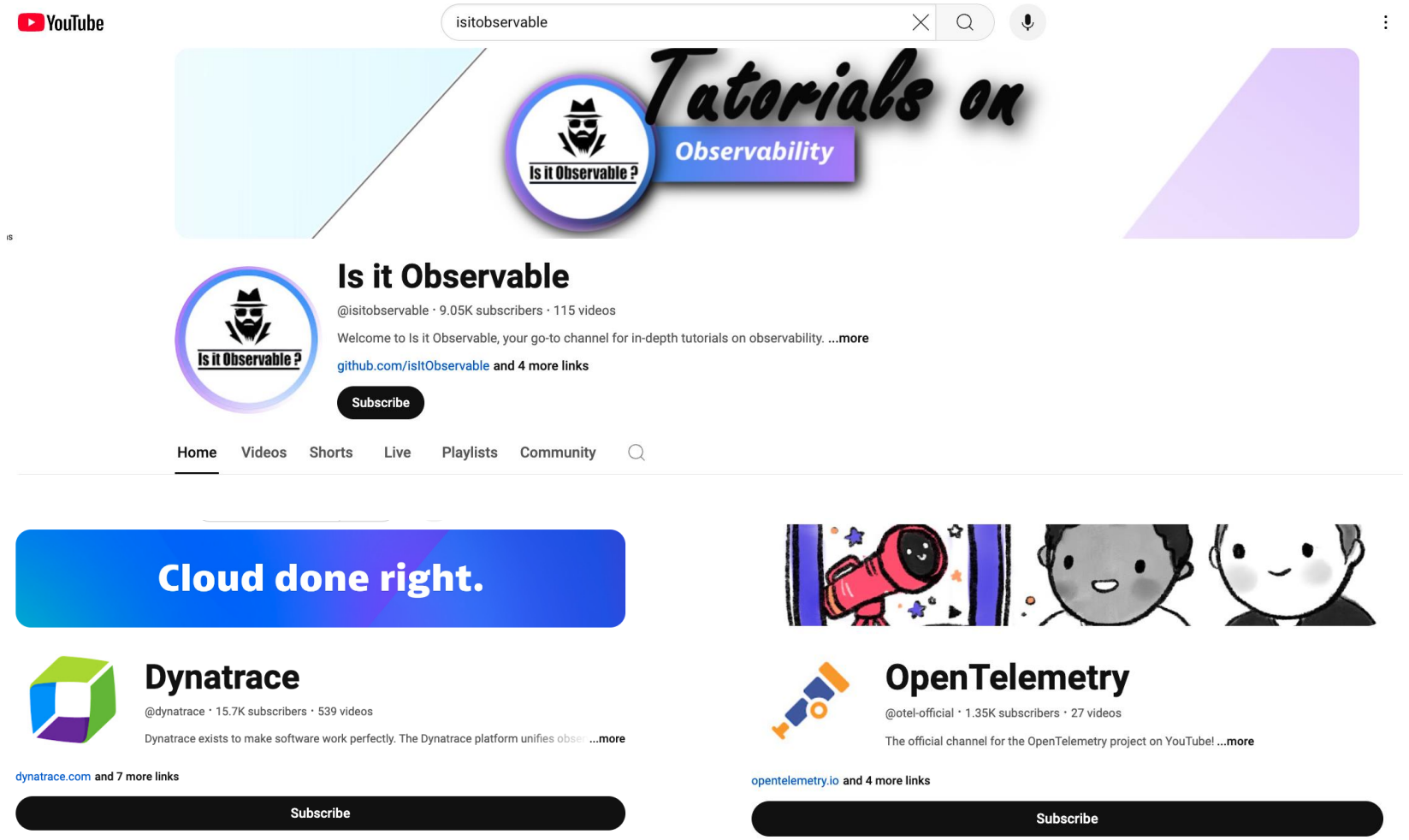
	Dynatrace Collector Distribution	Open Source Collector Distribution
Support	Dynatrace ONE Standard or Premium Support	Limited support
Collector Components	Stable and verified collector components to maximize service reliability	All collector components, including unstable and untested versions
Sample 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



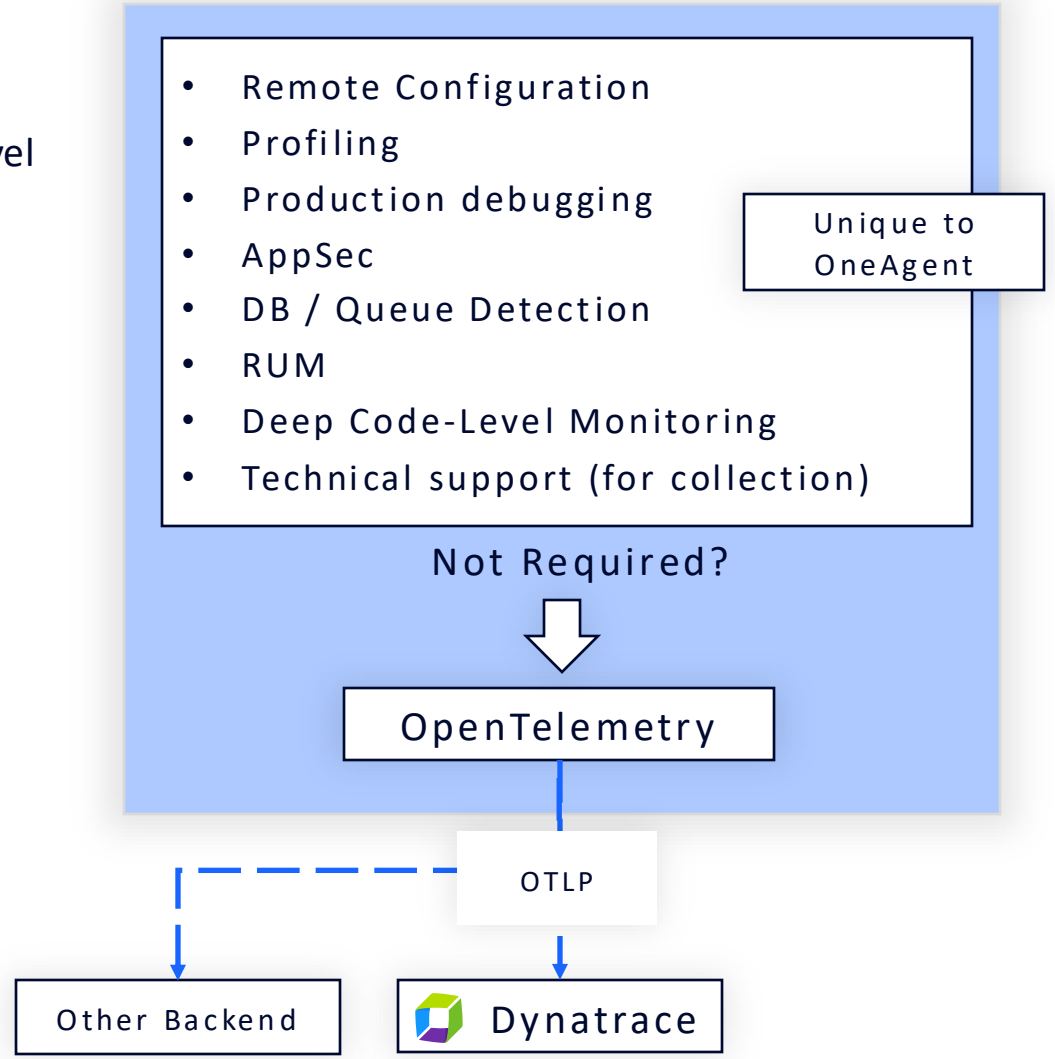
Questions?

Appendix

Additional OpenTelemetry + Dynatrace Information

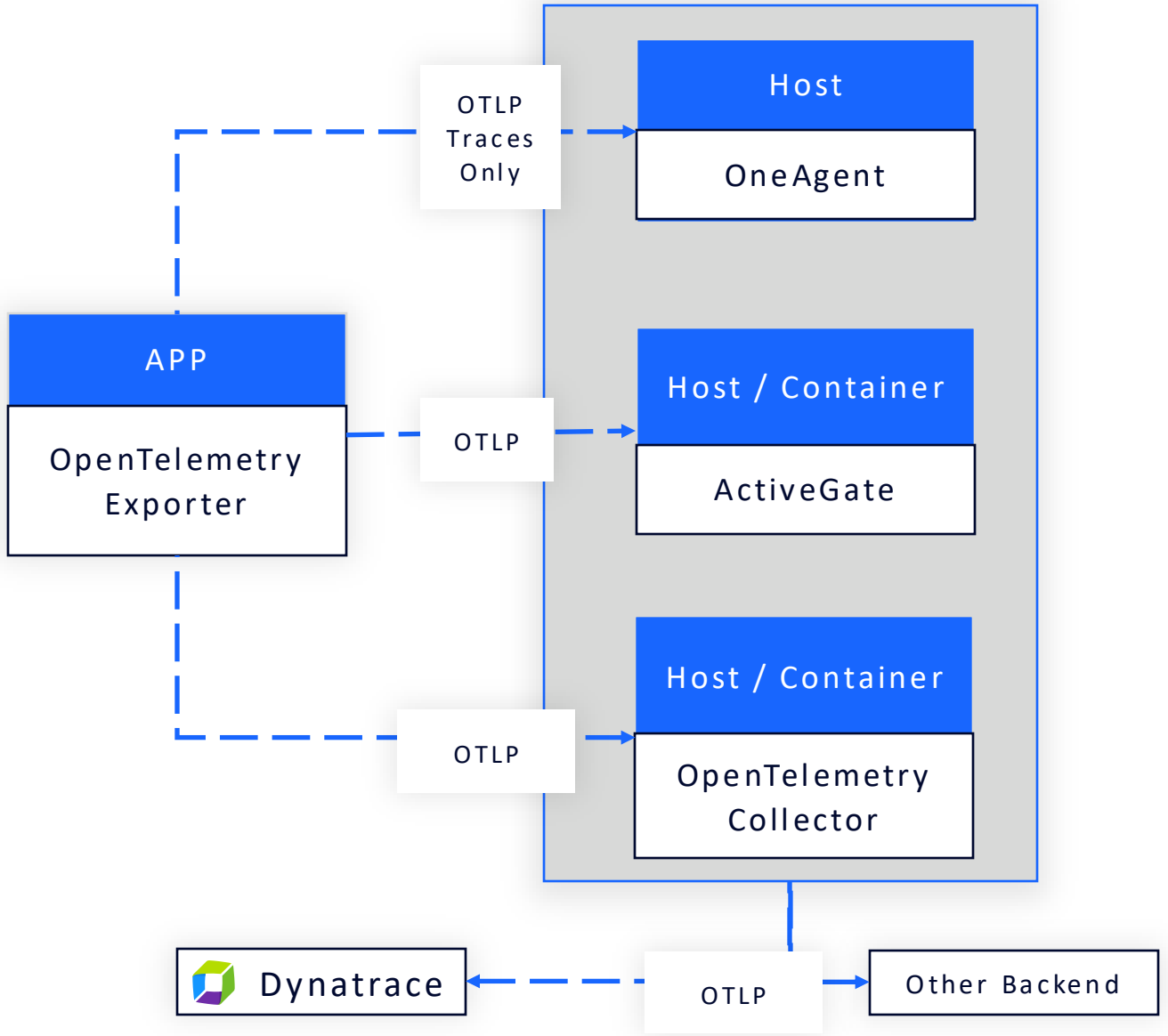
Instrument Services with OneAgent or OpenTelemetry

- 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



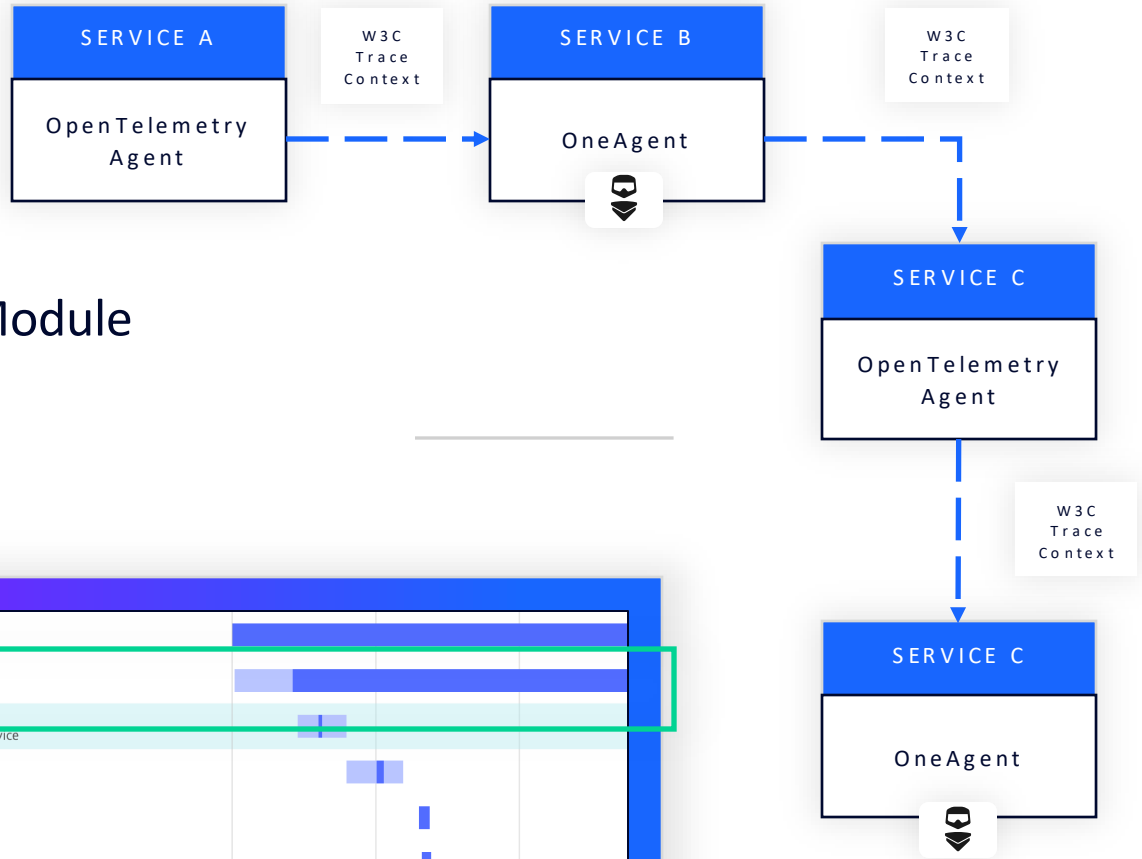
OpenTelemetry Collection Options

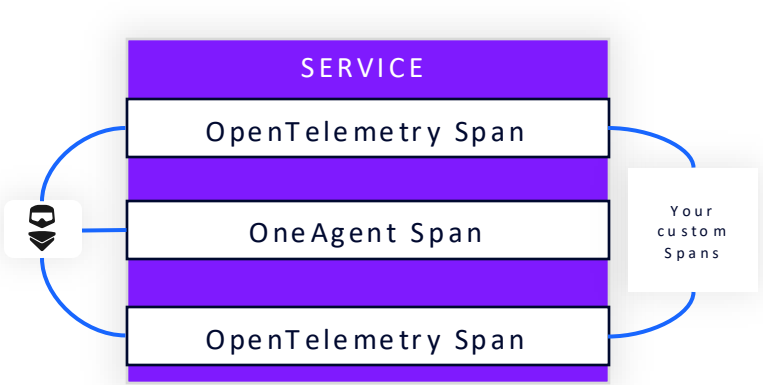
- Host with OneAgents as OpenTelemetry Collector
  - When OneAgent is deployed to host
  - Collects traces with licensing benefits
- Containerized or routable ActiveGates
  - When OneAgent is not available on host
  - Supports all OTLP signals at /v2/otlp/
- OpenTelemetry Collector
  - Fully supported 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





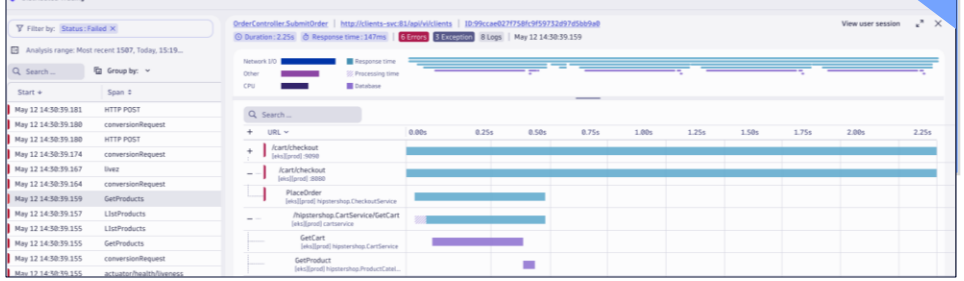
Native OpenTelemetry Across the Dynatrace Platform

- 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

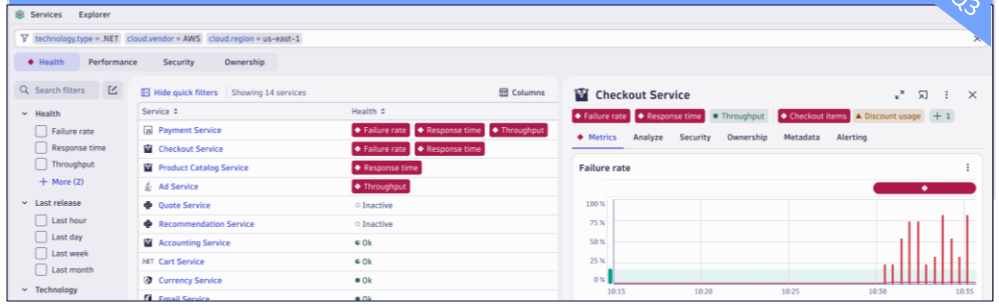
DQL Traces Powered by Grail — Q2

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

Distributed Traces — Q2

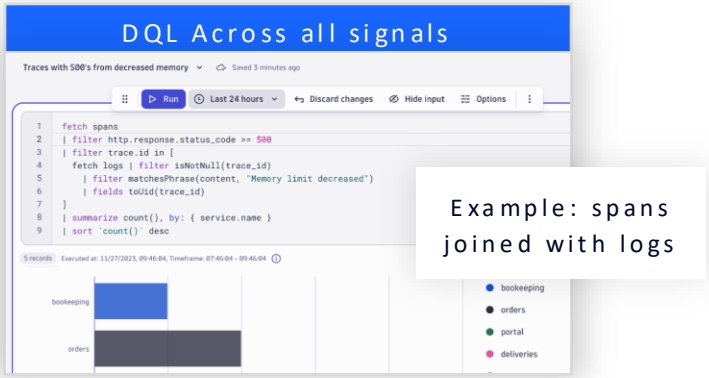


Services — Q3

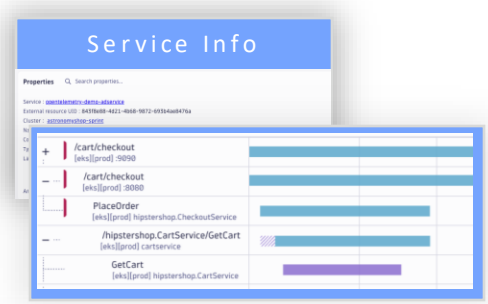


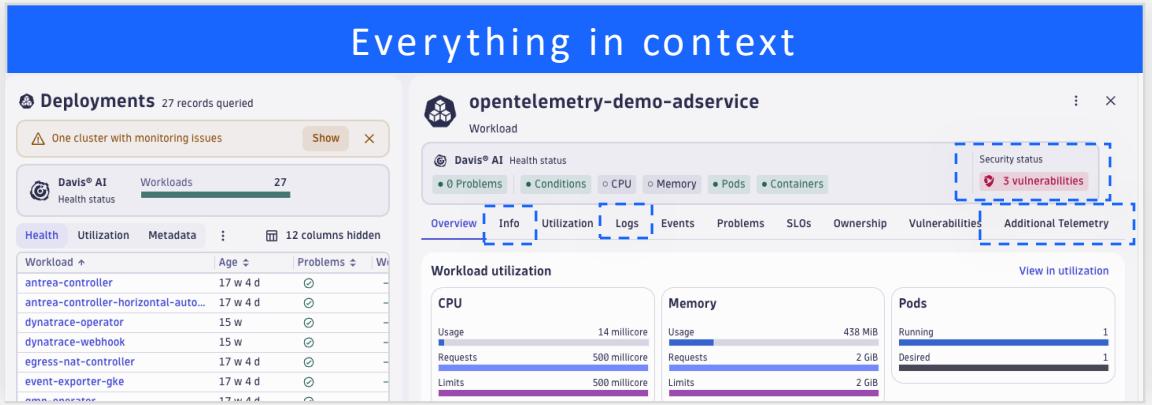
Unified Observability in a Single Platform

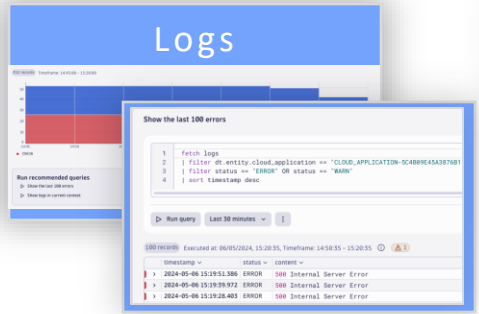
- Everything in context (example: Kubernetes)
  - Workload info opens OTel Services and Distributed Tracing experiences
  - High-level and detailed OTel log viewer
  - Additional Telemetry with associated (OTel/Prometheus) metrics
  - 3rd party and code-level security vulnerabilities
- DQL Across all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

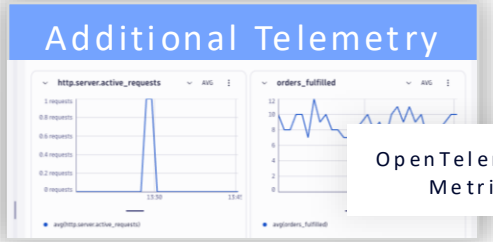


Example: spans joined with logs





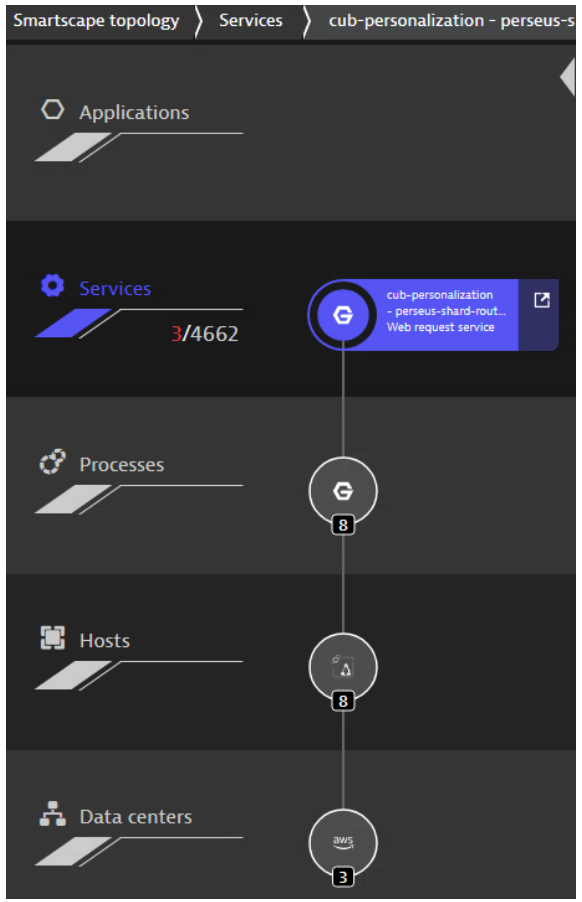




OpenTelemetry Metric

Topology – Entity Model

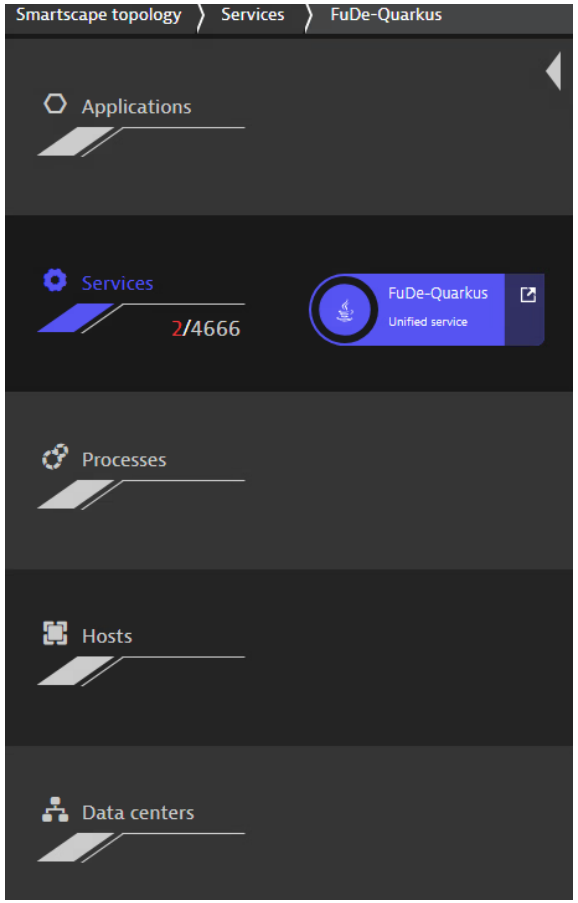
OTEL data
collected via OneAgent



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

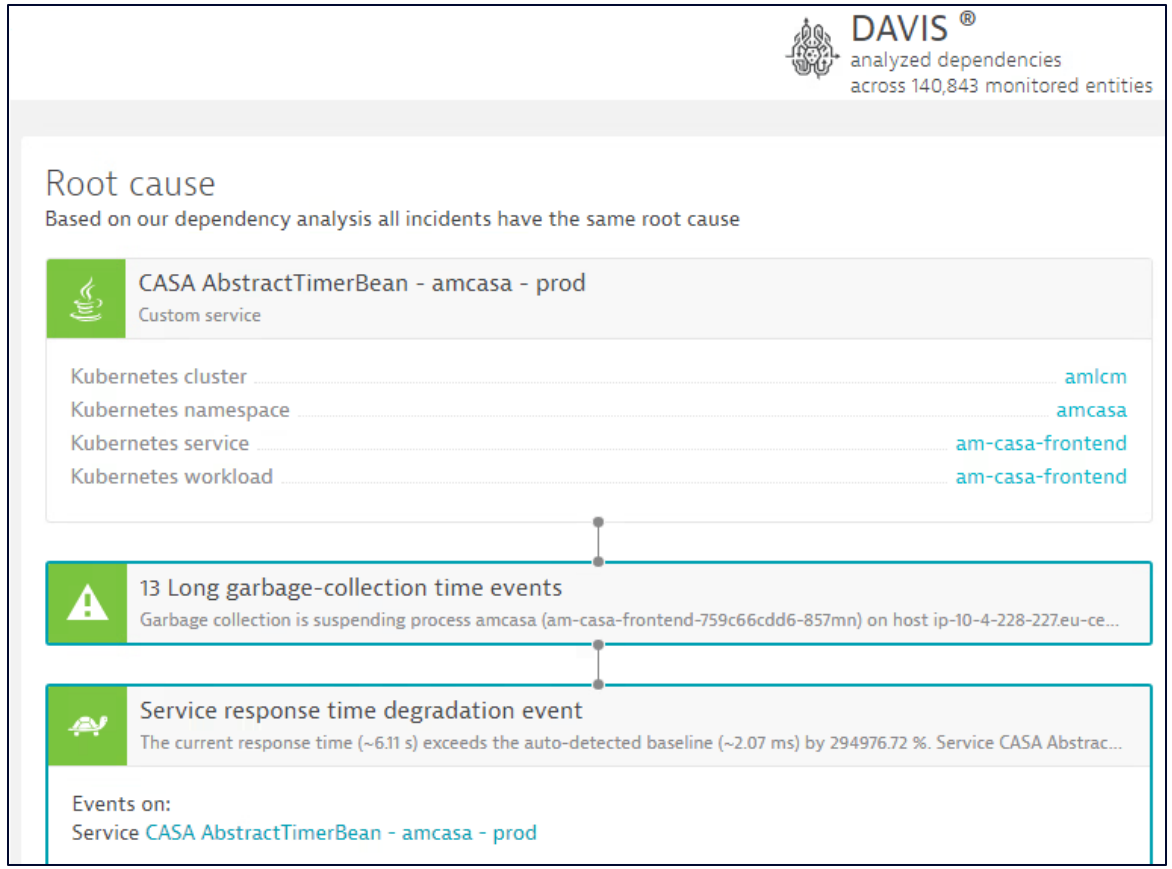
Topology – No connection to underlying process / hosts

- 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

causes

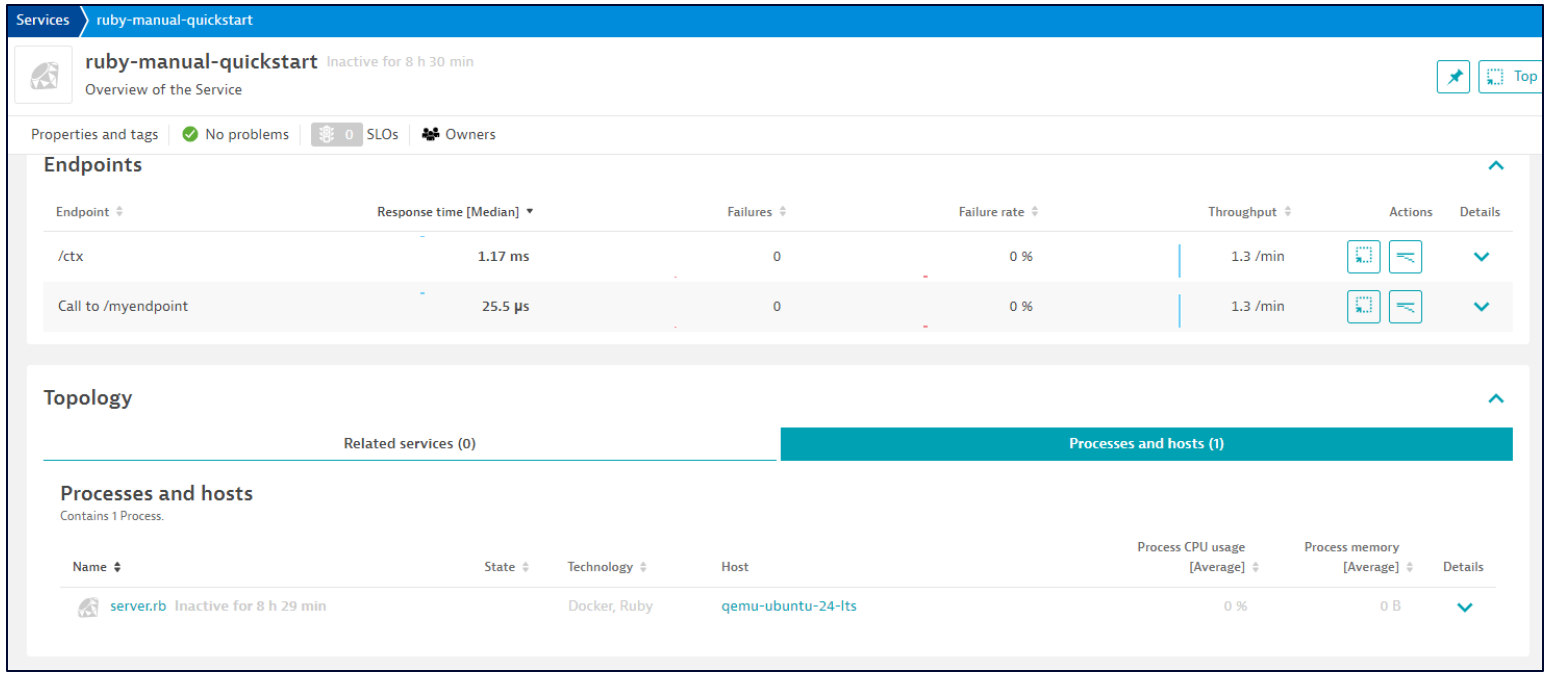
Problem on the Process level

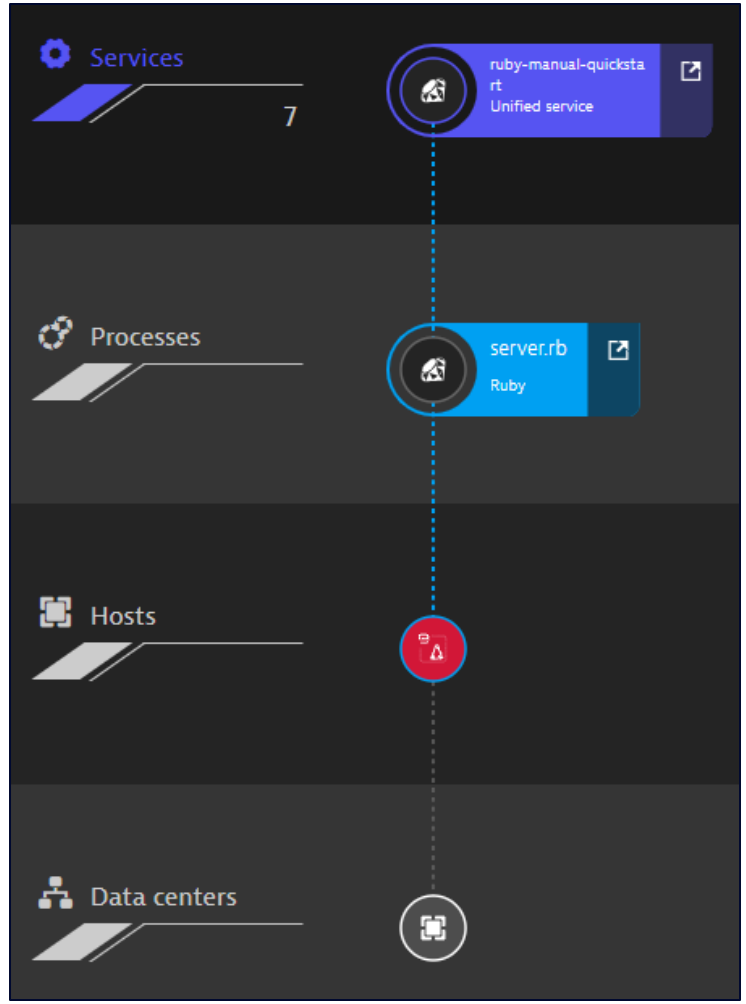
Slowness on the Service level



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





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

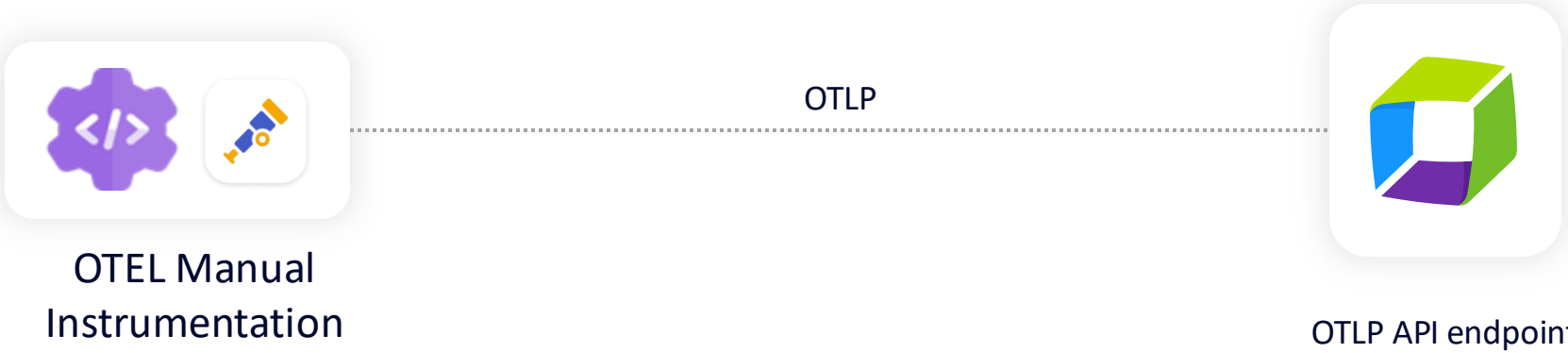
- Dynatrace API – OTLP endpoints
  - `https://{environment-id}.live.dynatrace.com/api/v2/otlp/v1/metrics`
  - `https://{environment-id}.live.dynatrace.com/api/v2/otlp/v1/traces`
  - `https://{environment-id}.live.dynatrace.com/api/v2/otlp/v1/logs`

- 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

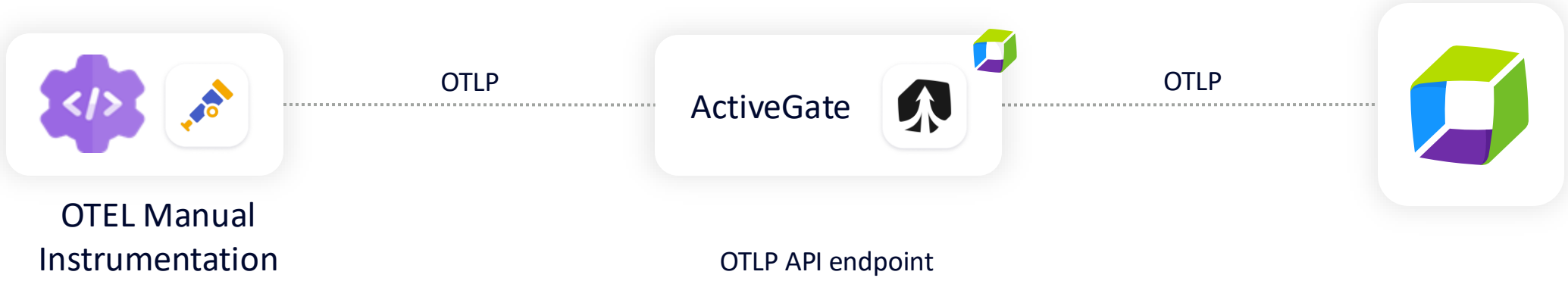
Metrics

- Metrics sent directly to Dynatrace SaaS cluster



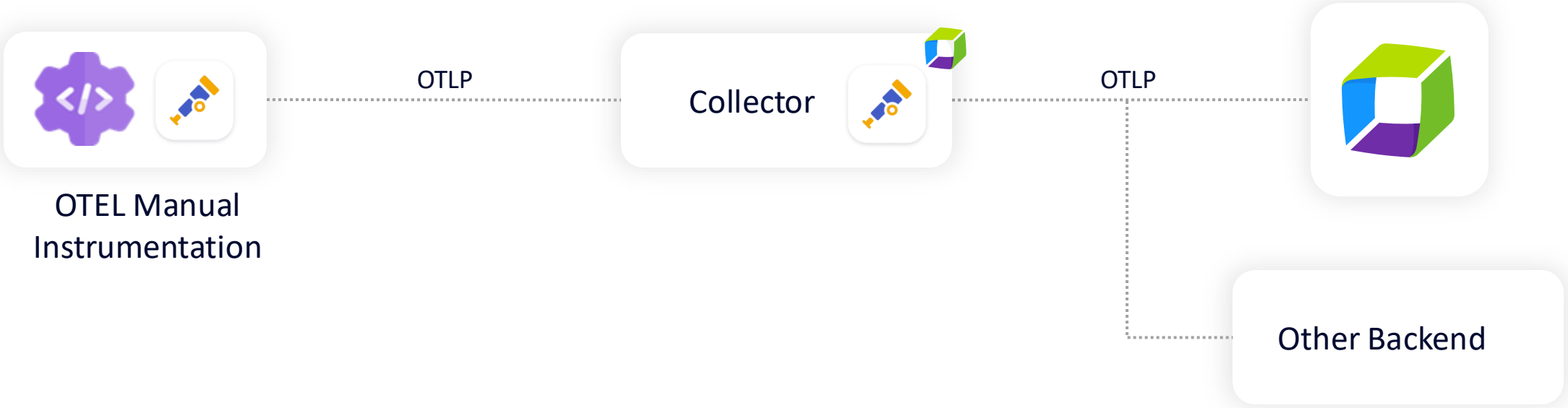
Metrics

- Metrics sent via ActiveGate



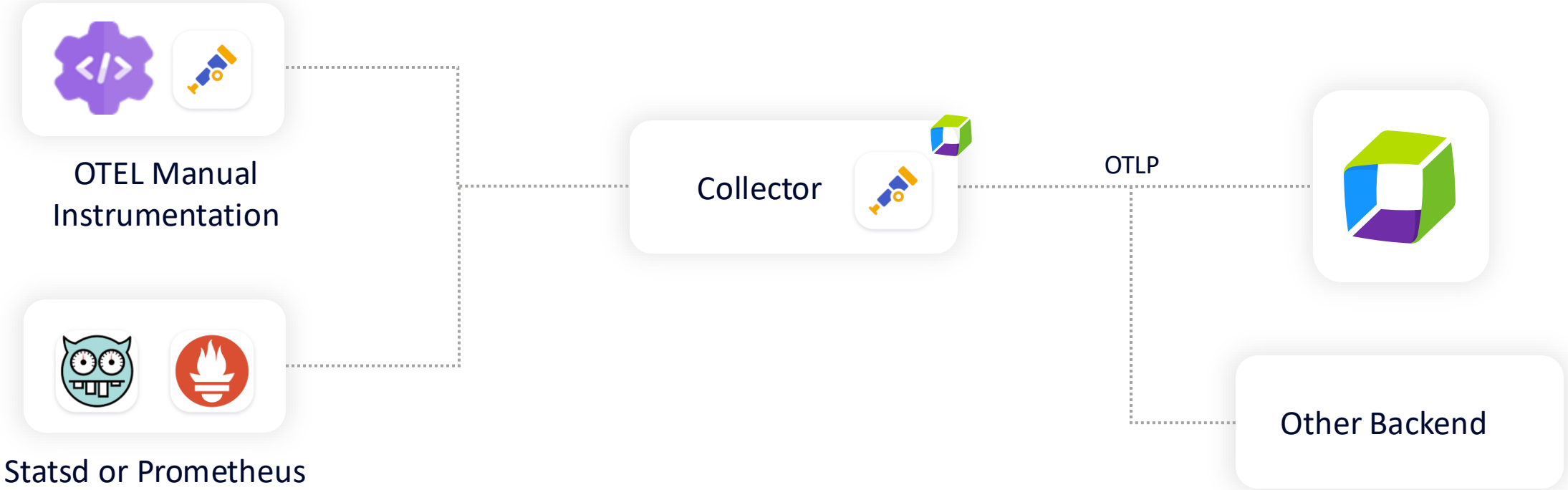
Metrics

- 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

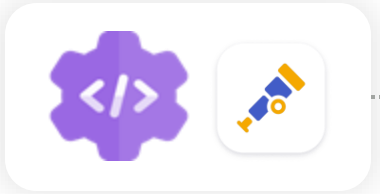
```
DynatraceMetricExporter dynatraceMetricExporter = DynatraceMetricExporter.builder()
        .setUrl("<URL>")
        .setApiToken("<TOKEN>")
        .build();

SdkMeterProvider meterProvider = SdkMeterProvider.builder()
        .registerMetricReader(PeriodicMetricReader.builder(dynatraceMetricExporter).build())
        .build();

OpenTelemetry openTelemetry = OpenTelemetrySdk
        .builder()
        .setMeterProvider(meterProvider)
        .buildAndRegisterGlobal();

Meter meter = openTelemetry.meterBuilder("instrumentation-library-name")
        .setInstrumentationVersion("1.0.0")
        .build();

Attributes attributes = Attributes
        .of(AttributeKey.stringKey("my-key-1"), "my-value-1",
        AttributeKey.stringKey("my-key-2"), Long.valueOf(new Random().nextInt(3)).toString());
```

```
private static LongCounter counter;

counter = meter
        .counterBuilder("my-counter")
        .setDescription("This is my cool counter.")
        .build();

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

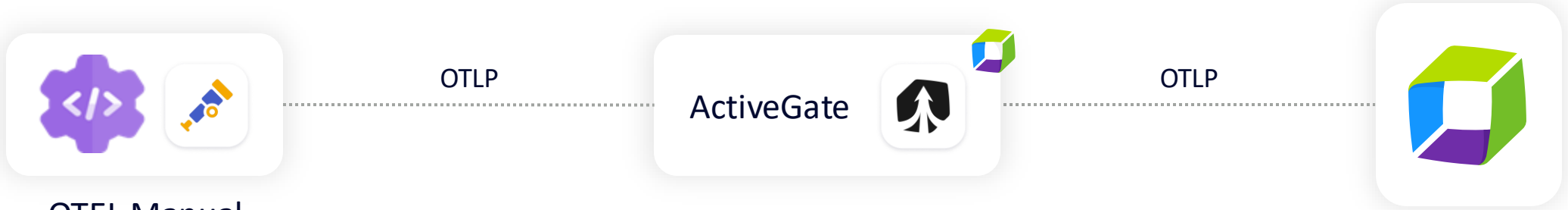
OTLP

OTLP API endpoint

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

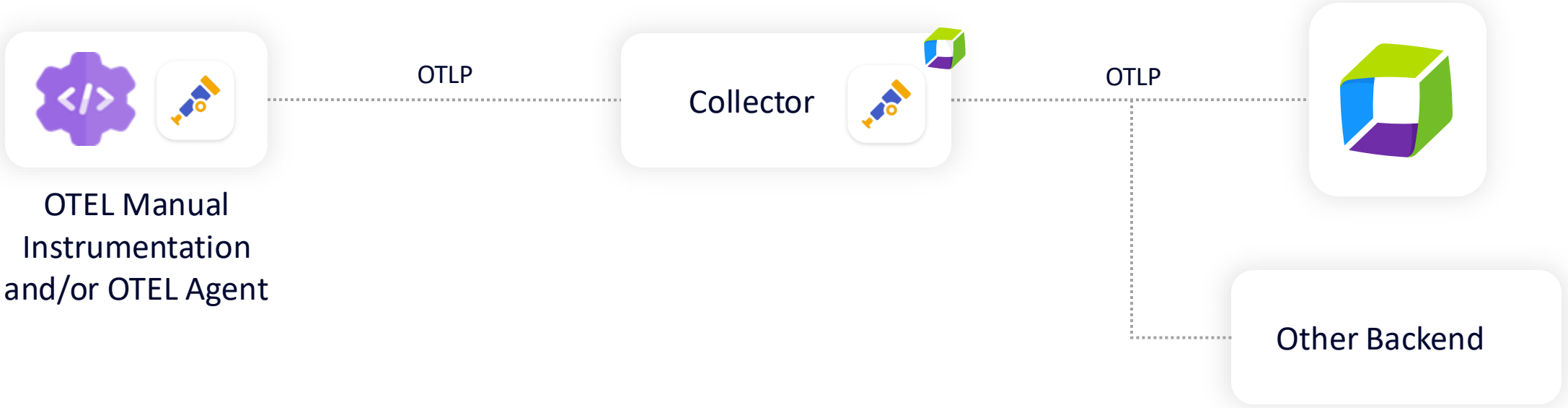
Traces

- Traces sent via ActiveGate



Traces

- Traces sent via OTEL Collector



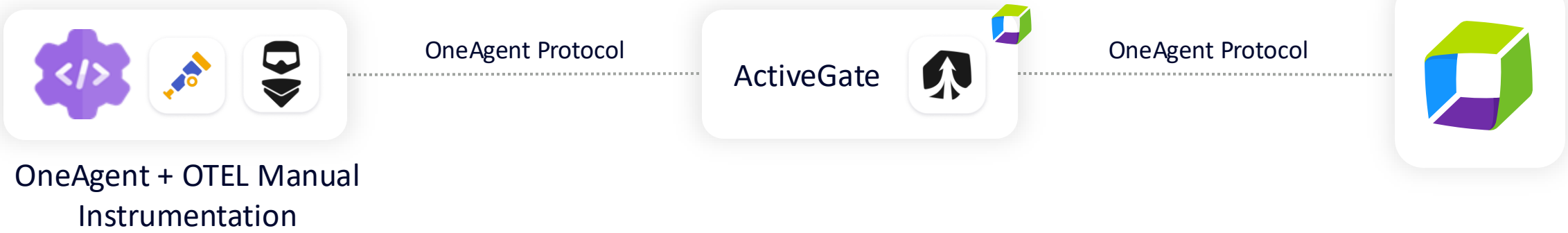
Traces

- 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

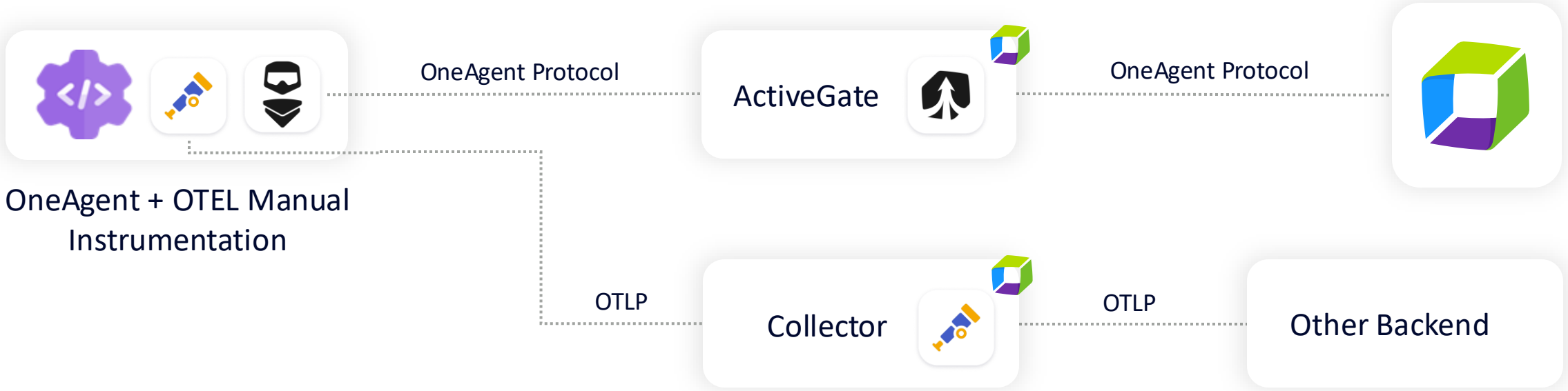
Traces

- 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



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

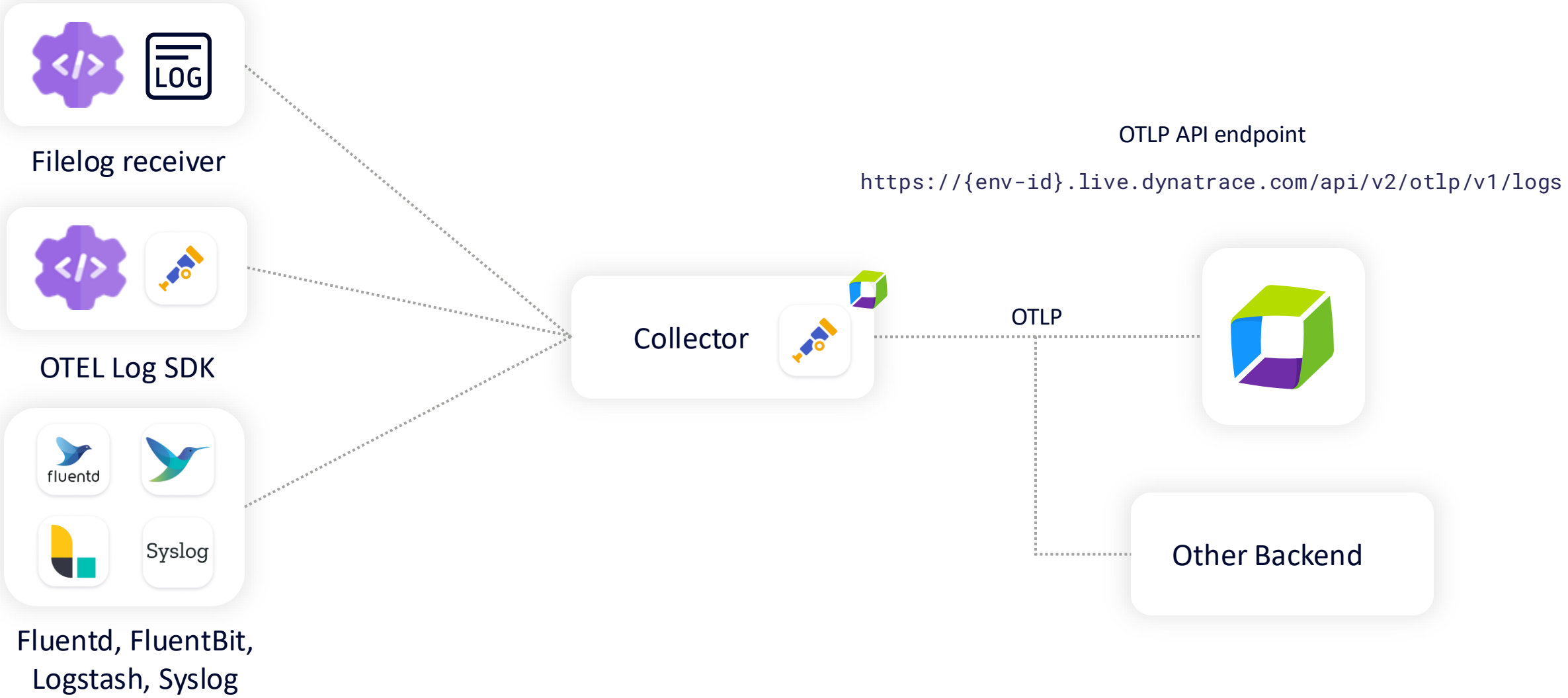
- Manual Instrumentation:

```
Tracer tracer = GlobalOpenTelemetry
    .getTracerProvider()
    .tracerBuilder("my-tracer") //TODO Replace with the name o
    .build();
```

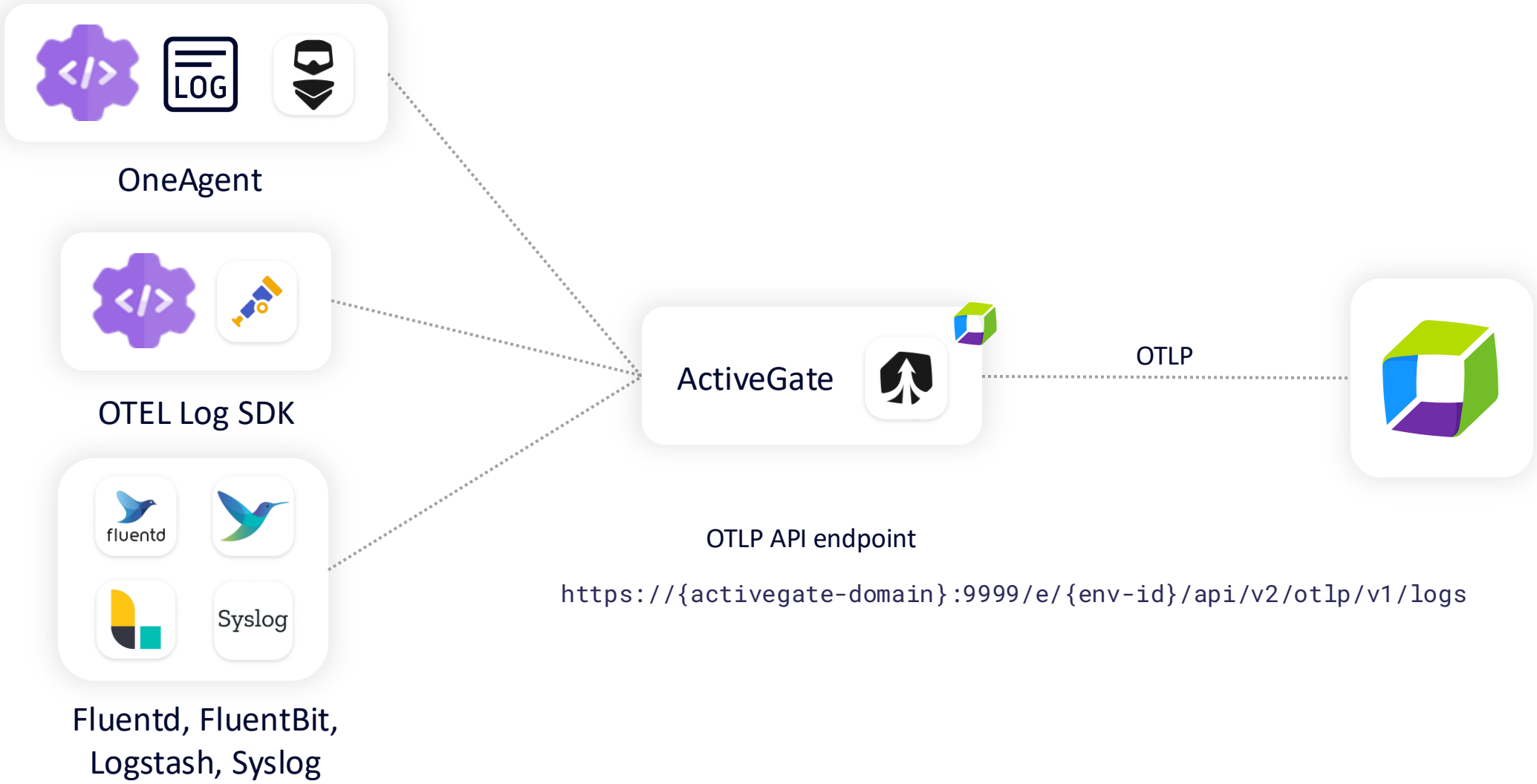
- Auto-Instrumentation Agents:
  - Java
  - PHP
  - Python
  - .NET
  - Go
  - Javascript (Node.js)
  - Ruby

```
// Obtain and name new span from tracer
Span span = tracer.spanBuilder("Call to /myendpoint")
    .setSpanKind(SpanKind.CLIENT)
    .startSpan();

// Set demo span attributes using semantic naming
span.setAttribute("http.method", "GET");
span.setAttribute("net.protocol.version", "1.1");

// Set the span as current span and parent for future child spans
try (Scope scope = span.makeCurrent())
{
    // TODO your code goes here
}
finally
{
    // Completing the span
    span.end();
}
```

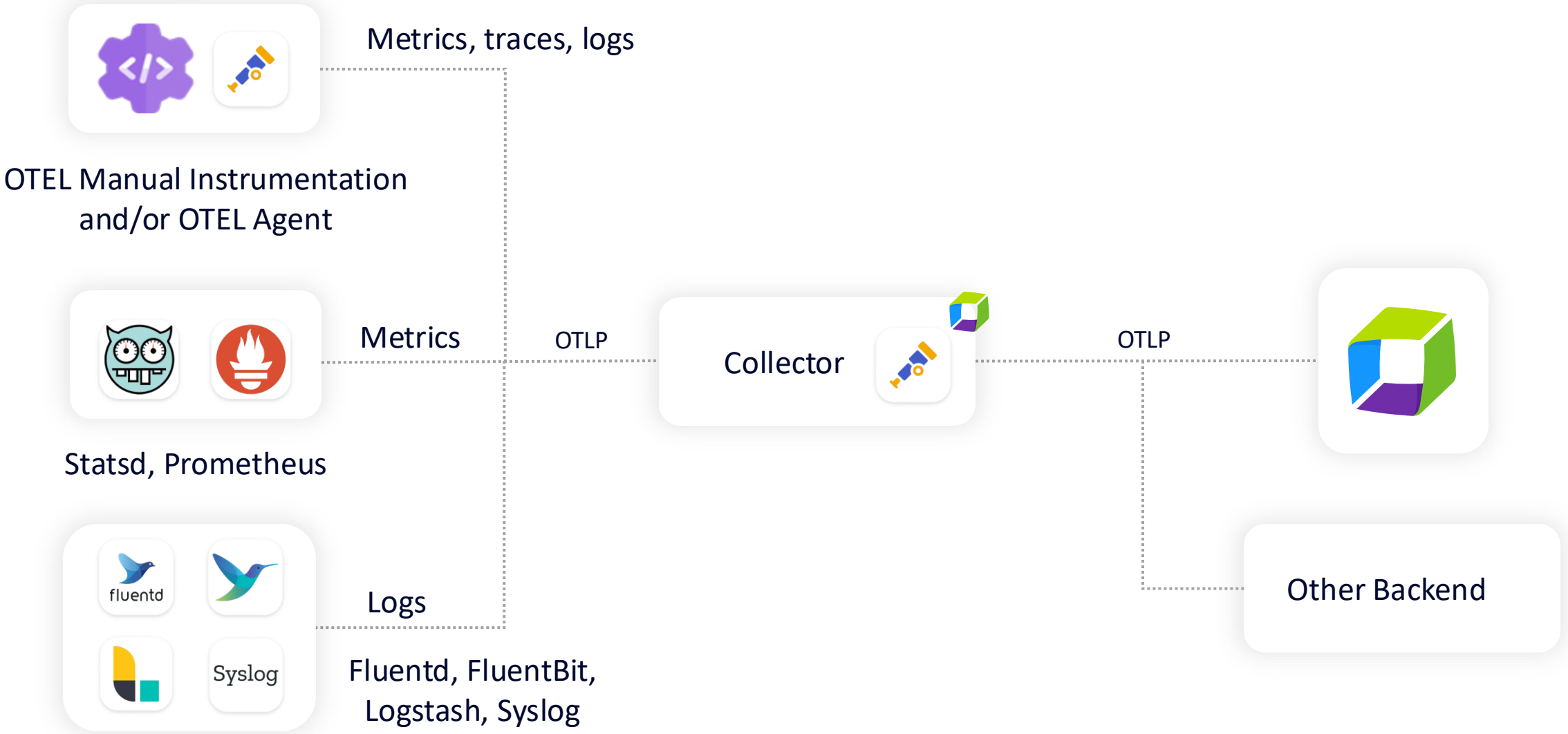
Logs


LOG
Filelog receiver
OTEL Log SDK
fluentd
Syslog
Fluentd, FluentBit,
Logstash, Syslog
Collector
OTLP API endpoint
https://{env-id}.live.dynatrace.com/api/v2/otlp/v1/logs
OTLP
Other Backend

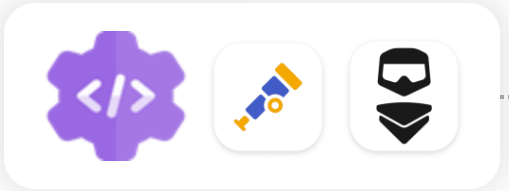
Logs


LOG
OneAgent
OTEL Log SDK
fluentd
Syslog
Fluentd, FluentBit,
Logstash, Syslog
ActiveGate
OTLP
OTLP API endpoint
https://{activegate-domain}:9999/e/{env-id}/api/v2/otlp/v1/logs

Metrics, Traces and Logs

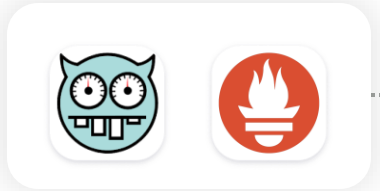


OTEL + OneAgent



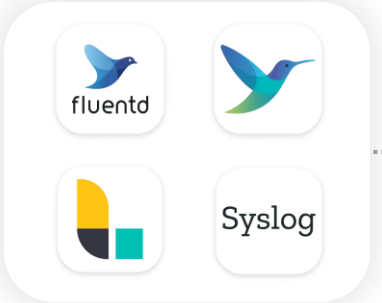
Metrics, traces, logs

OneAgent + OTEL Manual Instrumentation



Metrics

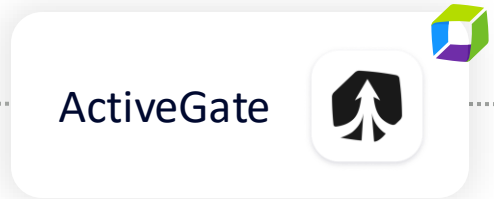
Statsd, Prometheus



Logs

Fluentd, FluentBit, Logstash, Syslog

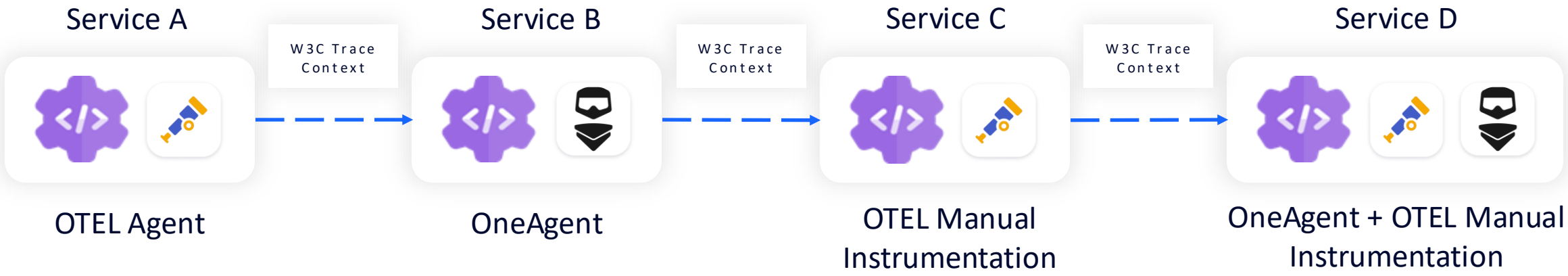
ActiveGate



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

Context Propagation

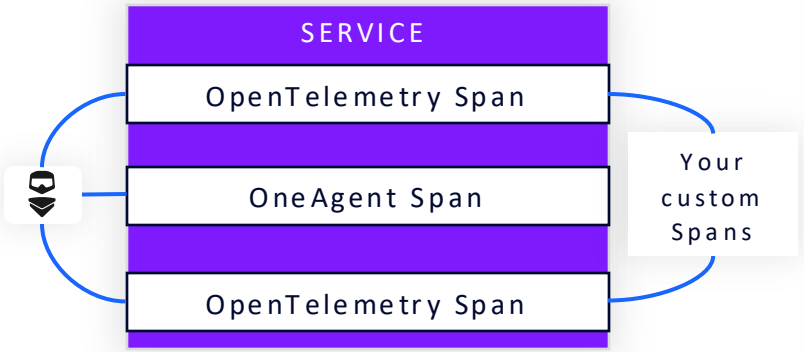
- W3C Trace Context is required to connect spans to a trace across services (HTTP/HTTPS only)
- OneAgents read/write W3C Context Headers
- Manual instrumentation must propagate context to avoid broken traces



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

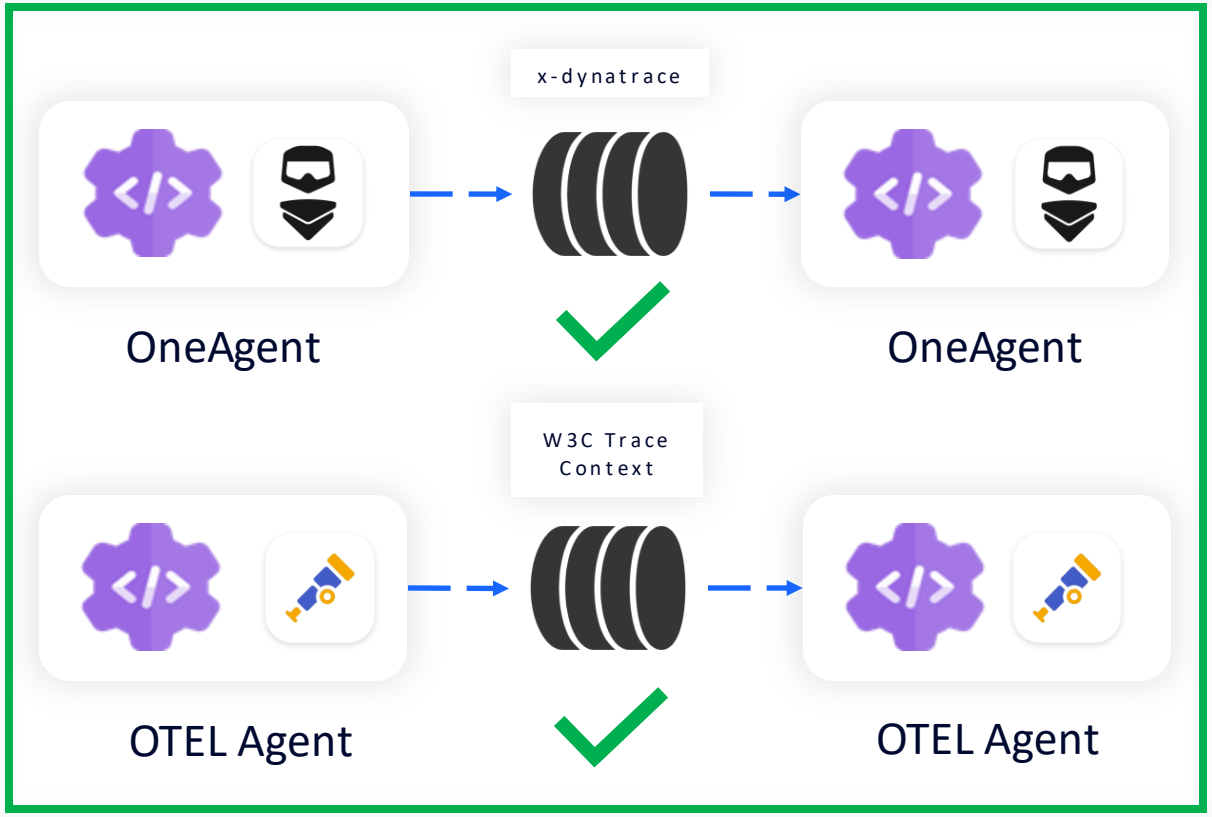
- Extract context when receiving requests
- Inject context when sending requests

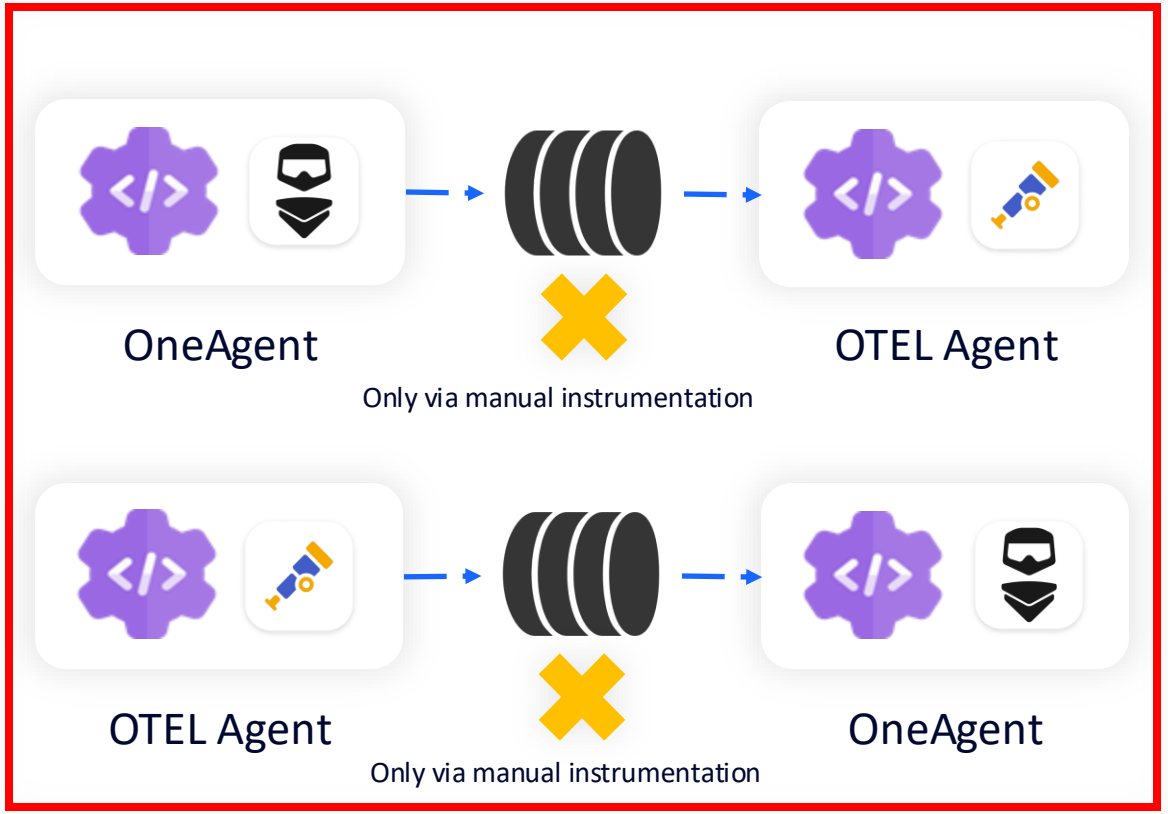
```
//The getter will be used for incoming requests
TextMapGetter<HttpExchange> getter =
    new TextMapGetter<>() {
        @Override
        public String get(HttpExchange carrier, String key) {
            if (carrier.getRequestHeaders().containsKey(key)) {
                return carrier.getRequestHeaders().get(key).get(0);
            }
            return null;
        }

        @Override
        public Iterable<String> keys(HttpExchange carrier) {
            return carrier.getRequestHeaders().keySet();
        }
    };

@Override
public void handle(HttpExchange httpExchange) {
    //Extract the SpanContext and other elements from the request
    Context extractedContext = GlobalOpenTelemetry.getPropagators().getTextMapPropagator()
        .extract(Context.current(), httpExchange, getter);
    try (Scope scope = extractedContext.makeCurrent()) {
        //This will automatically propagate context by creating child spans within the extracted context
        Span serverSpan = tracer.spanBuilder("my-server-span") //TODO Replace with the name of your span
            .setSpanKind(SpanKind.SERVER) //TODO Set the kind of your span
            .startSpan();
        serverSpan.setAttribute(SemanticAttributes.HTTP_METHOD, "GET"); //TODO Add attributes
        serverSpan.end();
    }
}
```

```
//The setter will be used for outgoing requests
TextMapSetter<HttpURLConnection> setter =
    (carrier, key, value) -> {
        assert carrier != null;
        // Insert the context as Header
        carrier.setRequestProperty(key, value);
    };

URL url = new URL("<URL>"); //TODO Replace with the URL of the service to be called
Span outGoing = tracer.spanBuilder("my-client-span") //TODO Replace with the name of your span
    .setSpanKind(SpanKind.CLIENT) //TODO Set the kind of your span
    .startSpan();
try (Scope scope = outGoing.makeCurrent()) {
    outGoing.setAttribute(SemanticAttributes.HTTP_METHOD, "GET"); //TODO Add attributes
    HttpURLConnection transportLayer = (HttpURLConnection) url.openConnection();
    // Inject the request with the *current*  Context, which contains our current span
    GlobalOpenTelemetry.getPropagators().getTextMapPropagator().inject(Context.current(), transportLayer, setter);
    // Make outgoing call
} finally {
    outGoing.end();
}
```

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 bandwith 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

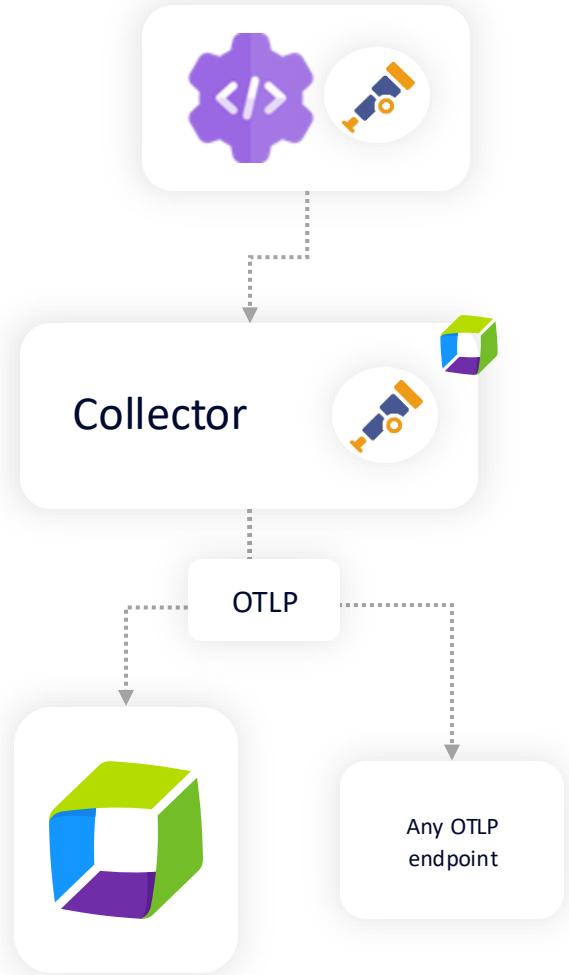
- ActiveGate
  - Automatic or manual updates possible

OpenTelemetry Collector

Additional Information

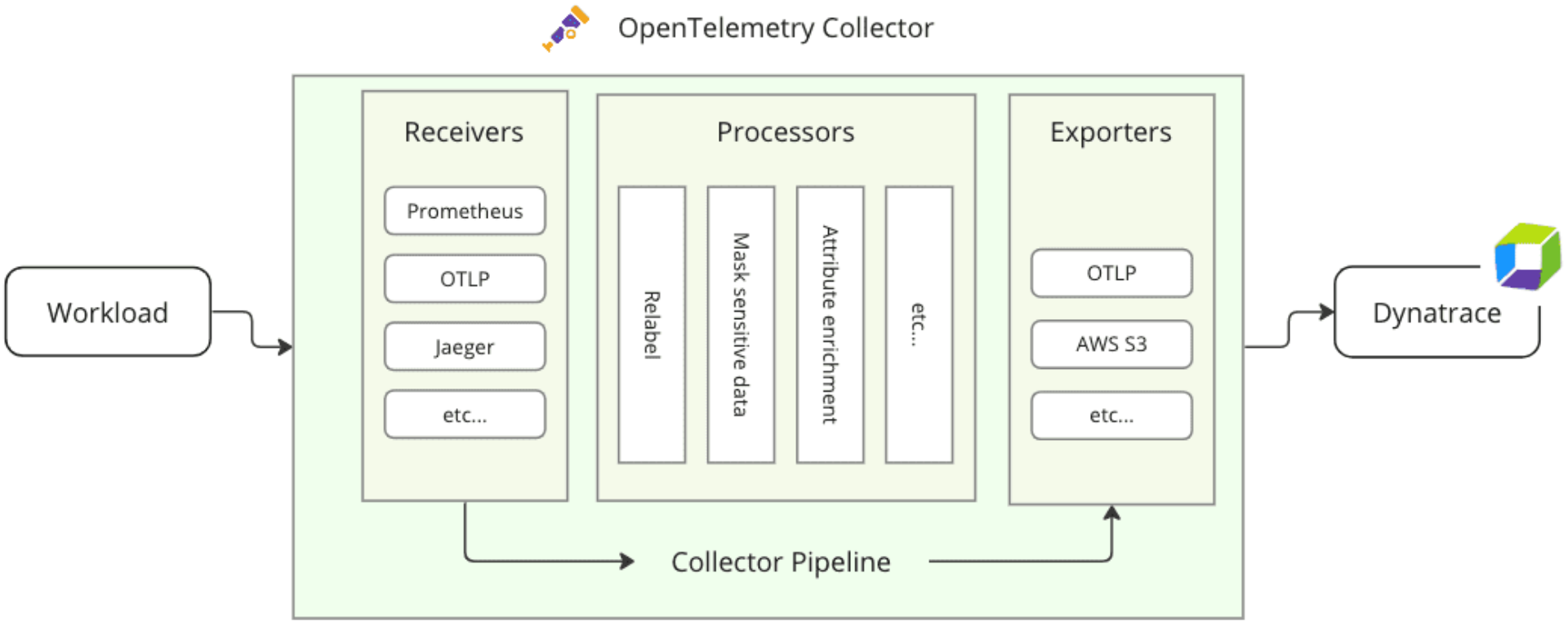
Dynatrace OpenTelemetry Collector Distribution

	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



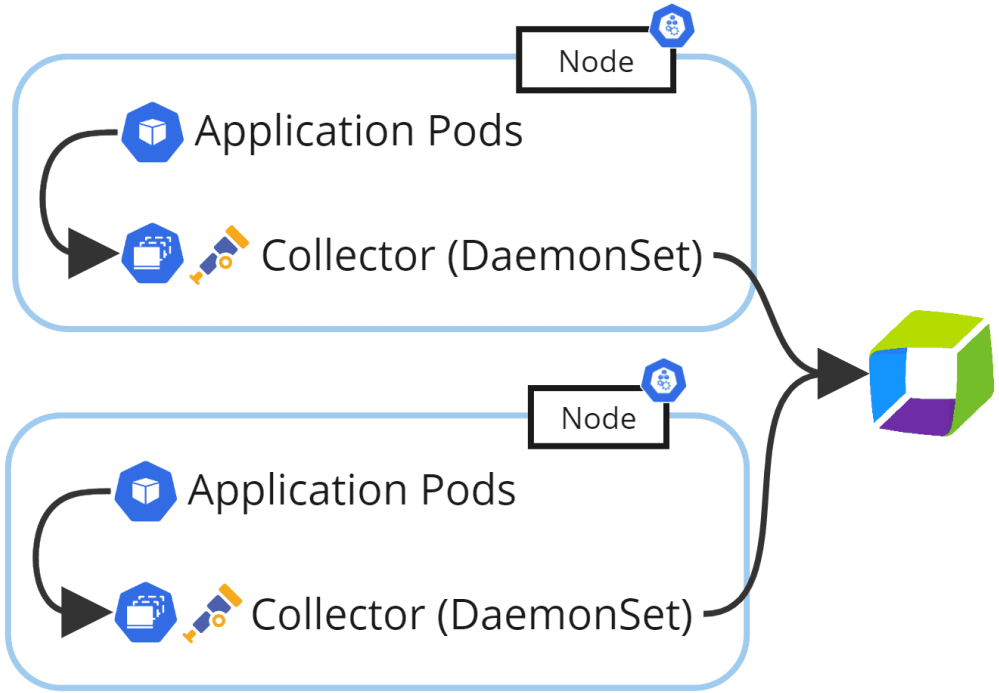
OTEL Collector

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



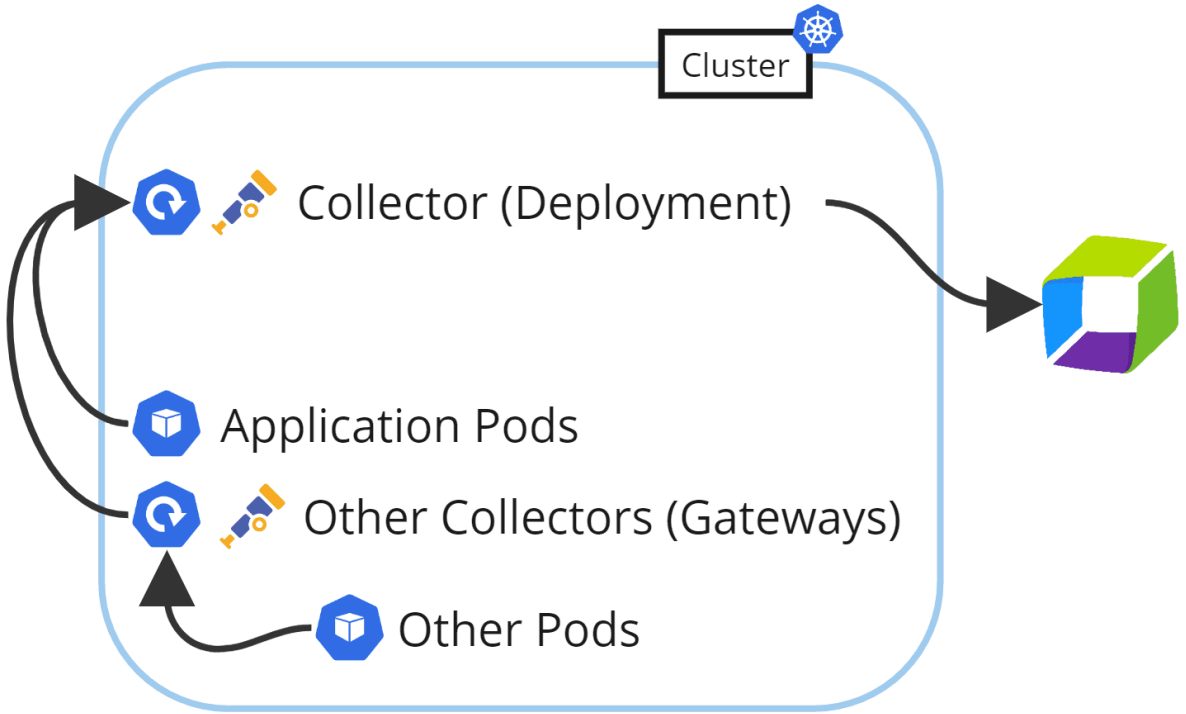
OTEL Collector – Agent

- 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.



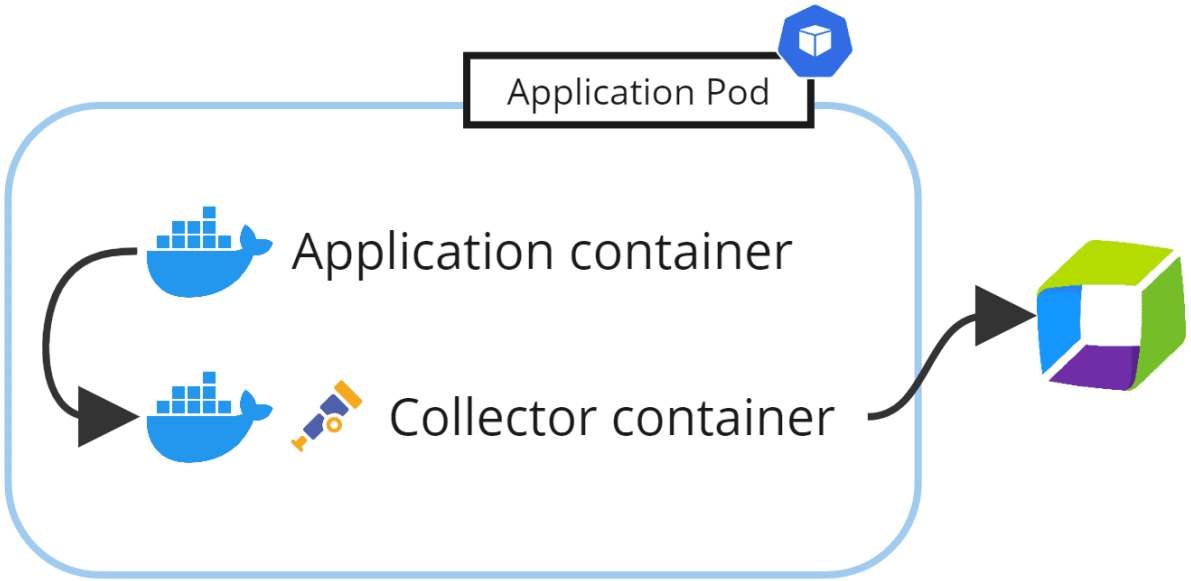
OTEL Collector – Gateway

- 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.



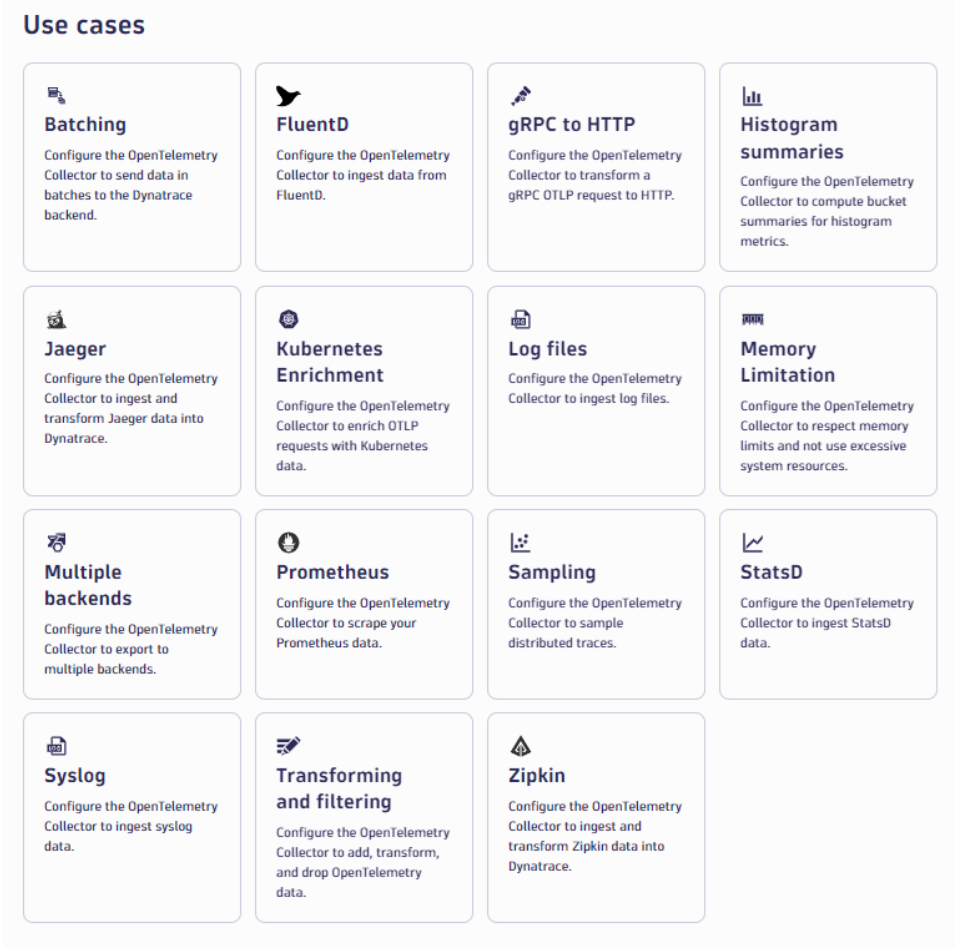
OTEL Collector – Sidecar

- 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.



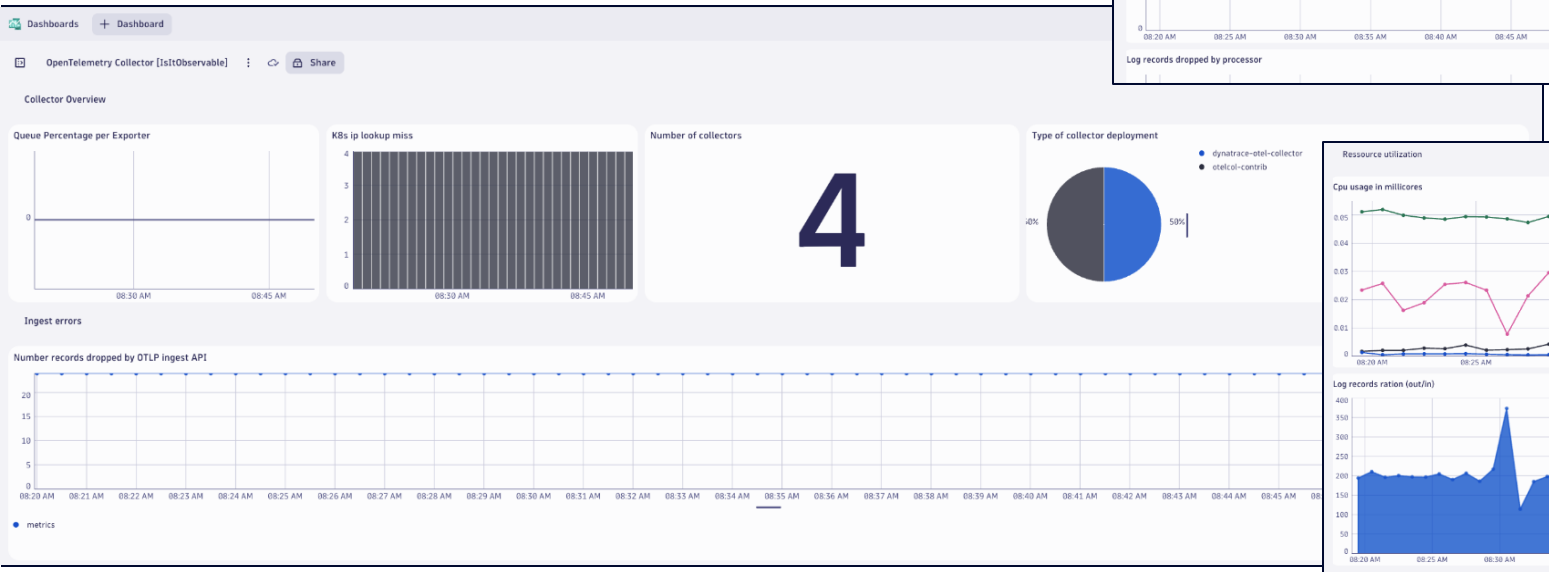
OTEL Collector – Use Cases

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



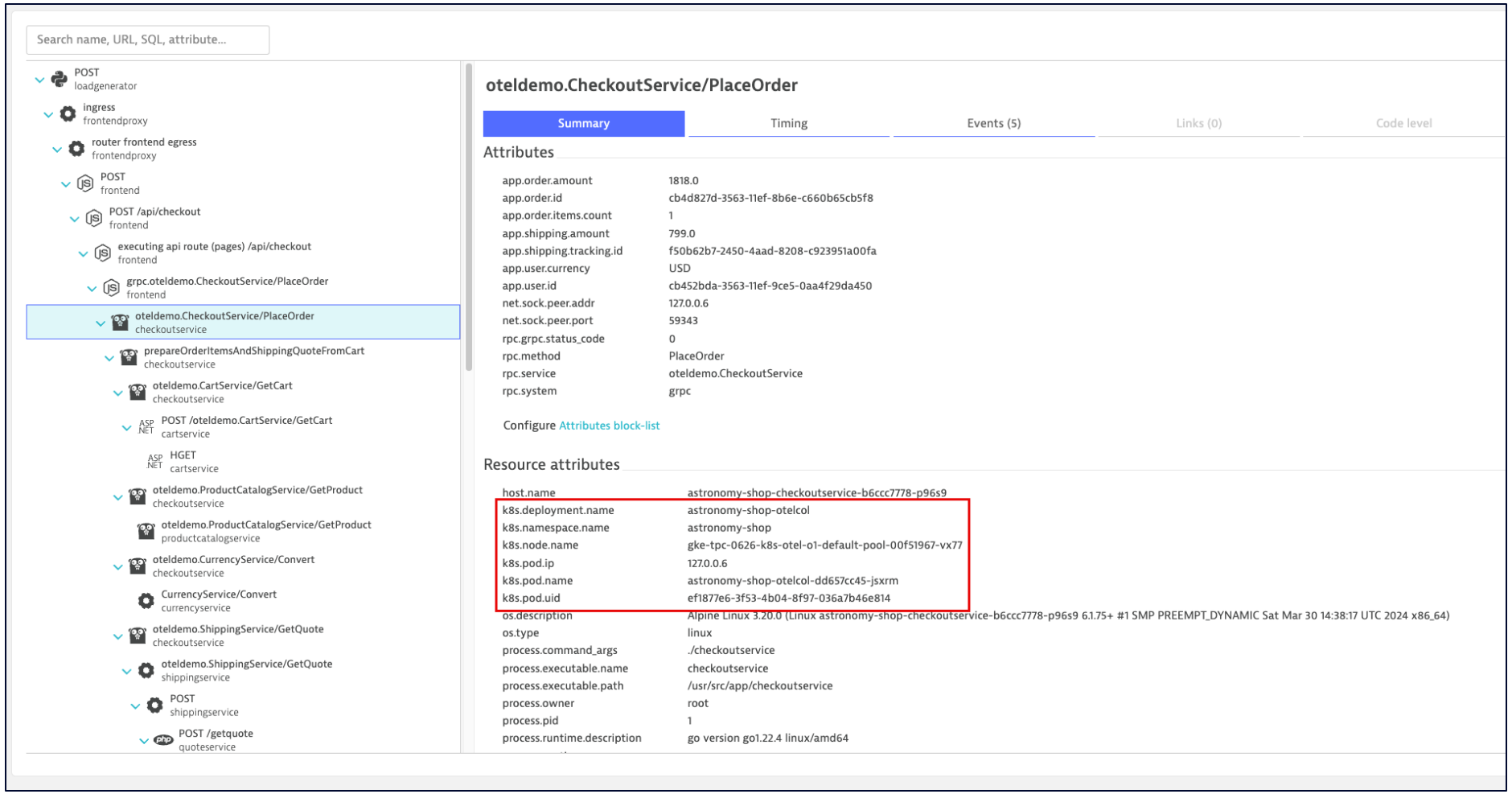
OTEL Collector Monitoring

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



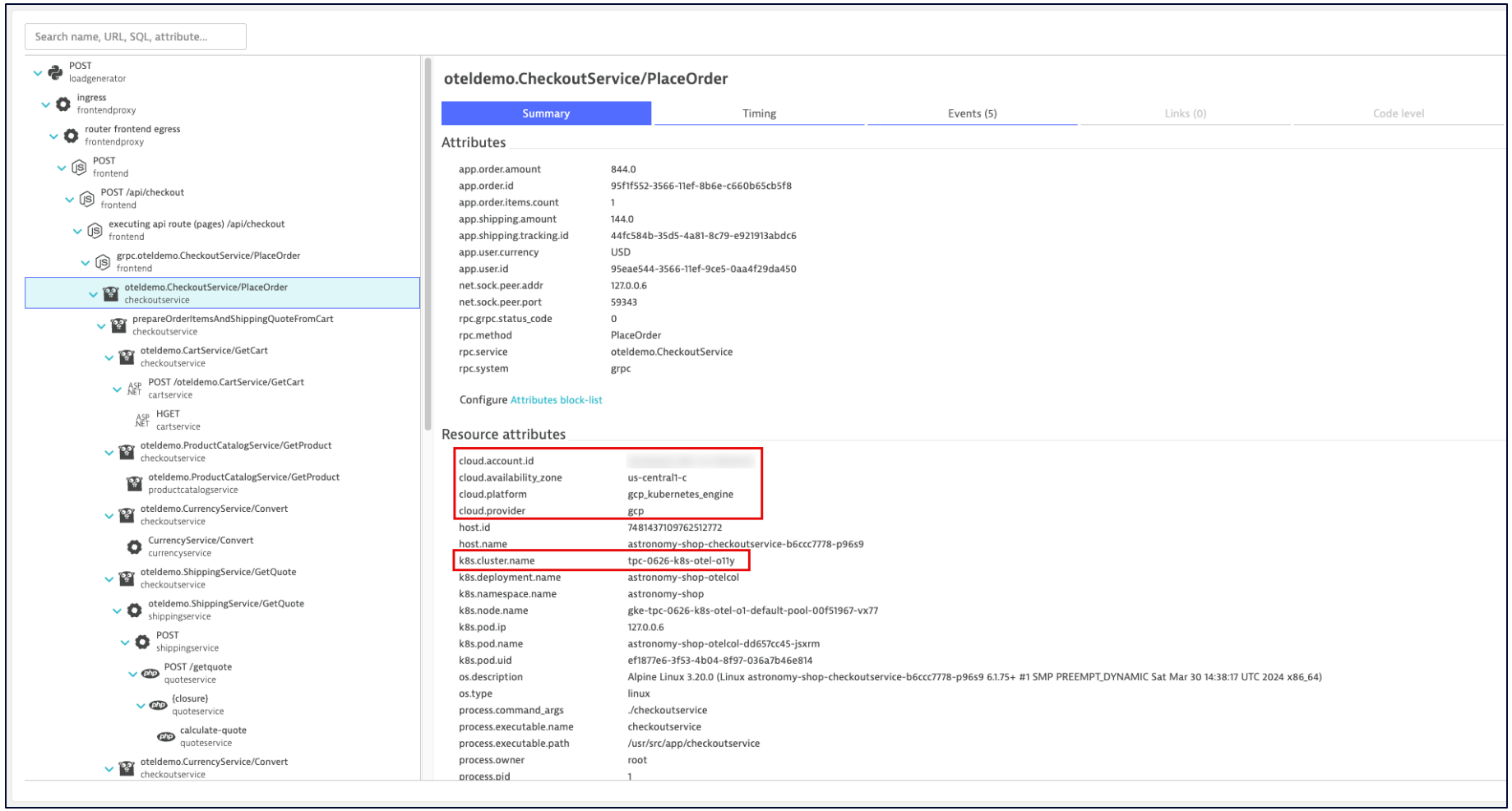
Enrichment via OTEL collector – Kubernetes metadata

- **k8sattributes** processor provides Kubernetes metadata

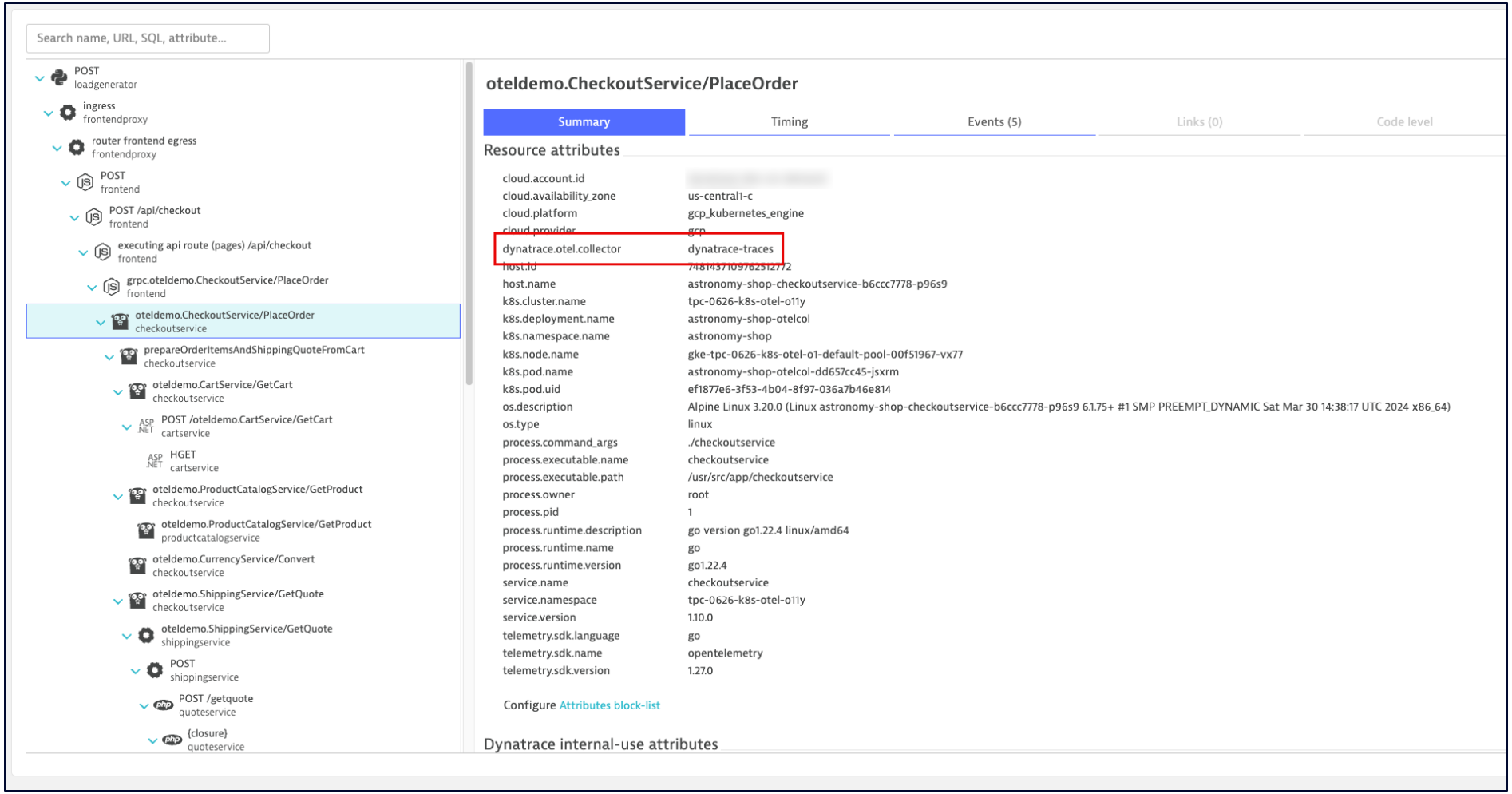


Enrichment via OTEL collector – Cloud metadata

- **resourcedetection** processor provides cloud metadata

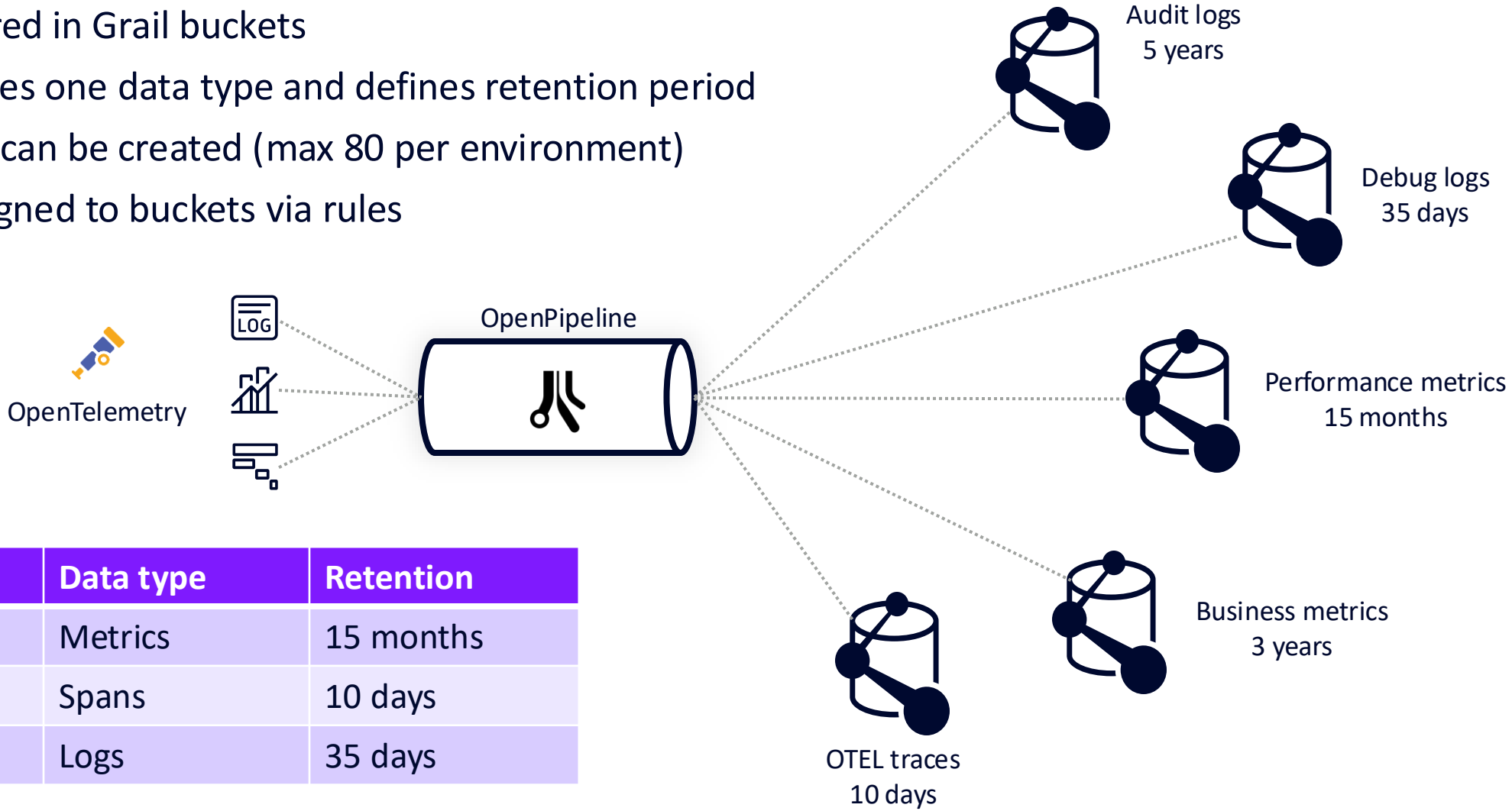


Custom Attributes



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



- Default buckets:

Bucket name	Data type	Retention
default_metrics	Metrics	15 months
default_spans	Spans	10 days
default_logs	Logs	35 days

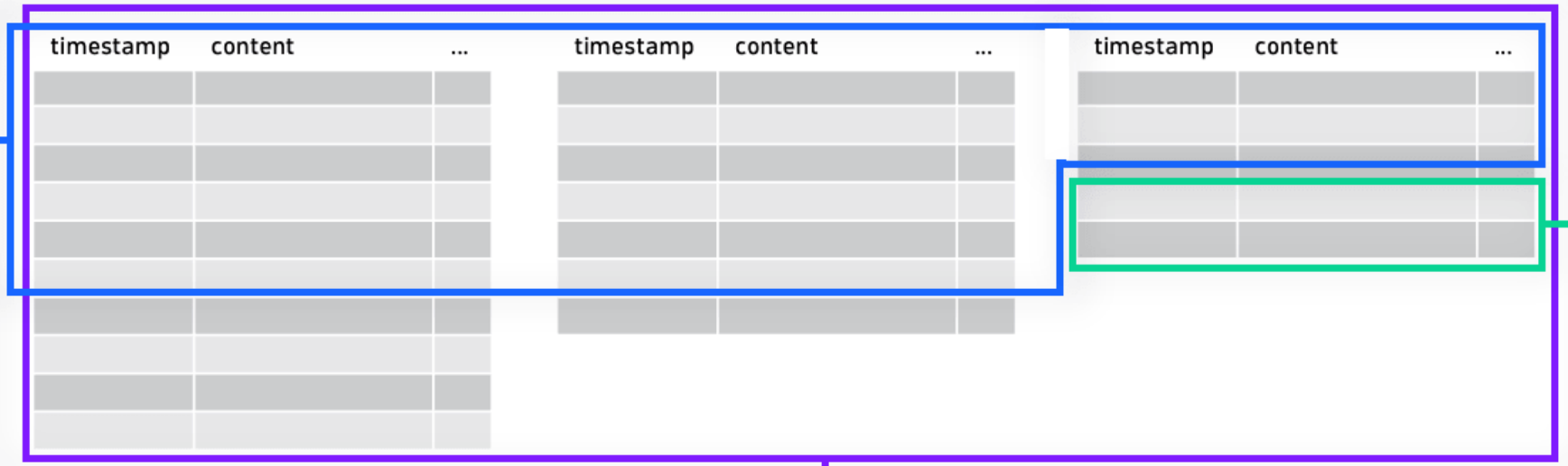
Record Level Security

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



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

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

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

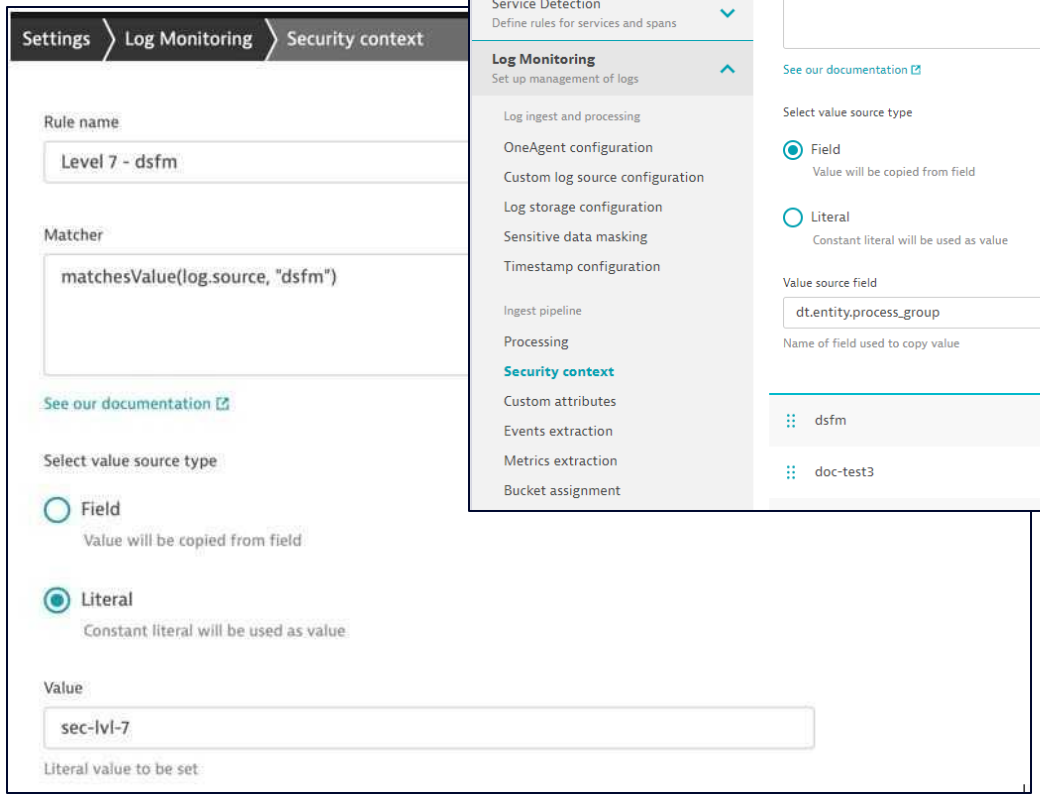
Grants permission to read records from the spans-table

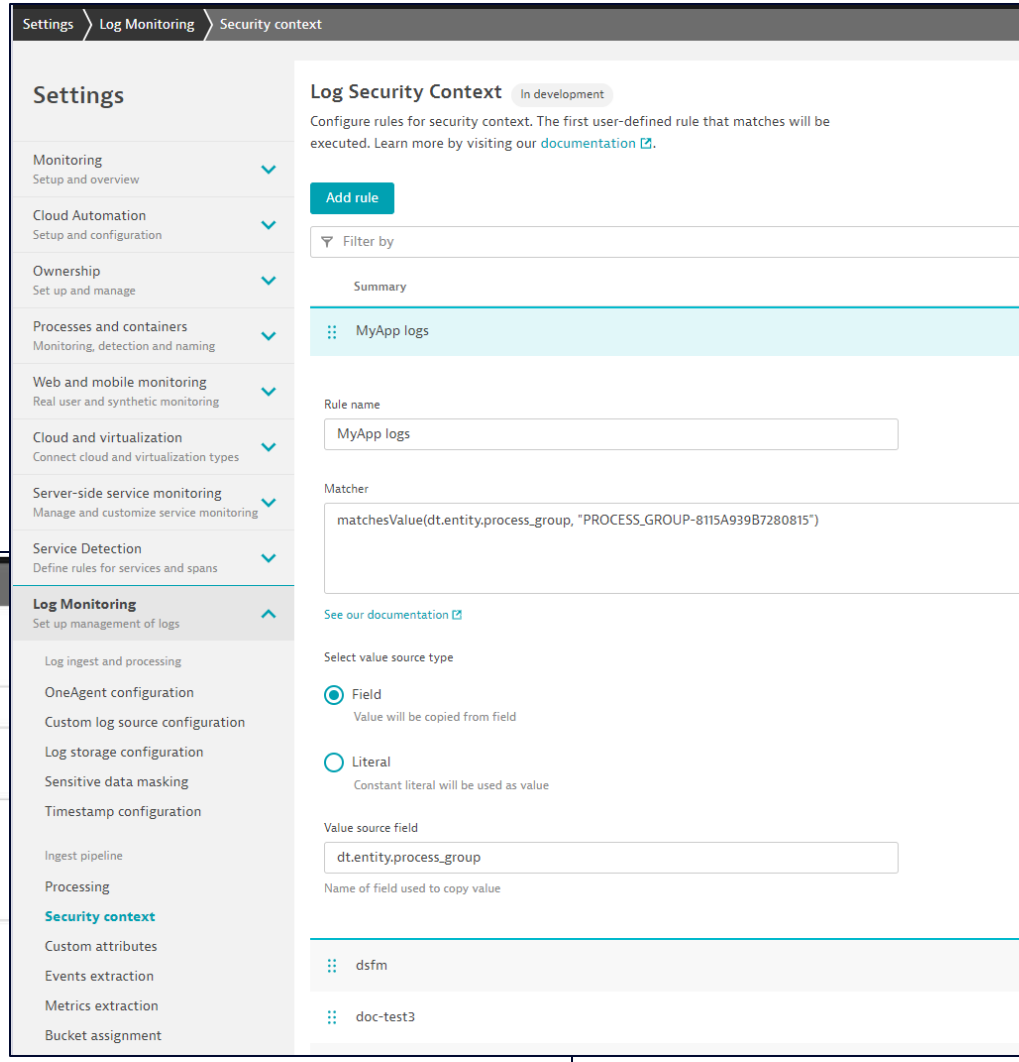
conditions:

- `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





CLOUD DONE RIGHT