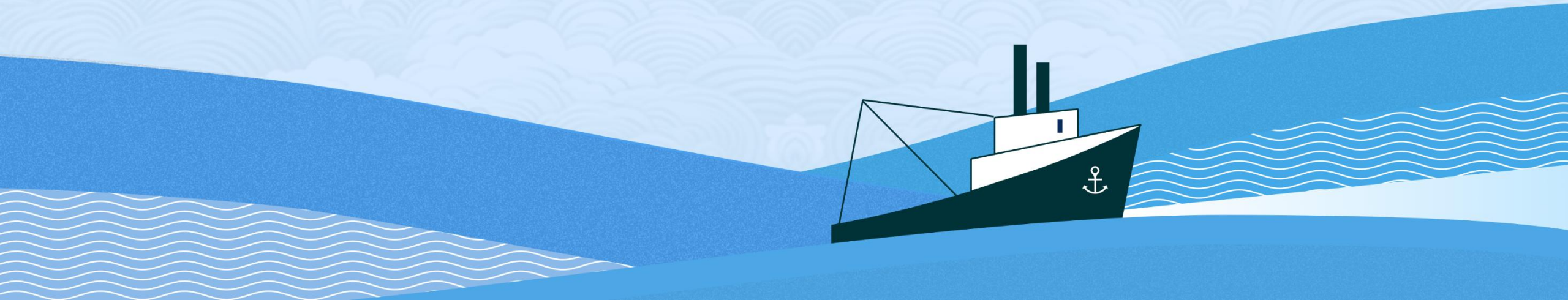
KUBERNETES COMMUNITY DAYS BEIJING 2023

KUBERNETES
COMMUNITY DAYS | BEIJING

使用OpenTelemetry实现开放可观测性的最佳实践

刘睿
liurui@cn.ibm.com



KUBERNETES
COMMUNITY DAYS | BEIJING

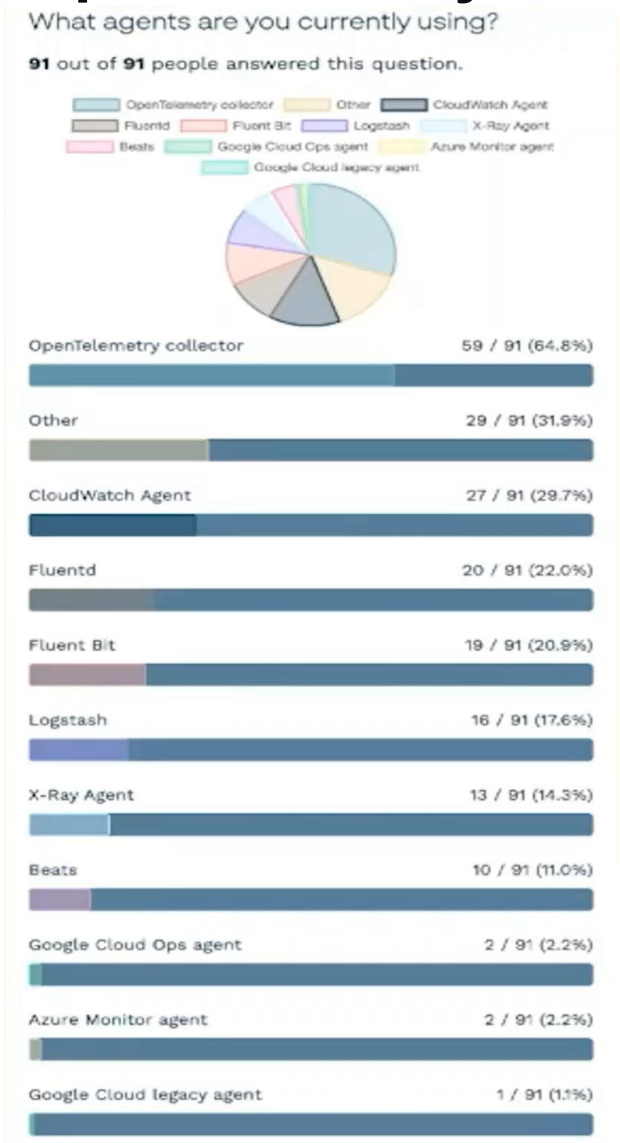
Content

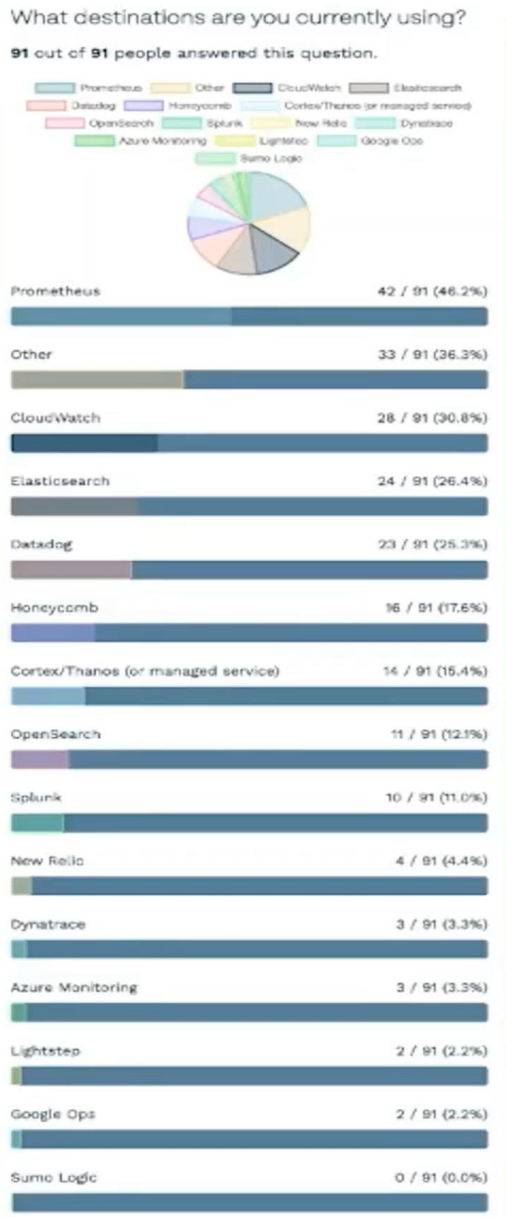
KUBERNETES
COMMUNITY DAYS | BEIJING

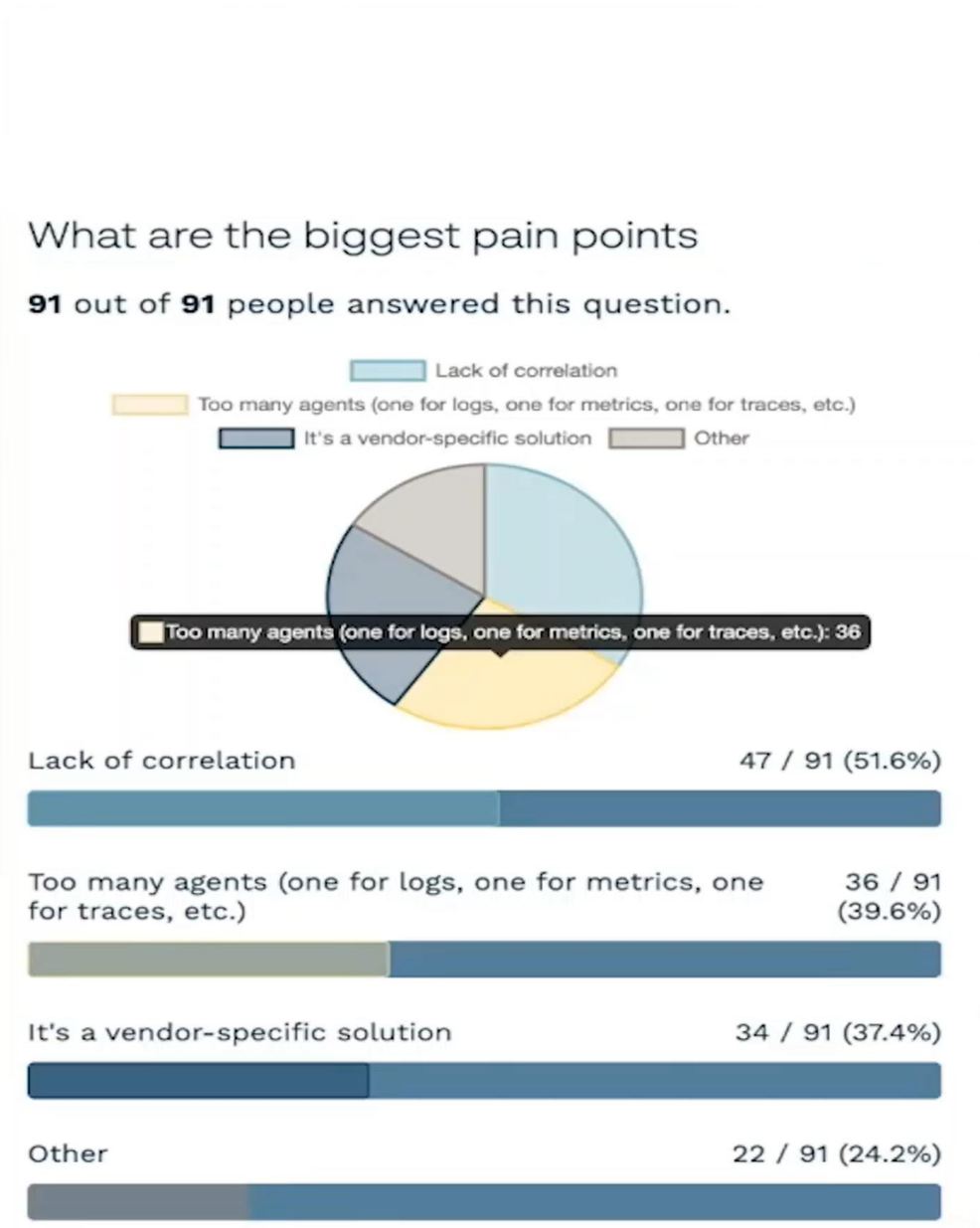
Part 01

OpenTelemetry conceptions

A survey showing that more and more system owners tend to use OpenTelemetry to collect data

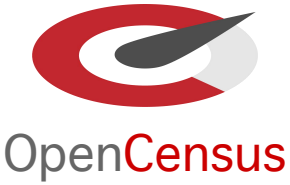






What is OpenTelemetry (OTel)

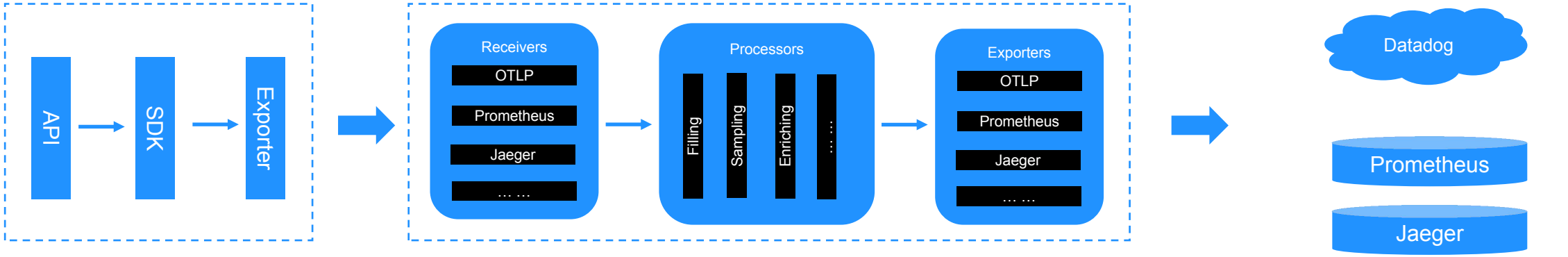
OpenTelemetry is an open-source project and unified standard for service instrumentation, or a way of measuring performance. Sponsored by the **CNCF**, it replaces OpenTracing and OpenCensus.



	Traces	Metrics	Logs
API	for each language		
SDK	for each language		
Infrastructure	Collector, “agent” for auto-instrumentation…		
Interop formats	OTLP, W3C trace-context…		

OpenTelemetry (OTel) Architecture

the industry standard to collector telemetry data for observability vendors



Client

Collector

Observability vendors

What does OTel mean to end users or system owners

- Standard and compatibility
- Avoid vendor lock-in and easy migration
- Can enable different vendors work together
- A standard way to enhance the observability of their systems

What does OTel mean to Observability vendors

- Easier to get telemetry data from open community and customers
- Easier to lose customers if their observability backend is not competitive enough
- It is not likely for OTel to replace all APM agents but it will become the most popular way for apps and micro-services to become observable.

Notes

- OTel code is still a little complex so that It is not likely for all apps to have built-in OTel support in near future.

Conceptions of 3 signal types of OpenTelemetry

Note: New signal of Profiling data is now under design…

KUBERNETES COMMUNITY DAYS BEIJING

Traces

- **Context:** W3C trace-context, B3, etc.
- **Tracer:** get context
- **Spans:** “call” in a trace
  - **Kind:** client/server, producer/consumer, internal
  - **Attributes:** key/value pairs
  - **Events:** named strings
  - **Links:** useful for batch operations
- **Sampler:** always, probabilistic, etc.
- **Span processor:** simple, batch, etc.
- **Exporter:** OTLP, Jaeger, etc.

Metrics

- **Meter: used to record a measurement**
- **Metric: a measurement**
  - **Kind:** Counter, UpDownCounter, Gauge, Histogram, Summary
  - **Label:** key/value pair
  - **Data point**
- **Exporter: OTLP, Prometheus, etc.**

Logs

- Use Collector or Appenders for integration.
- **Log model:**
  - **LogRecord:** time, severity, name, body, attributes, TraceId, SpanId…

Resources

- Immutable Attributes
- Detecting resource information from the environment
- Specifying resource information via an environment variable
  - **OTEL_RESOURCE_ATTRIBUTES**

Semantic conventions

Note: OpenTelemetry has adopted ECS (OpenTelemetry Semantic Convention) to be part of its own Semantic Conventions.

KUBERNETES COMMUNITY DAYS BEIJING

Resources

- https://opentelemetry.io/docs/reference/specification/resource/semantic_conventions/
- Examples:
  - service.name
  - host.id

Traces

- https://opentelemetry.io/docs/reference/specification/trace/semantic_conventions/
- Examples:
  - peer.service
  - messaging.system

Metrics

- https://opentelemetry.io/docs/reference/specification/metrics/semantic_conventions/
- Examples:
  - system.cpu.time
  - system.cpu.utilization
  - system.memory.usage
  - ...

Logs

- https://opentelemetry.io/docs/reference/specification/logs/semantic_conventions/
- Examples:
  - log.file.name

KUBERNETES
COMMUNITY DAYS | BEIJING

Part 02

Automatic & manual instrumentation

Solutions of automatic instrumentation for different languages

Language	Support?	Solution
Java	Yes	Java Agent
Python	Yes	Python agent
.Net	Yes	.NET Profiler
Javacript	Yes	Preloaded module
GoLang	Yes(Beta)	eBPF
PHP	Yes(At least PHP 8.0)	the OpenTelemetry PHP extension
Ruby	minor global code changes	use the opentelemetry-instrumentation-all package
C++	N/A	Manu-instrumentation, vcpkg
Rust	N/A	Manu-instrumentation

Note: OpenTelemetry Operator supports convenient auto-instrumentation installation in Kubernetes environments.

Example of Java auto-instrumentation

KUBERNETES COMMUNITY DAYS BEIJING

Example-1 for Java

```
export JAVA_OPTS="-javaagent:/opt/dev/otel/res/opentelemetry-javaagent.jar"

export OTEL_TRACES_EXPORTER=logging

export OTEL_METRICS_EXPORTER=logging

export OTEL_LOGS_EXPORTER=logging
```

Example-2 for Java

```
export JAVA_OPTS="-javaagent:/opt/dev/otel/res/opentelemetry-javaagent.jar"

export OTEL_TRACES_EXPORTER=otlp

export OTEL_METRICS_EXPORTER=none

export OTEL_LOGS_EXPORTER=none

export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT="https://otlp-orange-saas.instana.io:4317"

export OTEL_SERVICE_NAME=lr1service

export OTEL_EXPORTER_OTLP_PROTOCOL=grpc

export OTEL_EXPORTER_OTLP_HEADERS="x-instana-key=xxxxxxxxx,x-instana-host=dk-liurui"
```

Example of Java manual-instrumentation

KUBERNETES COMMUNITY DAYS BEIJING

Initialization

```
static void init() {
    //Create Resource
    AttributesBuilder attrBuilders = Attributes.builder()
            .put(ResourceAttributes.SERVICE_NAME, SERVICE_NAME)
            .put(ResourceAttributes.SERVICE_NAMESPACE, t: "US-West-1")
            .put(ResourceAttributes.HOST_NAME, t: "prodsvc.us-west-1.example.com");

    Resource serviceResource = Resource
            .create(attrBuilders.build());
    //Create Span Exporter
    OtlpGrpcSpanExporter spanExporter = OtlpGrpcSpanExporter.builder()
            .setEndpoint("http://dk-liurui1.fyre.ibm.com:4317")
            //.setEndpoint("http://dk-liurui1.fyre.ibm.com:55689") //through OTel collector
            .build();

    //Create SdkTracerProvider
    SdkTracerProvider sdkTracerProvider = SdkTracerProvider.builder()
            .addSpanProcessor(BatchSpanProcessor.builder(spanExporter)
                    .setScheduleDelay( delay: 100, TimeUnit.MILLISECONDS).build())
            .setResource(serviceResource)
            .build();

    //This Instance can be used to get tracer if it is not configured as global
    OpenTelemetry openTelemetry = OpenTelemetrySdk.builder()
            .setTracerProvider(sdkTracerProvider)
            .buildAndRegisterGlobal();
}
```

Instrumentation for traces

```
public void doLogin(String username, String password) {
    Span parentSpan = tracer.spanBuilder( s: "doLogin").startSpan();
    parentSpan.setAttribute("priority", "business.priority");
    parentSpan.setAttribute("prodEnv", true);

    try (Scope scope = parentSpan.makeCurrent()) {
        Thread.sleep( millis: 200);
        boolean isValid = isValidAuth(username, password);
        //Do login

    } catch (Throwable t) {
        parentSpan.setStatus(StatusCode.ERROR, s: "Change it to your error message");
    } finally {
        parentSpan
                .end(); // closing the scope does not end the span, this has to be do
    }

}
```

KUBERNETES
COMMUNITY DAYS | BEIJING

Part 03

OpenTelemetry Collector

Configuration of OTel Collector

- Key Components of OTel Collector:
  - Receivers
  - Processors
  - Exporters
  - Connector (new)
- Types of collector:
  - Core Collector
  - Contrib Collector
  - Custom Collector
- Releases and deployments
  - Binary
  - Container based

```
receivers:
  otlp:
    protocols:
      grpc:
      http:

processors:
  batch:

exporters:
  otlp:
    endpoint: otelcol:4317

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlp]
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlp]
    logs:
      receivers: [otlp]
      processors: [batch]
      exporters: [otlp]
```

Another example of collector configuration

```yaml
receivers:
  hostmetrics:
    collection_interval: 30s
    scrapers:
      cpu:
      memory:
      load:
      network:
      processes:
      process:

  hostmetrics/disk:
    collection_interval: 1m
    scrapers:
      disk:
      filesystem:
      paging:

exporters:
  otlp:
    endpoint: "10.21.15.86:4317"
    tls:
      insecure: true

  prometheus:
    endpoint: "0.0.0.0:8889"

  logging:
    verbosity: detailed
```

```yaml
processors:
  batch:

  resourcedetection:
    detectors: [env, system]
    timeout: 2s
    override: true
    system:
      hostname_sources: ["os"]

extensions:
  health_check:
  pprof:
    endpoint: :1888
  zpages:
    endpoint: :55679

service:
  extensions: [pprof, zpages, health_check]
  pipelines:
    metrics:
      receivers: [hostmetrics, hostmetrics/disk]
      processors: [batch, resourcedetection]
      exporters: [logging, prometheus]
```

Best practice to use OpenTelemetry Collector

❖ Key points:

- Collectors can be chained
- Receiver instance can be shared
- Processor instance cannot be shared
- Exporter instance can be shared
- Use Connector can build complex stream flows within a single Collector.

❖ Some popular components of Collector:

- Receivers ~ 91
  otlp, filelog, hostmetric, prometheus, jaeger…
- Processors ~ 25
  batch, filter, resourcedetection, spanmetrics, transform, probabilisticsampler…
- Exporters ~ 49
  otlp, otlphttp, prometheus, jaeger, logging, file…
- Connectors ~ 5
  forward, spanmetrics, count

KUBERNETES
COMMUNITY DAYS | BEIJING

Part 04
Deployment considerations

Deployments of OpenTelemetry

- With Binary OpenTelemetry Collector – for example for the usage of host metrics receiver…
- With Container based OpenTelemetry Collector – optionally using docker compose.
- Enable OpenTelemetry in Kubernetes
  - Deploy OpenTelemetry Collector with built-in Kubernetes components (deployments or daemonset, service, configmap…), optionally using Helm chart.
  - Using OpenTelemetry Operator
    - Have built-in OpenTelemetry Collector
    - Support auto-instrumentation (inject sidecar into workloads with predefined annotation) for Java, Python, .Net, NodeJS, GoLang…

An example of deployment using OTel operator

KUBERNETES COMMUNITY DAYS BEIJING

```yaml
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  labels:
    app.kubernetes.io/managed-by: opentelemetry-operator
  name: otel
spec:
  config: |
    receivers:
      otlp:
        protocols:
          grpc:
          http:
      otlp/spanmetrics:
        protocols:
          grpc:
            endpoint: 0.0.0.0:55677

    processors:
      batch:
        timeout: 10s
        send_batch_size: 10000
      spanmetrics:
        metrics_exporter: prometheus
        latency_histogram_buckets:
          [10ms, 100ms, 200ms, 400ms, 800ms, 1s, 1200ms, 1400ms, 1600ms, 1800ms, 2s, 4s, 6s, 8s, 1
        dimensions:
          - name: http.method
          - name: http.status_code
          - name: http.target
          - name: http.url
      metricstransform:
        transforms:
          - include: duration
            match_type: regexp
            action: update
            operations:
              - action: update_label
                label: http.url
                new_label: url
              - action: update_label
                label: http.method
                new_label: method
              - action: update_label
                label: http.status_code
                new_label: code

    exporters:
      otlp:
        endpoint: ${env:BACKEND_SERVER}
        tls:
          insecure: true
      jaeger:
        endpoint: ${env:JAEGER_COLLECTOR}
        tls:
          ca_file: "/capath/service-ca.crt"
      prometheus:
        endpoint: "0.0.0.0:8889"
        send_timestamps: true
        metric_expiration: 1440m

    connectors:
      spanmetrics:
        histogram:
          unit: s
          explicit:
            buckets: [10ms, 100ms, 200ms, 400ms, 800ms, 1s, 1200ms, 1400ms, 1600ms, 1800ms, 2s, 4s, 6s, 8s, 10s]
        dimensions:
          - name: http.method
          - name: http.status_code
          - name: http.target
          - name: http.url
          - name: http.route

    service:
      pipelines:
        traces:
          receivers: [otlp]
          processors: [batch]
          exporters: [spanmetrics, logging, jaeger, otlp]
        metrics:
          receivers: [otlp/spanmetrics, spanmetrics]
          processors: [batch, metricstransform]
          exporters: [prometheus, logging, otlp]
  mode: statefulset
  resources: {}
  upgradeStrategy: automatic
  volumeMounts:
    - mountPath: /capath
      name: cabundle-volume
  ingress:
    route: {}
  volumes:
    - configMap:
        name: my-otelcol-cabundle
      name: cabundle-volume
  targetAllocator:
    prometheusCR: {}
  image: >-
    ghcr.io/open-telemetry/opentelemetry-collector-releases/opentelemetry-collector-contrib:0.75.0
  replicas: 1
```

KUBERNETES
COMMUNITY DAYS | BEIJING

Thanks.