KubeCon | CloudNativeCon

North America 2024

KubeCon CloudNativeCon

North America 2024

Mastering OpenTelemetry Collector Configuration

Steve Flanders

Steve Flanders

Engineering Leader, Splunk
https://sflanders.net

Founding member of OpenCensus and OpenTelemetry projects

10+ years of monitoring and observability experience

- OpenTelemetry + Metrics at Splunk (acquired by Cisco)
- OpenCensus + Traces (APM) at Omnition (acquired by Splunk)
- Logs at VMware

Author of the book: Mastering OpenTelemetry and Observability!

OTel Me Why

An Introduction to OpenTelemetry

What is OpenTelemetry?

OpenTelemetry is an Observability framework and toolkit designed to create and manage telemetry data such as traces, metrics, and logs. OpenTelemetry is vendor- and tool-agnostic, meaning that it can be used with a broad variety of Observability platforms, including open source tools like Jaeger and Prometheus, as well as commercial offerings. - https://opentelemetry.io/docs/what-is-opentelemetry/

INSTRUMENTATION LIBRARIES
Single library per language

DATA COLLECTION
Single binary in multiple form factors to receive, process, and export data

SPECIFICATION
API (conventions + definition), SDK (API + configuration), and Data (data formats, protocols, and semantic conventions)

Why does OpenTelemetry matter?

Open Standard and Implementation ✓

A vendor-agnostic specification for telemetry data, including API, SDK, and data aspects.

A single instrumentation library per language and a collector binary that support multiple form factors.

Data Portability and Control ✓

The freedom to send your data to your observability platforms of choice in a consistent manner.

The ability to control how data is generated, enriched, and transmitted.

#2 in CNCF ✓

OpenTelemetry is the second most active project in CNCF behind only Kubernetes per CNCF DevStats.

It is also widely adopted by OSS projects, including Kubernetes, Istio, Prometheus, and Jaeger.

End User Adopted and Vendor Supported ✓

Many end users contribute to and have adopted OpenTelemetry in their environment.

All major observability vendors and cloud providers contribute to and support OpenTelemetry.

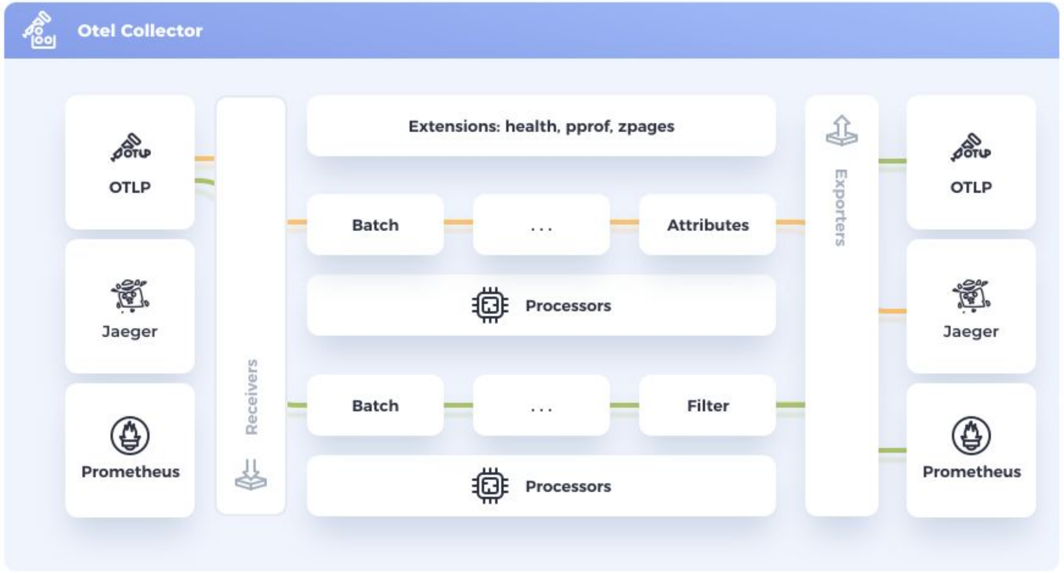
OpenTelemetry

OTel Me More

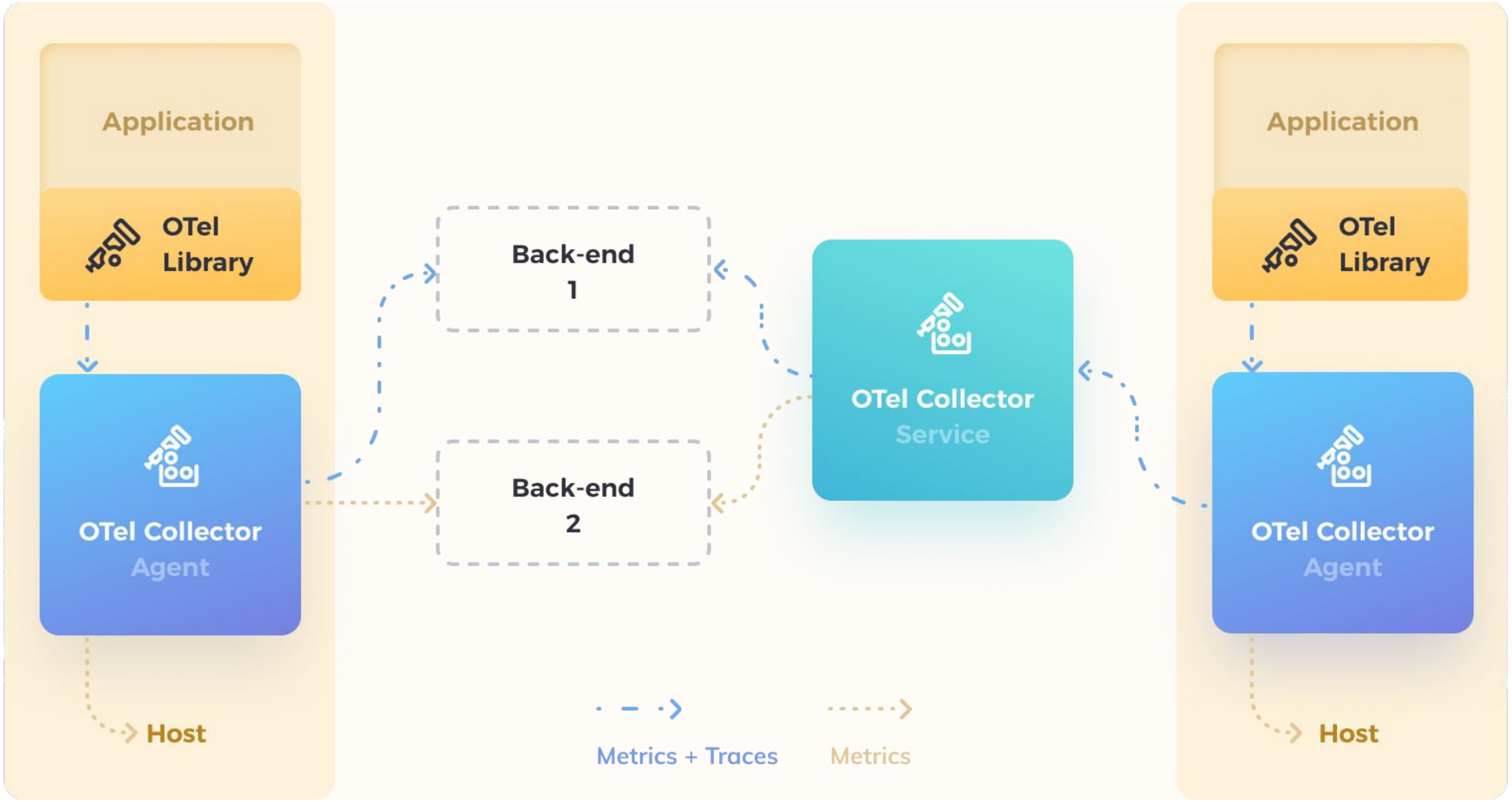
The OpenTelemetry Collector

What is the OpenTelemetry Collector?

The OpenTelemetry Collector offers a vendor-agnostic implementation of how to receive, process and export telemetry data. It removes the need to run, operate, and maintain multiple agents/collectors.
- https://opentelemetry.io/docs/collector/#introduction



OpenTelemetry



Reference architecture

Collector Packaging

Go

Go binaries with packaging for Linux, MacOS, and Windows.

https://github.com/open-telemetry/opentelemetry-collector-releases/releases

Docker

Docker containers available on DockerHub and ghcr.io.

https://opentelemetry.io/docs/collector/installation/#docker

Kubernetes (K8s)

A Helm chart and Operator are available.

https://opentelemetry.io/docs/kubernetes/helm/

Custom

Given Go is a compiled language, you can also create your own packaging.

OpenTelemetry

Collector Distributions

Core

OTel supported components, including the OTLP receiver and exporter.

https://github.com/open-telemetry/opentelemetry-collector/

Contrib

Community supported components, including various processors and vendor exporters. Most environments will require at least some of these components.

https://github.com/open-telemetry/opentelemetry-collector-contrib/

K8s

A Helm chart and Operator are available for K8s deployments, which includes K8s-specific receivers and processors.

https://opentelemetry.io/docs/kubernetes/helm/

Custom

Custom distributions can be created using the builder utility called ocb.

https://opentelemetry.io/docs/collector/custom-collector/

OpenTelemetry

OTel Me How

Collector Configuration

Collector Components

Receivers

How you get data in
(can be push or pull-based)

Processors

What you do to the data
(e.g. batch, metadata, etc.)

Exporters

How you get data out
(can be push or pull-based)

Extensions

Things done outside
processing data
(e.g. health check)

Connectors

A Connector is also a component
which is a Receiver _and_ Exporter.

How is the OTel Collector configured?

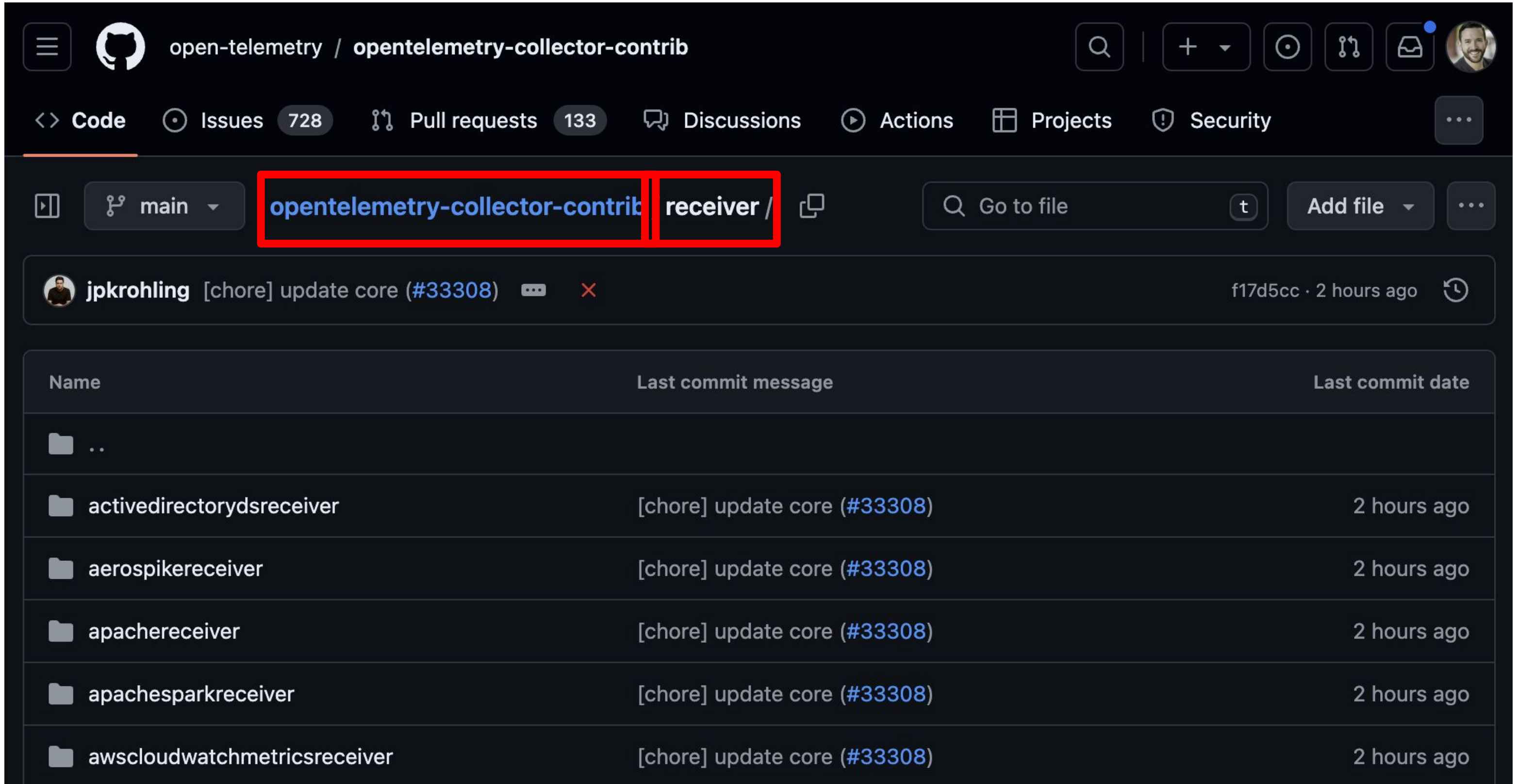
Two step YAML: 1) Define and configure components 2) Enable components

```yaml
receivers:
  hostmetrics:
    scrapers:
      cpu:
      memory:
  otlp:
    protocols:
      http:
exporters:
  zipkin:
    endpoint: "https://zipkin:9411/api/v2/spans"
  prometheus:
    endpoint: prometheus:9091
service:
  pipelines:
    metrics:
      receivers: [hostmetrics]
      exporters: [prometheus]
    traces:
      receivers: [otlp]
      exporters: [zipkin]
```

How to configure components. Many components come with default configuration baked in.

How to enable components. Note the order of processors matters!

Where do you find the configuration options?


open-telemetry / opentelemetry-collector-contrib

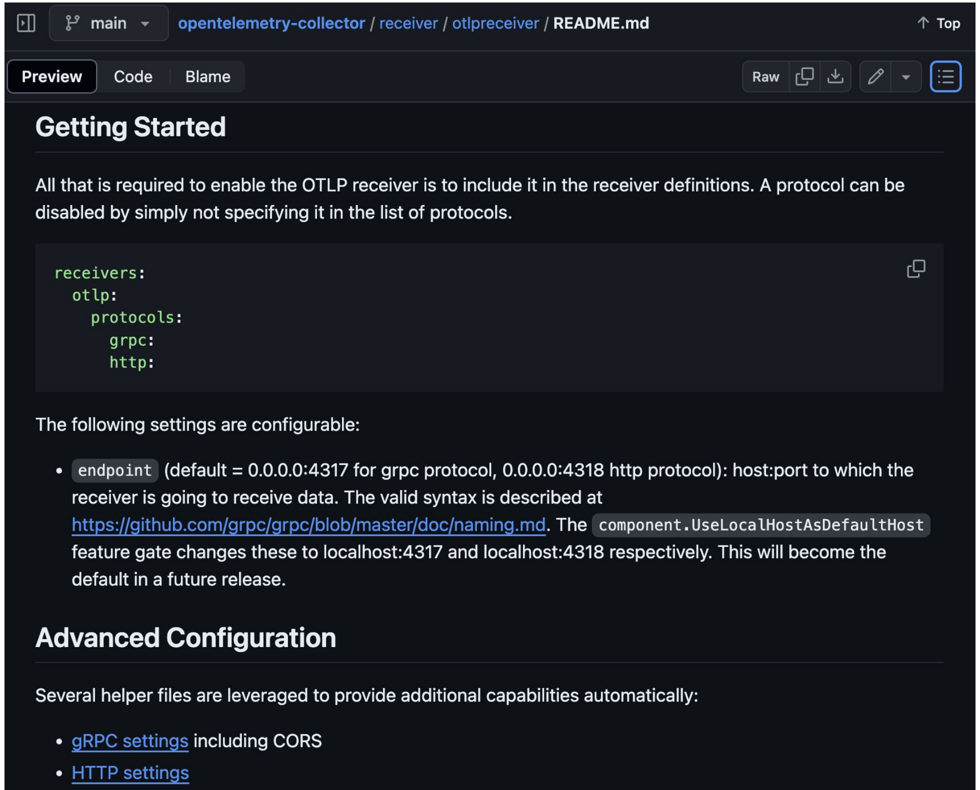
Code · Issues 728 · Pull requests 133 · Discussions · Actions · Projects · Security

main · opentelemetry-collector-contrib / receiver / · Go to file · Add file

jpkrohling [chore] update core (#33308) · f17d5cc · 2 hours ago

Name	Last commit message	Last commit date
..		
activedirectorydsreceiver	[chore] update core (#33308)	2 hours ago
aerospikereceiver	[chore] update core (#33308)	2 hours ago
apachereceiver	[chore] update core (#33308)	2 hours ago
apachesparkreceiver	[chore] update core (#33308)	2 hours ago
awscloudwatchmetricsreceiver	[chore] update core (#33308)	2 hours ago

Where do you find the configuration options?


main
opentelemetry-collector / receiver / otlpreceiver / README.md
Top
Preview Code Blame
Raw

Getting Started

All that is required to enable the OTLP receiver is to include it in the receiver definitions. A protocol can be disabled by simply not specifying it in the list of protocols.

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

The following settings are configurable:

- `endpoint` (default = 0.0.0.0:4317 for grpc protocol, 0.0.0.0:4318 http protocol): host:port to which the receiver is going to receive data. The valid syntax is described at https://github.com/grpc/grpc/blob/master/doc/naming.md. The `component.UseLocalHostAsDefaultHost` feature gate changes these to localhost:4317 and localhost:4318 respectively. This will become the default in a future release.

Advanced Configuration

Several helper files are leveraged to provide additional capabilities automatically:

- gRPC settings including CORS
- HTTP settings

OpenTelemetry

How else can the Collector be configured?

- Multiple, merged configurations

```
otelcol --config config_1.yaml --config config_2.yaml
```

- Environment variables

```
processors:
  attributes/example:
    actions:
      - key: ${env:DB_KEY}
        action: ${env:OPERATION}
```

- OpenTelemetry Transformation Language (OTTL)

```
traces:
  keep_keys(attributes, ["http.method", "http.status_code"])
```

~~OTel~~ OShow Me

Demo Time!

OTel Me Again

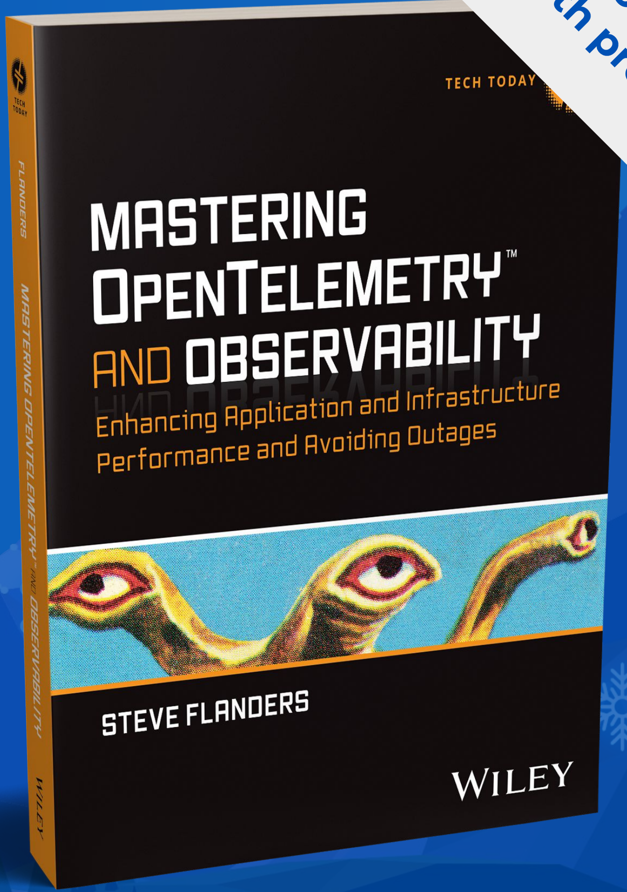
Where can I learn more?

Resources

- https://opentelemetry.io and https://opentelemetry.io/community/
- https://opentelemetry.io/docs/collector/quick-start/
- https://opentelemetry.io/docs/collector/configuration/
- https://github.com/open-telemetry/opentelemetry-collector/
- https://github.com/open-telemetry/opentelemetry-collector-contrib/
- https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/pkg/ottl/README.md
- https://github.com/open-telemetry/opentelemetry-collector-releases/releases/
- https://opentelemetry.io/docs/demo/

Thank You!

Questions?


TECH TODAY
MASTERING OPENTELEMETRY™ AND OBSERVABILITY
Enhancing Application and Infrastructure Performance and Avoiding Outages
STEVE FLANDERS
WILEY

35% off from Wiley (only)
With promo code: KUB35