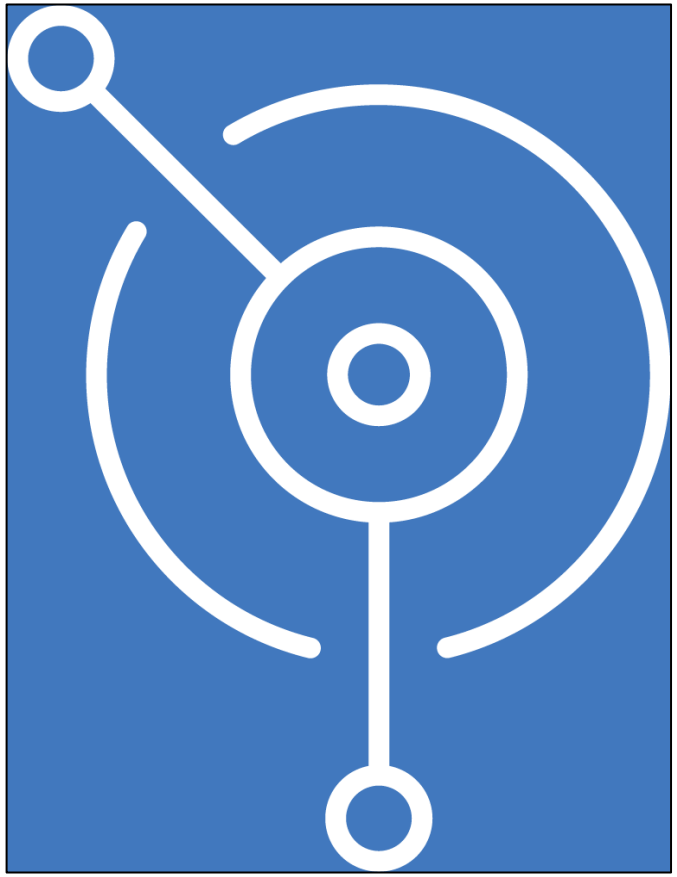
Integrating IBM MQ with 3 pillars of OpenTelemetry



Mark Taylor

MQ Ecosystem Developer

marke_taylor@uk.ibm.com
IBM Hursley

What is OpenTelemetry?

- An observability framework with a set of APIs, SDKs and integrations designed to create and manage trace, metrics, and log data
- It is vendor and tool agnostic - it can be used with a broad variety of observability solutions:
  - Commercial offerings such as IBM Instana
  - Open-source tools like Jaeger and Prometheus
- It is NOT a specific product

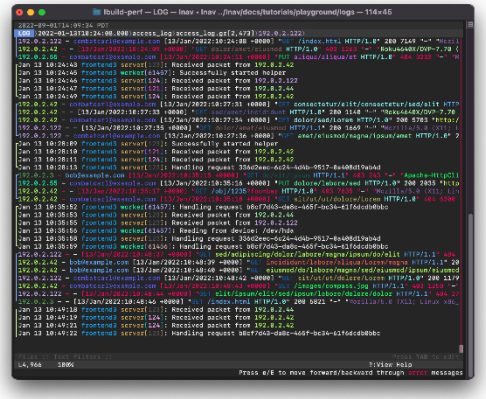
Benefits:

Diagnose problems across multiple systems fast, using data to pinpoint where issues are

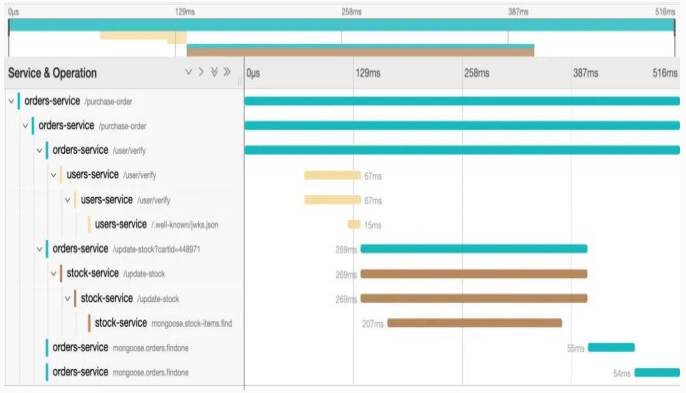
Gain insights to optimise performance end-to-end, ultimately improving customer experience

Futureproof – open and likely to become a cross-industry de facto standard for observability

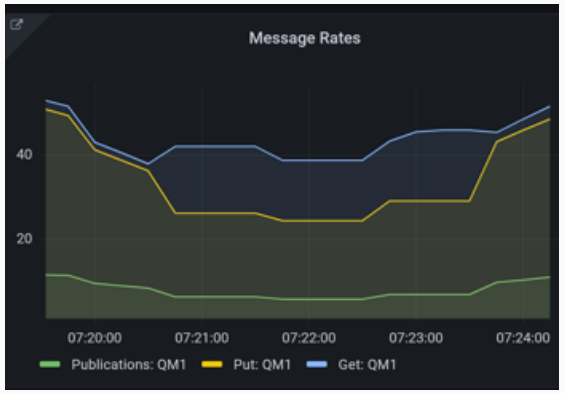
CLOUD NATIVE COMPUTING FOUNDATION



Logs



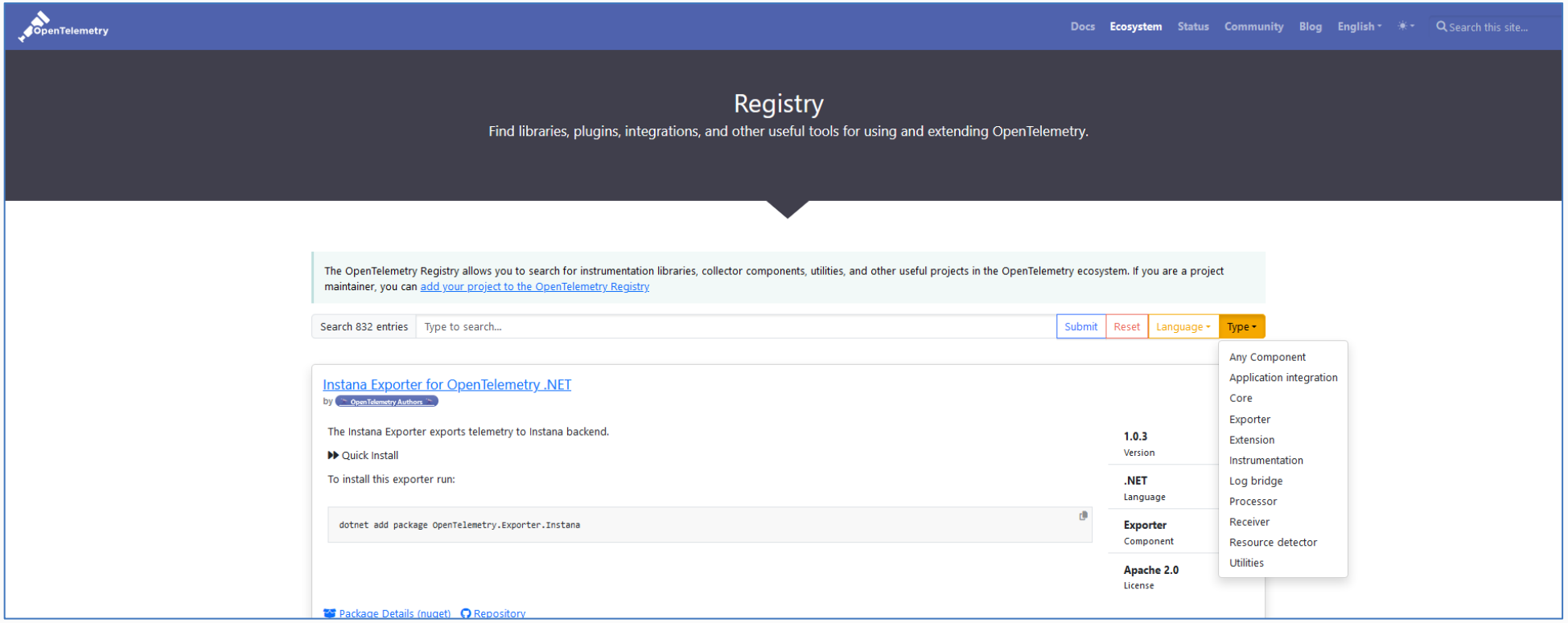
Traces



Metrics

OpenTelemetry Registry

Find your integrations here



OTel Collector

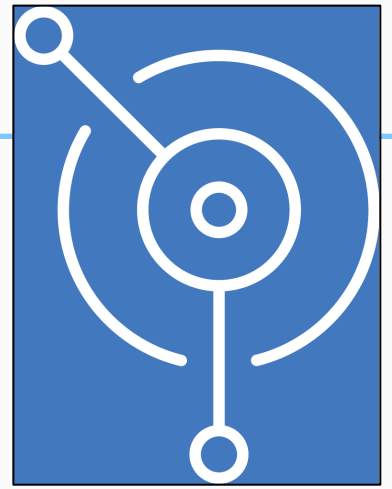
A proxy/converter for many protocols

Can apply filters early in flow

If your preferred tools do not have native OTel support, can probably use Collector to transform

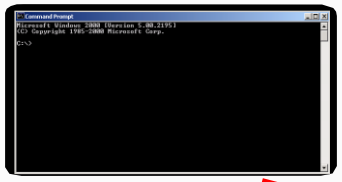
Various deployment models

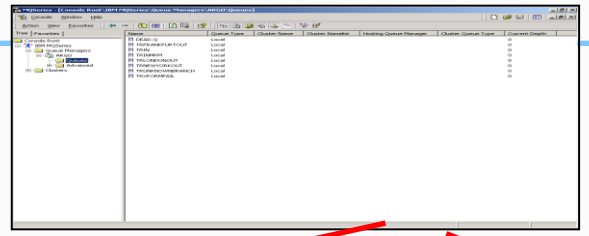
Perhaps the only OTel-provided program you would run



Overview: MQ admin interfaces

MQ Administration





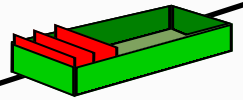
MQSC

OS

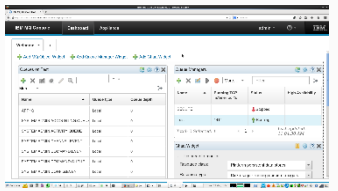
PCF

OS configuration

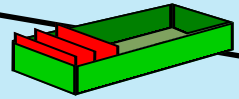
ini files



REST



Queue Manager



Events (async)

Status (polled)

Stats (polled/async)

Observability Data

Log files

These are all documented and supported interfaces – it has encouraged a huge management ecosystem over many years

Event Message - PCF

```
****   Message length - 300 of 300 bytes ***

00000000:  0000 0007 0000 0024 0000 0003 0000 0063 '.......$.......c'
00000010:  0000 0001 0000 0001 0000 0000 0000 096C '...............l'
00000020:  0000 0002 0000 0014 0000 0010 0000 1F41 '...............A'
00000030:  0000 0004 0000 0004 0000 0020 0000 0BE5 '........... ...å'
00000040:  0000 0333 0000 000C 6D65 7461 796C 6F72 '...3....metaylor'
00000050:  2020 2020 0000 0003 0000 0010 0000 03F3 '    ...........ó'
00000060:  0000 0001 0000 0004 0000 0044 0000 0BE7 '...........D...ç'
00000070:  0000 0333 0000 0030 5638 3030 335F 4120 '...3...0V8003_A '
00000080:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000090:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
000000A0:  2020 2020 2020 2020 0000 0003 0000 0010 '        ........'
000000B0:  0000 03FD 0000 005A 0000 0014 0000 0010 '...ý...Z........'
000000C0:  0000 1F42 0000 0004 0000 0004 0000 0018 '...B............'
000000D0:  0000 0BFB 0000 0000 0000 0001 5800 0000 '...û........X...'
000000E0:  0000 0003 0000 0010 0000 03F8 0000 0001 '...........ø....'
000000F0:  0000 0006 0000 0024 0000 0BF9 0000 0000 '.......$...ù....'
00000100:  0000 0001 0000 0008 6D65 7461 796C 6F72 '........metaylor'
00000110:  0000 0000 0000 0005 0000 0018 0000 045C '...............\'
00000120:  0000 0002 0000 000B 0000 0009           '............    '
```

Event Message - decoded

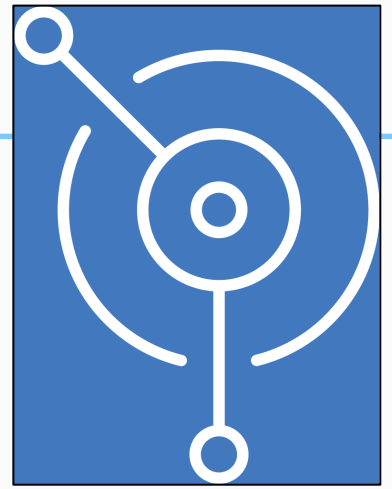
```
Event Type                    : Command Event
Reason                        : Command MQSC
Event created                 : 2015/06/03 13:28:20.51 GMT
Correlation ID : 414D5120563830303335F41202020202020556F00F120001E05
  COMMAND CONTEXT
    Event User Id             : metaylor
    Event Origin              : Console
    Event Queue Mgr           : V8003_A
    Command                   : Set Auth Rec
  COMMAND DATA
    Auth Profile Name         : X
    Object Type               : Queue
    Principal Entity Names    : metaylor
    Auth Add Auths            : Output
                              : Input
```

This one is useful for admin audit trail

Metrics Message - PCF

```
00000000:  1500 0000 2400 0000 0300 0000 0000 0000          '....$..........'
00000010:  0100 0000 0100 0000 0000 0000 0000 0000          '................'
00000020:  0900 0000 0400 0000 1800 0000 DF07 0000          '................'
00000030:  3303 0000 0300 0000 514D 3100 0300 0000          '3.......QM1.....'
00000040:  1000 0000 4703 0000 0300 0000 0300 0000          '....G...........'
00000050:  1000 0000 4803 0000 0400 0000 1700 0000          '....H...........'
00000060:  1800 0000 4D03 0000 0100 0000 8096 9800          '....M...........'
00000070:  0000 0000 0400 0000 1C00 0000 E007 0000          '................'
00000080:  3303 0000 0500 0000 4150 502E 3045 5452          '3.......APP.0ETR'
00000090:  1700 0000 1800 0000 0000 0000 0000 0000          '................'
000000A0:  0000 0000 0000 0000 1700 0000 1800 0000          '................'
000000B0:  0100 0000 0100 0000 0000 0000 0000 0000          '................'
000000C0:  1700 0000 1800 0000 0200 0000 0200 0000          '................'
000000D0:  0000 0000 0000 0000 1700 0000 1800 0000          '................'
000000E0:  0300 0000 0300 0000 C200 0000 0000 0000          '................'

==> APP.0
Publication received PutDate:20240307 PutTime:08213709 Interval:8.040 seconds
APP.0       messages expired 0
APP.0       queue purged count 0
APP.0       average queue time 0 uSec
APP.0       Queue depth 29
```



OpenTelemetry: Logs

MQ Events and Error Logs: No-new-code option

- MQ error logs can be written in JSON format
  - And then ingested directly through OTel Collector
  - Local files or via syslog

```
DiagnosticMessages:
   Service = File
   Name = JSONLogs
   Format = json
   FilePrefix = AMQERR
```

- Event messages
  - MQ sample amqsevt can format them in JSON
  - Then also pass via the Collector

- Examples here go as far as the Collector
  - You configure exporters to send to real analysis tools

OTel Collector with an MQ Error Log entry

```
filelog/mqlog:
  include:
  - /var/mqm/qmgrs/APIX/errors/AMQERR01.js
  start_at: beginning
  operators:
    - type: json_parser
      timestamp:
        parse_from: attributes.ibm_datetim
        layout: "%Y-%m-%dT%H:%M:%S.%LZ"
      severity:
        parse_from: attributes.loglevel
```

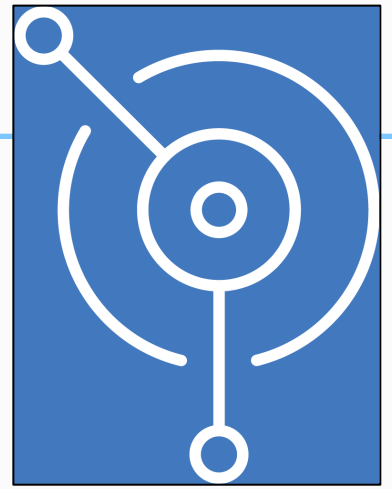
```
ScopeLogs #0
ScopeLogs SchemaURL:
InstrumentationScope
LogRecord #0
ObservedTimestamp: 2024-03-10 10:26:45.226965363 +0000 UTC
Timestamp: 2024-03-10 10:26:45.189 +0000 UTC
SeverityText: WARNING
SeverityNumber: Warn(13)
Body: Str({"ibm_messageId":"AMQ8077W","ibm_arithInsert1":0,"ibm_arithInsert2":0,"ibm_commentInsert1":"mqgu
est","ibm_commentInsert2":"APIX [qmgr]","ibm_commentInsert3":"connect","ibm_datetime":"2024-03-10T10:26:45
.189Z","ibm_serverName":"APIX","type":"mq_log","host":"klein","loglevel":"WARNING","module":"amqzfubx.c:16
77","ibm_sequence":"1710066405_189924116","ibm_qmgrId":"APIX_2022-07-20_08.35.15","ibm_processId":"2318101
","ibm_threadId":"12","ibm_version":"9.3.5.0","ibm_processName":"amqzlaa0","ibm_userName":"metaylor","ibm_
installationName":"Installation1","ibm_installationDir":"/opt/mqm","message":"AMQ8077W: Entity 'mqguest' h
as insufficient authority to access object APIX [qmgr]."})
Attributes:
     -> ibm_arithInsert1: Double(0)
     -> ibm_commentInsert2: Str(APIX [qmgr])
     -> ibm_serverName: Str(APIX)
     -> ibm_messageId: Str(AMQ8077W)
     -> ibm_commentInsert1: Str(mqguest)
     -> ibm_qmgrId: Str(APIX_2022-07-20_08.35.15)
     -> host: Str(klein)
     -> module: Str(amqzfubx.c:1677)
     -> ibm_datetime: Str(2024-03-10T10:26:45.189Z)
     -> ibm_userName: Str(metaylor)
     -> ibm_sequence: Str(1710066405_189924116)
     -> ibm_version: Str(9.3.5.0)
     -> type: Str(mq_log)
     -> ibm_processId: Str(2318101)
     -> log.file.name: Str(AMQERR01.json)
     -> ibm_arithInsert2: Double(0)
     -> ibm_installationDir: Str(/opt/mqm)
     -> ibm_processName: Str(amqzlaa0)
     -> ibm_commentInsert3: Str(connect)
     -> loglevel: Str(WARNING)
     -> ibm_installationName: Str(Installation1)
     -> message: Str(AMQ8077W: Entity 'mqguest' has insufficient authority to access object APIX [qmgr].)
     -> ibm_threadId: Str(12)
```

```
$ amqsevt -b -m QM1 -w 1 -o json_compact > ~/Docker/otel/pipedir/npipe
$
```

```
namedpipe/mqevt:
  path: /pipedir/npipe
  operators:
    - type: json_parser
      timestamp:
        parse_from: attributes.eventCreation.
        layout: "%Y-%m-%dT%H:%M:%S.%LZ"
```

```
Trace ID:
Span ID:
Flags: 0
LogRecord #78
ObservedTimestamp: 2024-03-07 07:20:02.206248405 +0000 UTC
Timestamp: 2024-03-07 05:14:43 +0000 UTC
SeverityText:
SeverityNumber: Unspecified(0)
Body: Str({ "eventSource" : { "objectName": "SYSTEM.ADMIN.PERFM.EVENT",
      "objectType" : "Queue",                  "queueMgr" : "QM1"}, "eventType" :
 {     "name" : "Perfm Event",     "value" : 45   }, "eventReason" : {     "name" :
 "Queue Depth High",     "value" : 2224   }, "eventCreation" : {     "timeStamp"  :
 "2024-03-07T05:14:43Z",     "epoch"      : 1709788483   }, "eventData" : {   "queu
eMgrName" : "QM1",   "baseObjectName" : "APP.0",   "timeSinceReset" : 129,   "highQ
ueueDepth" : 160,   "msgEnqCount" : 337,   "msgDeqCount" : 217 } })
Attributes:
     -> eventReason: Map({"name":"Queue Depth High","value":2224})
     -> eventCreation: Map({"epoch":1709788483,"timeStamp":"2024-03-07T05:14:43Z"})
     -> eventData: Map({"baseObjectName":"APP.0","highQueueDepth":160,"msgDeqCount"
:217,"msgEnqCount":337,"queueMgrName":"QM1","timeSinceReset":129})
     -> eventSource: Map({"objectName":"SYSTEM.ADMIN.PERFM.EVENT","objectType":"Que
ue","queueMgr":"QM1"})
     -> eventType: Map({"name":"Perfm Event","value":45})
Trace ID:
Span ID:
Flags: 0
```

OTel Collector with the MQ Event formatter



OpenTelemetry: Metrics

8-year path to OTel metrics in an Open Source tool

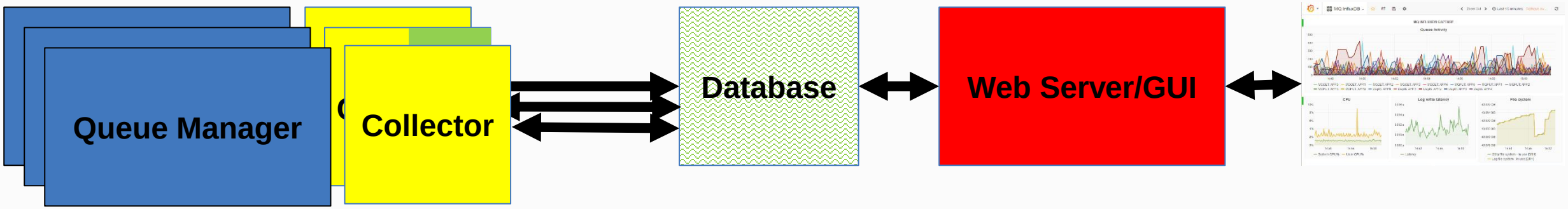
- 2016 – project to show how MQ could be used with a variety of "cloud" tools
  - Security, deployment, HA, monitoring etc

- For the "monitoring" topic...
- Wrote Go APIs and programs to send subset of MQ metrics to Prometheus
  - Released on github under Apache license
  - UNSUPPORTED, AS-IS code.
- Enough user interest to enhance it: function and databases
  - Now collects all metrics from any MQ platform
  - Outputs to Prometheus, Influx, OpenTSDB, collectd, Cloudwatch, JSON ...
- Typical metrics: queue depth, MQPUTs, channel messages, pageset % usage
  - More than 200 metrics available

Architecture

- Collector/Reader/Exporter runs alongside queue manager
  - Or using remote client connection if Go programs cannot run locally

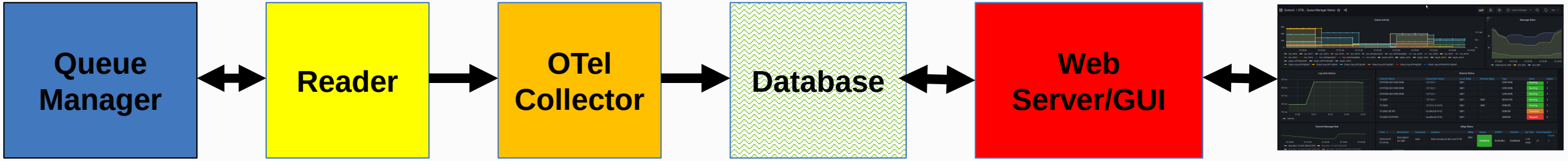
- Extracts MQ metrics at intervals through PCF messages
  - Record in database



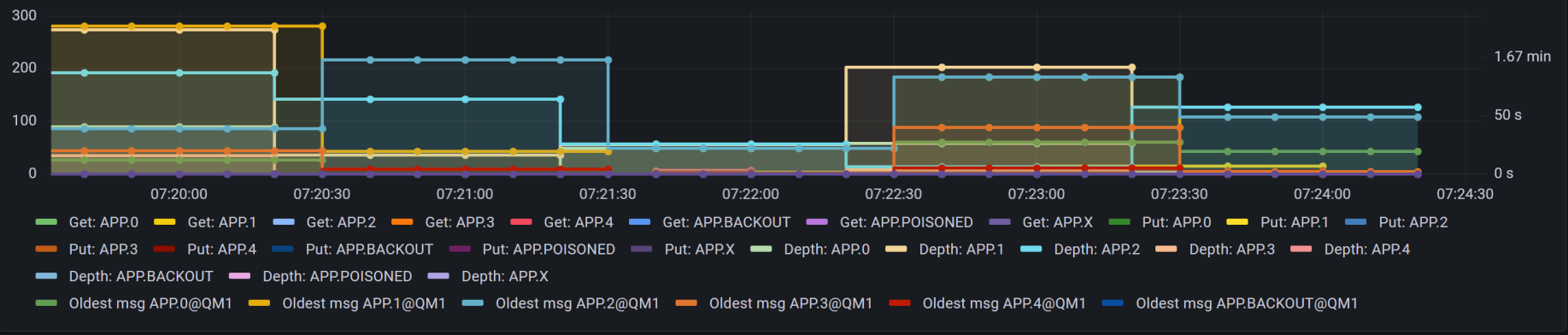
Four equivalent Grafana dashboards

2024: OpenTelemetry Metrics

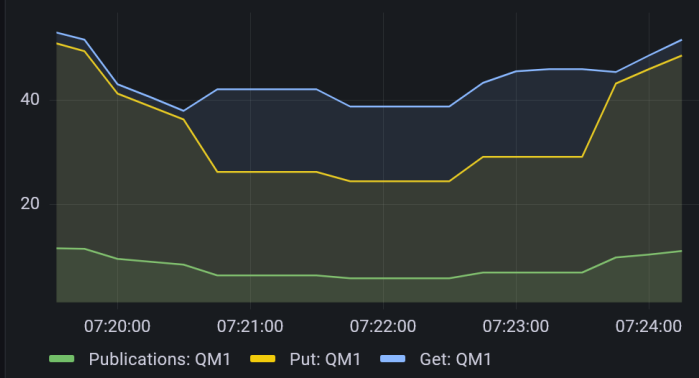
- Same pattern as other database collectors
- Allows stdout, OTLP/gRPC, OTLP/http exporters
  - Sends directly into OTel Collector or many other tools
  - No need to go via the MQ Prometheus exporter
- Counters and Gauges differentiated



Queue Activity



Message Rates



Log write latency

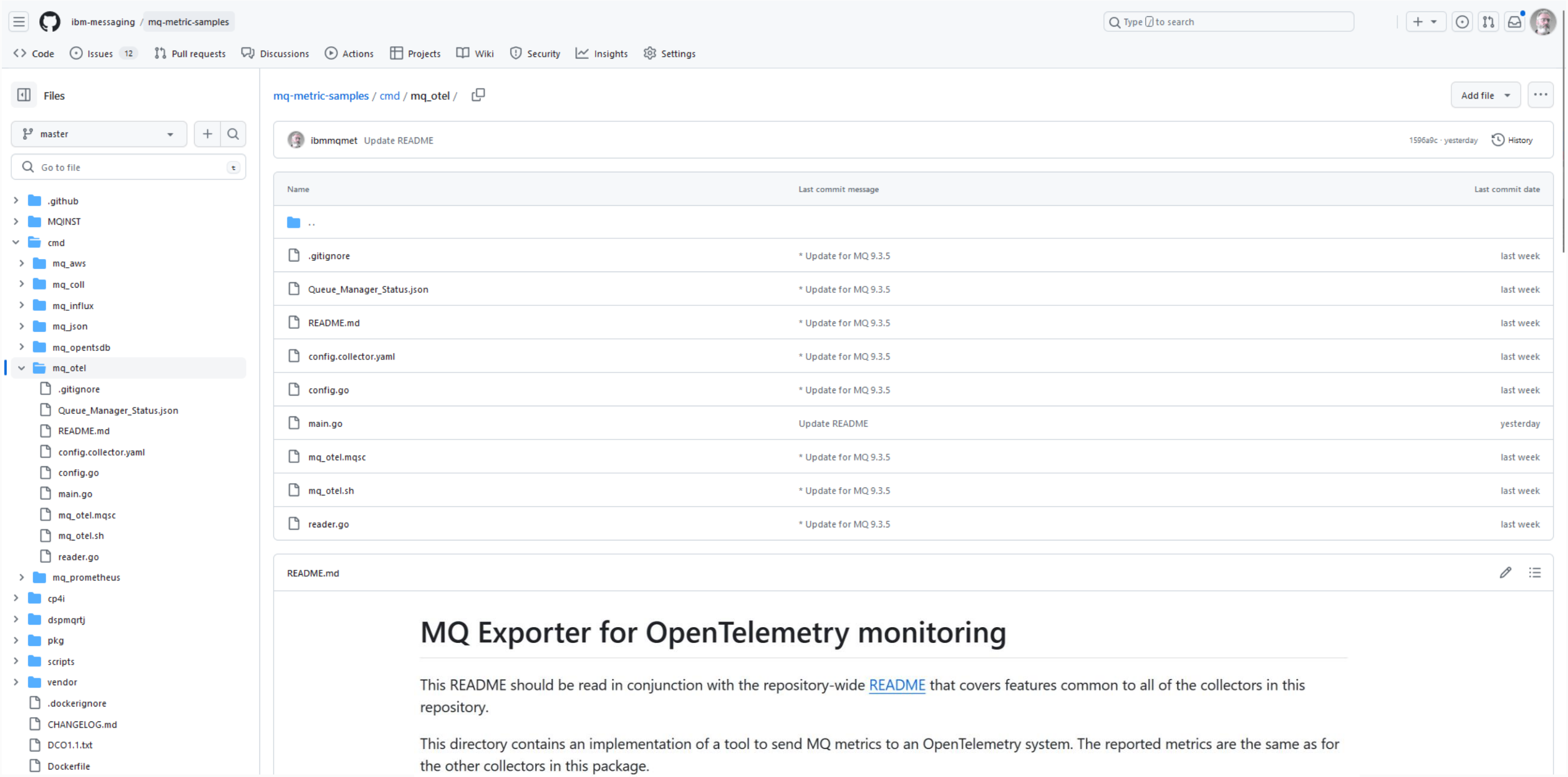
Channel Status

Channel Name	Connection Name	Local QMgr	Remote QMgr	Type	State	Status
SYSTEM.DEF.SVRCONN	127.0.0.1	QM1	-	SVRCONN	Running	3
SYSTEM.DEF.SVRCONN	127.0.0.1	QM1	-	SVRCONN	Running	3
SYSTEM.DEF.SVRCONN	127.0.0.1	QM1	-	SVRCONN	Running	3
TO.QM1	127.0.0.1	QM1	QM2	RECEIVER	Running	3
TO.QM2	127.0.0.1(1415)	QM1	QM2	SENDER	Running	3
TO.QM2.RETRY	localhost(1415)	QM1	-	SENDER	Transition	5
TO.QM2.STOPPED	localhost(1415)	QM1	-	SENDER	Stopped	6

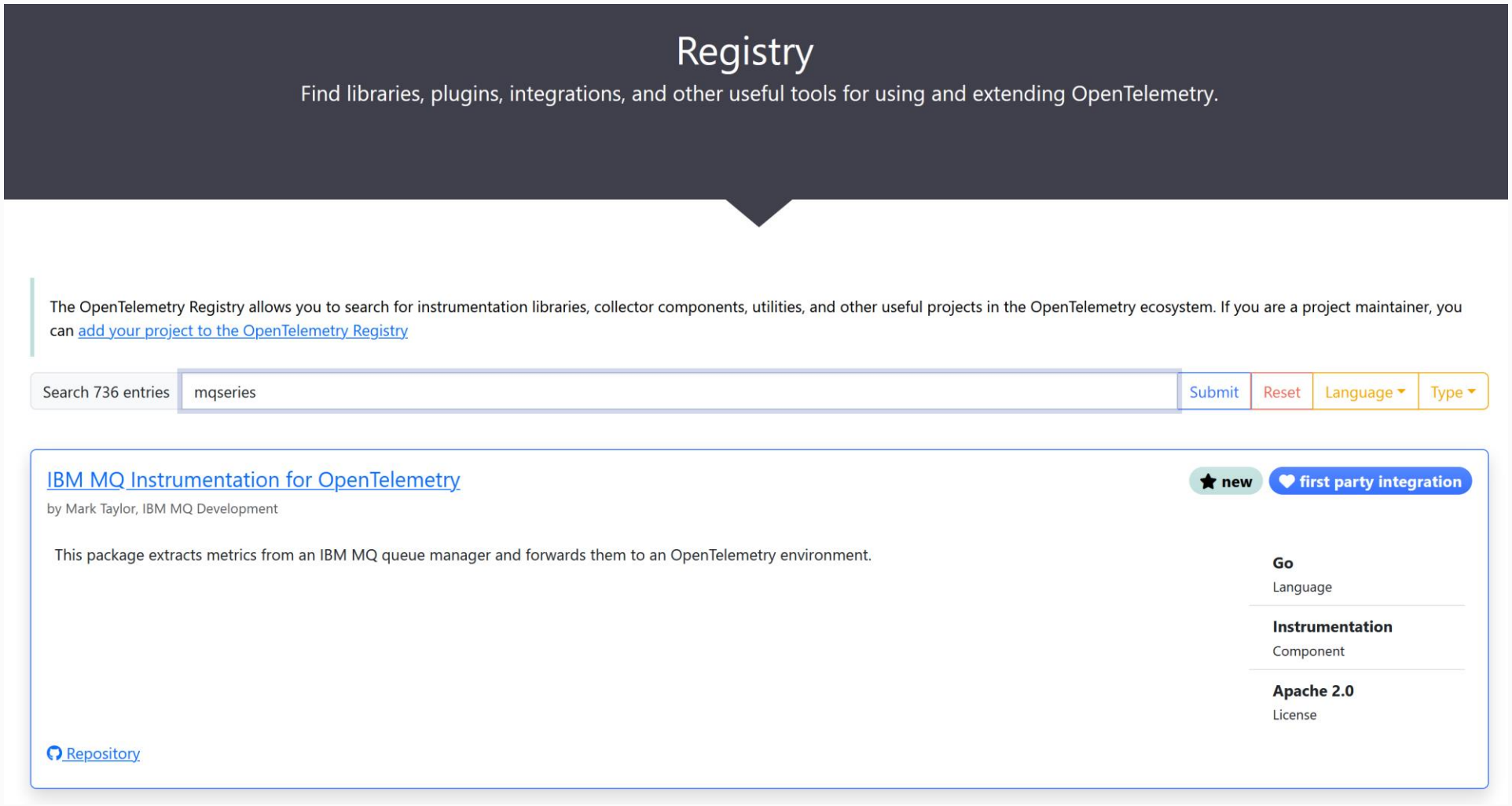
Channel Message Rate

QMgr Status

Time	description	hostname	instance	QMgr	Status	CHINIT	Cmd Svr	Up Time	Conn Count	Lsr Count
2024-03-07 07:24:26	Description for QM1	localhost	localhost:9159	QM1	RUNNING	RUNNING	RUNNING	1.09 week	27	1


ibm-messaging / mq-metric-samples
Type / to search
Code
Issues 12
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
Files
master
Go to file
.github
MQINST
cmd
mq_aws
mq_coll
mq_influx
mq_json
mq_opentsdb
mq_otel
.gitignore
Queue_Manager_Status.json
README.md
config.collector.yaml
config.go
main.go
mq_otel.mqsc
mq_otel.sh
reader.go
mq_prometheus
cp4i
dspmqrtj
pkg
scripts
vendor
.dockerignore
CHANGELOG.md
DCO1.1.txt
Dockerfile
mq-metric-samples / cmd / mq_otel /
Add file
ibmmqmet Update README
1596a9c · yesterday
History
Name
Last commit message
Last commit date
..
.gitignore * Update for MQ 9.3.5 last week
Queue_Manager_Status.json * Update for MQ 9.3.5 last week
README.md * Update for MQ 9.3.5 last week
config.collector.yaml * Update for MQ 9.3.5 last week
config.go * Update for MQ 9.3.5 last week
main.go Update README yesterday
mq_otel.mqsc * Update for MQ 9.3.5 last week
mq_otel.sh * Update for MQ 9.3.5 last week
reader.go * Update for MQ 9.3.5 last week
README.md
MQ Exporter for OpenTelemetry monitoring
This README should be read in conjunction with the repository-wide README that covers features common to all of the collectors in this repository.
This directory contains an implementation of a tool to send MQ metrics to an OpenTelemetry system. The reported metrics are the same as for the other collectors in this package.

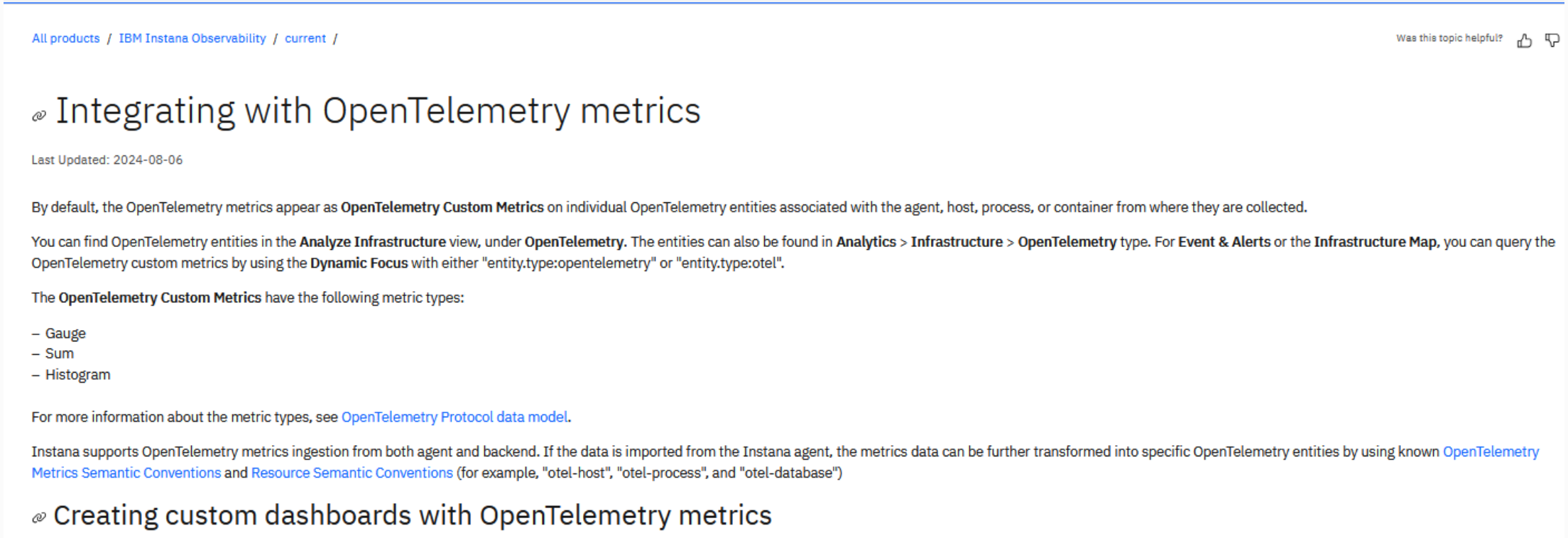
OTel Registry entry



https://opentelemetry.io/ecosystem/registry/?s=mq

Other tools

- Commercial tools exist that can collect similar metrics and emit in OTel format
  - Including Instana



All products / IBM Instana Observability / current /

Was this topic helpful?

Integrating with OpenTelemetry metrics

Last Updated: 2024-08-06

By default, the OpenTelemetry metrics appear as **OpenTelemetry Custom Metrics** on individual OpenTelemetry entities associated with the agent, host, process, or container from where they are collected.

You can find OpenTelemetry entities in the **Analyze Infrastructure** view, under **OpenTelemetry**. The entities can also be found in **Analytics** > **Infrastructure** > **OpenTelemetry** type. For **Event & Alerts** or the **Infrastructure Map**, you can query the OpenTelemetry custom metrics by using the **Dynamic Focus** with either "entity.type:opentelemetry" or "entity.type:otel".

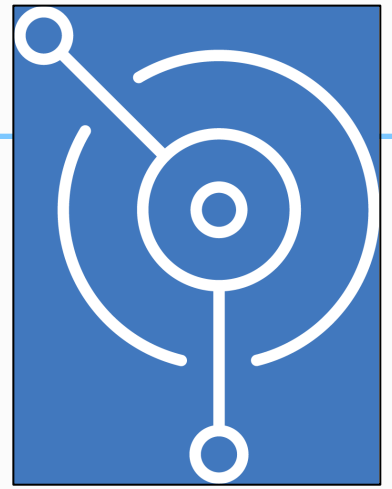
The **OpenTelemetry Custom Metrics** have the following metric types:

- Gauge
- Sum
- Histogram

For more information about the metric types, see OpenTelemetry Protocol data model.

Instana supports OpenTelemetry metrics ingestion from both agent and backend. If the data is imported from the Instana agent, the metrics data can be further transformed into specific OpenTelemetry entities by using known OpenTelemetry Metrics Semantic Conventions and Resource Semantic Conventions (for example, "otel-host", "otel-process", and "otel-database")

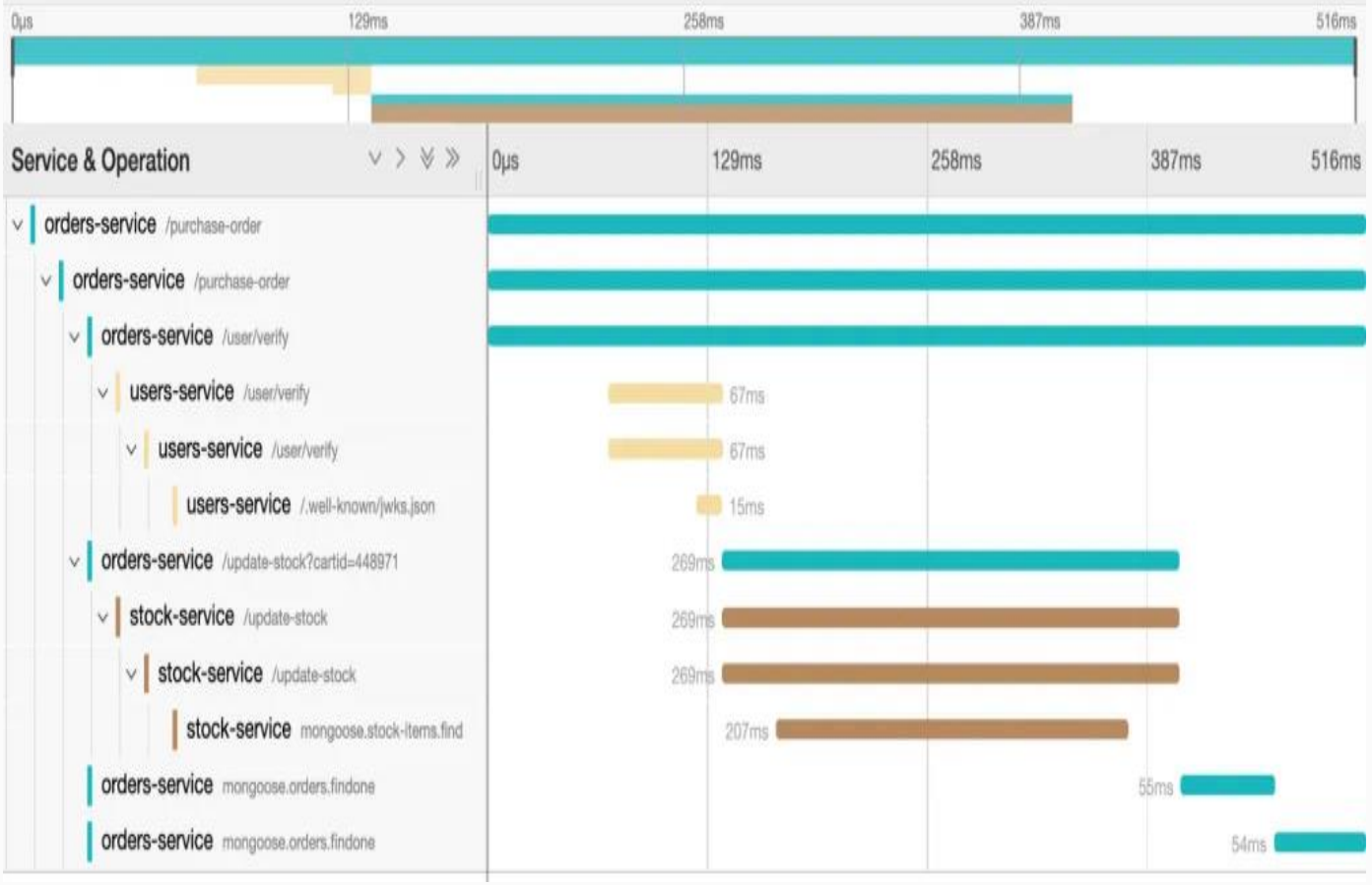
Creating custom dashboards with OpenTelemetry metrics



OpenTelemetry: Tracing

What is Distributed Tracing

- A technique for tracking and monitoring traces across multiple servers
- Visualize and understand the flow of a request as it traverses through different services.



- Key Components:
  - Trace: A collection of spans representing a single request or transaction.
  - Span: A single unit of work within a trace, representing a specific operation.

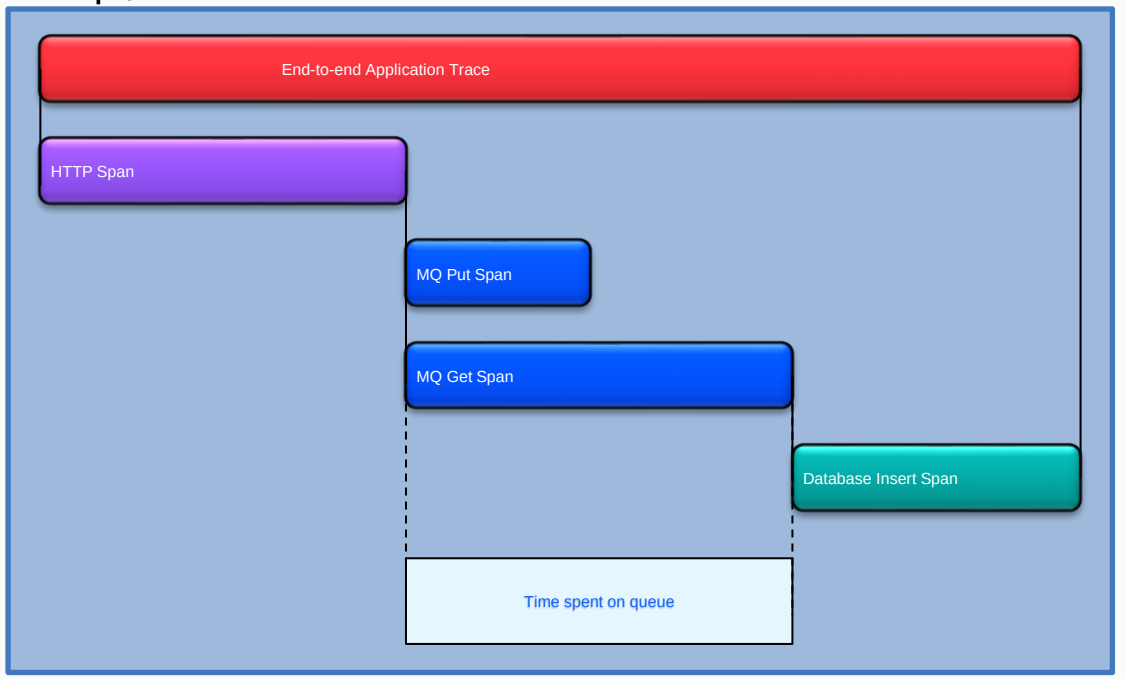
- Do not confuse with MQ service traces
  - From strmqtrc

Tracing for MQ: product extension

- Instana-provided API Exit
  - Spans for entry/exit to the MQ network and processing applications
  - Adds properties to messages passing through the system

- Reports in either Instana or OTel (via gRPC or http) format

- MQ users entitled to download/install the exit
  - No other Instana entitlement
  - Use with OTel tracing without needing Instana



MQ Configuration

/var/mqm/mqs.ini

all queue managers

```
ApiExitCommon:
   Sequence=100
   Function=EntryPoint
   Module=/var/mqm/exits64/mqtracingexit
   Name=TracingApiExit
```

/var/mqm/exits64/mqtracingexit.conf

OpenTelemetry Collector

```
LOG_LEVEL="info"          # Log level of the tracing user exit
SPAN_FORMAT="otel"        # Default value is "instana", but you can change it to "otel"
MONITOR_LEVEL="normal" # The tracing level of the queue manager(s
IBMMQ_DEST_MONITOR_LEVEL_NORMAL = "^DEV.*"      # The regex for message destinations
OTLP_EXPORTER_GRPC_ENDPOINT = "localhost:4317" # The grpc endpoint to which span data is sent
```

OTel Collector showing Trace

```
Resource SchemaURL:
Resource attributes:
     -> telemetry.sdk.version: Str(1.0.1)
     -> telemetry.sdk.name: Str(opentelemetry)
     -> telemetry.sdk.language: Str(cpp)
     -> service.name: Str(INSTANA)
ScopeSpans #0
ScopeSpans SchemaURL:
InstrumentationScope klein-INSTANA 3.1.8-202312260256-1780933(ibmmq-tracing)
Span #0
    Trace ID       : 00000000000000009e91b299d5322006
    Parent ID      :
    ID             : 9e91b299d5322006
    Name           : PUT INSTTEST
    Kind           : Consumer
    Start time     : 2024-03-09 09:25:28.036647 +0000 UTC
    End time       : 2024-03-09 09:25:28.040529 +0000 UTC
    Status code    : Unset
    Status message :
Attributes:
     -> messaging.destination: Str(INSTTEST)
     -> messaging.ibmmq.buffer_length: Str(49)
     -> messaging.ibmmq.completion_code: Str(0)
     -> messaging.ibmmq.destination.type: Str(queue)
     -> messaging.ibmmq.exit.appl_name: Str(myput)
     -> messaging.ibmmq.exit.appl_type: Str(6)
     -> messaging.ibmmq.exit.channel_name: Str()
     -> messaging.ibmmq.exit.connection_name: Str()
     -> messaging.ibmmq.exit.process_id: Str(1499127)
     -> messaging.ibmmq.exit.security_id: Str(0000000000000000000000000000000000000000000000000000000000000000
00000000000000000000)
                                                                                43,1          6%
```

ACE Configuration

server.conf.yaml

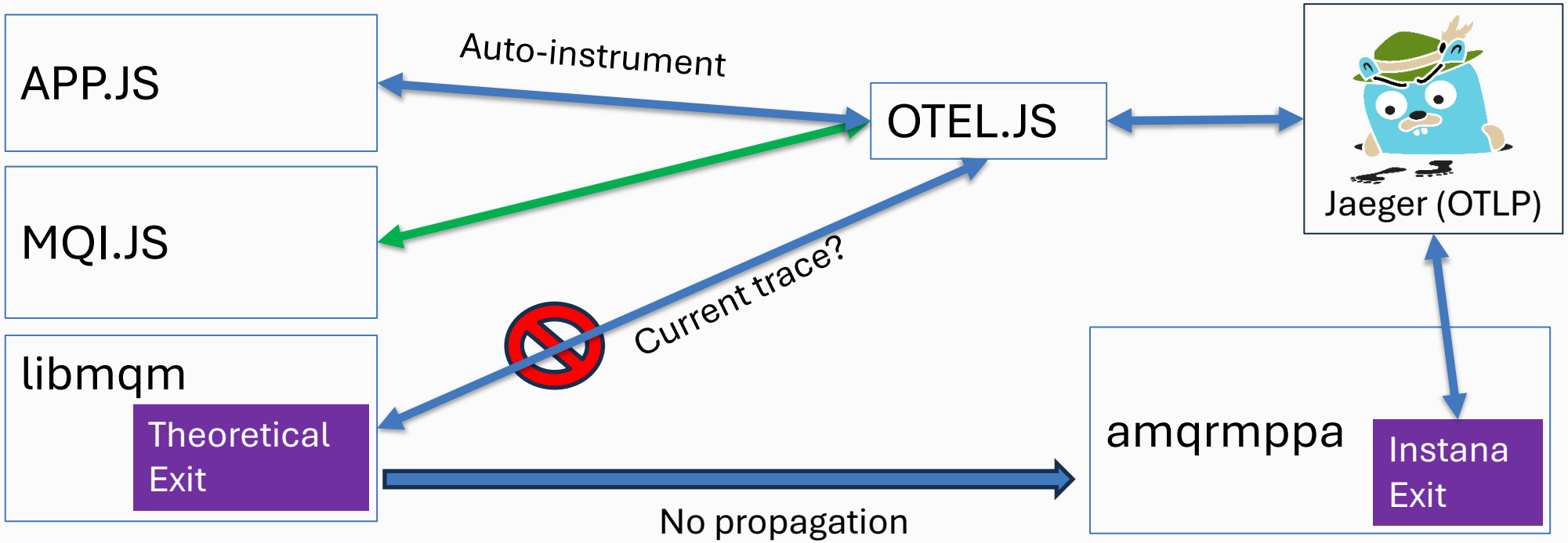
```
ResourceManagers:
  OpenTelemetryManager:
    openTelemetryEnabled: true
    openTelemetryServiceName: ''
    openTelemetryHostName: ''
    exporterOtlpGrpcEndpoint: 'localhost:4317'
    exporterOtlpHttpUrl: ''

    openTelemetryTruststoreType: ''
    openTelemetryTruststoreFile:
    openTelemetryTruststorePass: ''
    openTelemetryTrustAlias: ''
    openTelemetrySpanProcessor: 'batch'
    openTelemetryBatchSpanOptions: ''
```

Context Propagation

- To be useful, tracing needs to be available throughout an application stack and related components
- OTel implements tracing via language-specific SDKs
  - Some of which can “auto-instrument” common packages
    - For example, most web-server/http stacks have a way to get OTel traces captured
  - The Java/JMS stack can be automatically monitored
  - Applications might need updates to emit traces
- Propagating trace context through MQ (non-JMS) requires work
  - ACE can set standard properties on the message that the Instana exit recognises
  - Without those properties, the Instana reports are isolated from other activity
- I’ve been working on mechanisms for near-automatic propagation from several environments into MQ

Problem: Cross-language context propagation



Solution: an implementation per binding

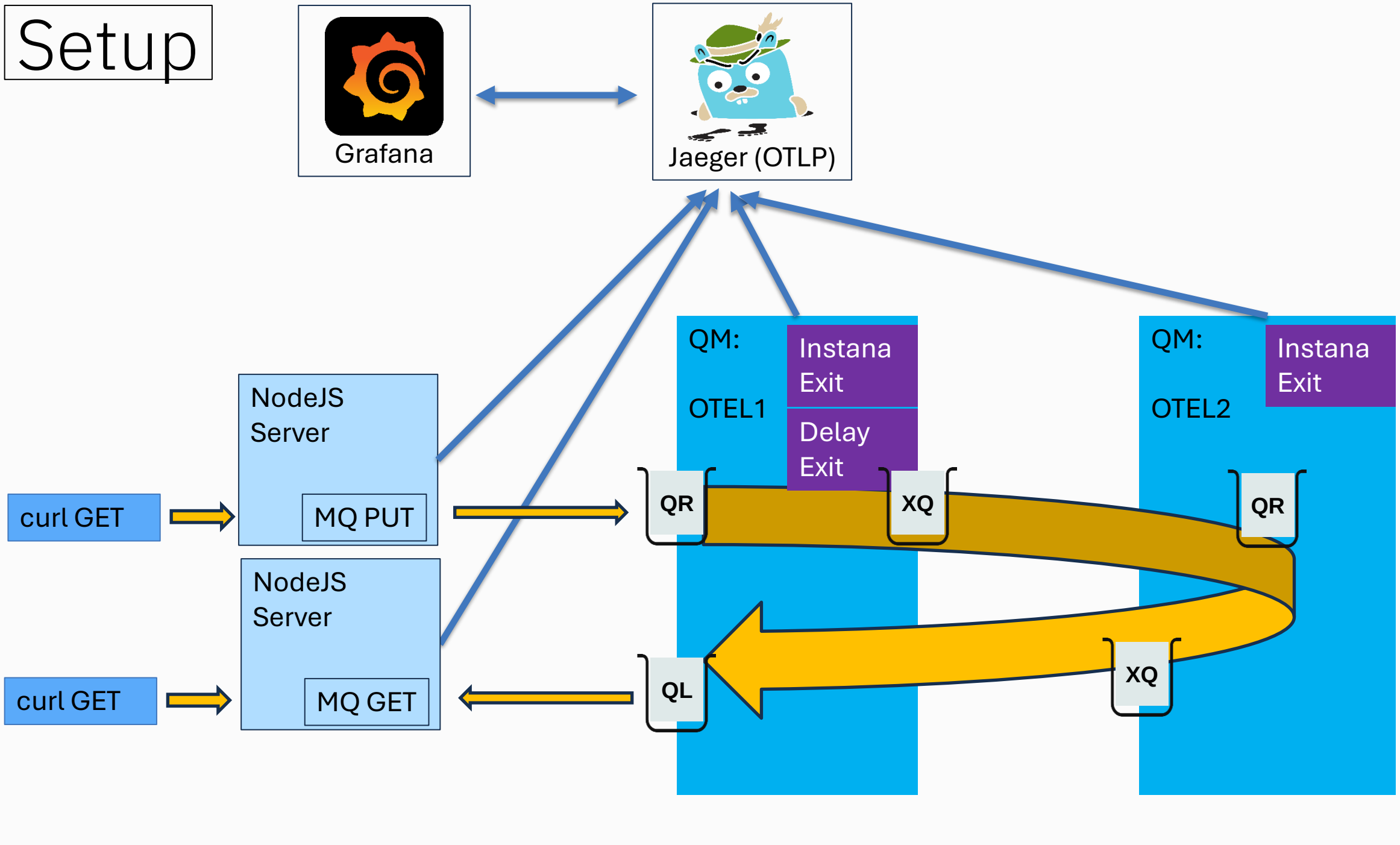
- Versions for NodeJS and Go for release with 9.4.1 in respective repositories

- MQ wrapper gets trace context, converting to properties during MQPUT
  - Which Instana exit recognises and reports during its span emissions
  - Does not emit its own span: client/server transits are not traced

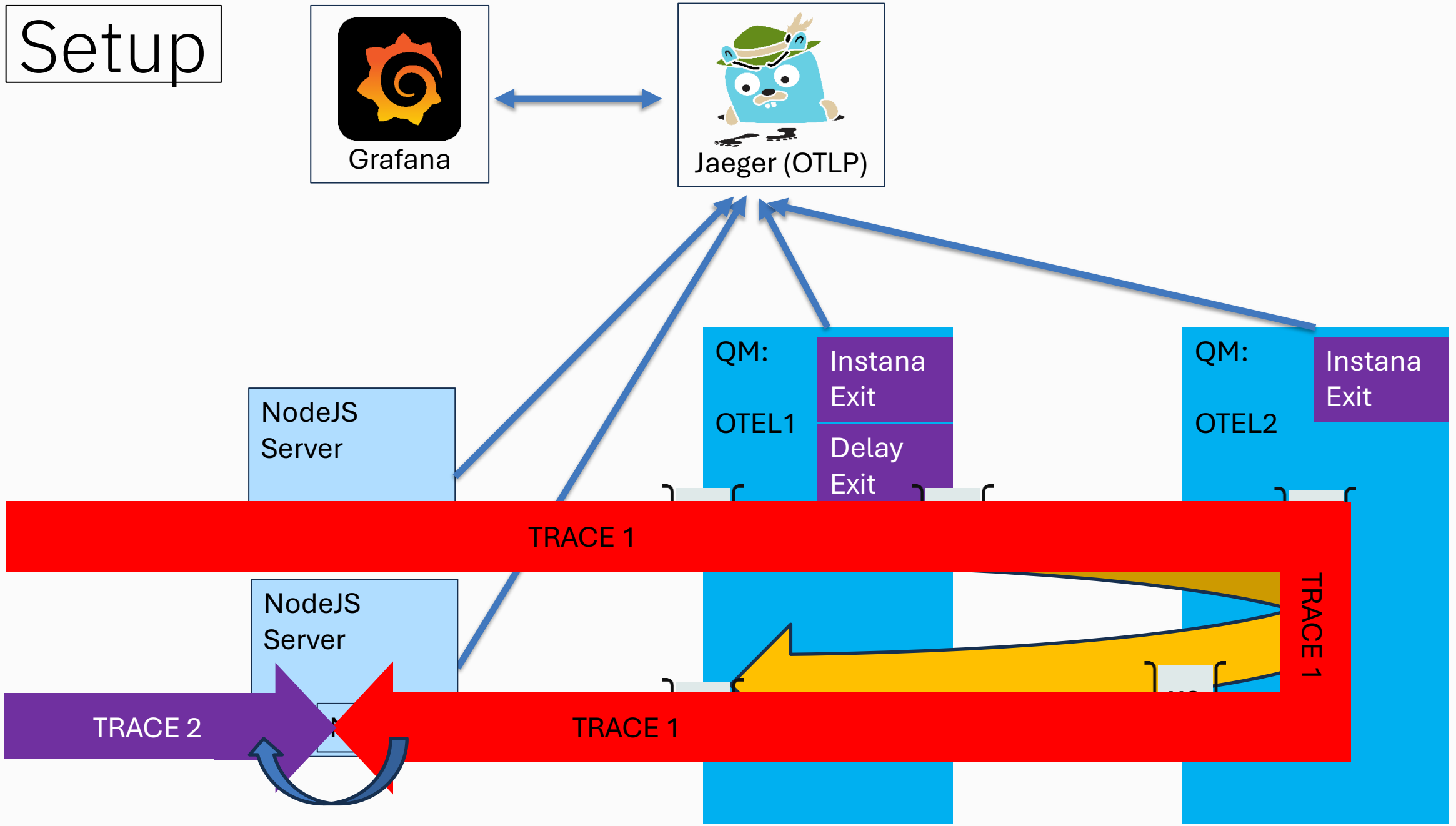
- Does not force OTel dependencies on non-instrumented apps

- The NodeJS version is completely transparent/automatic
- For Go, you currently have to instrument applications by hand
  - OTel are still developing automatic techniques
  - Some minor changes needed for MQ apps to set up

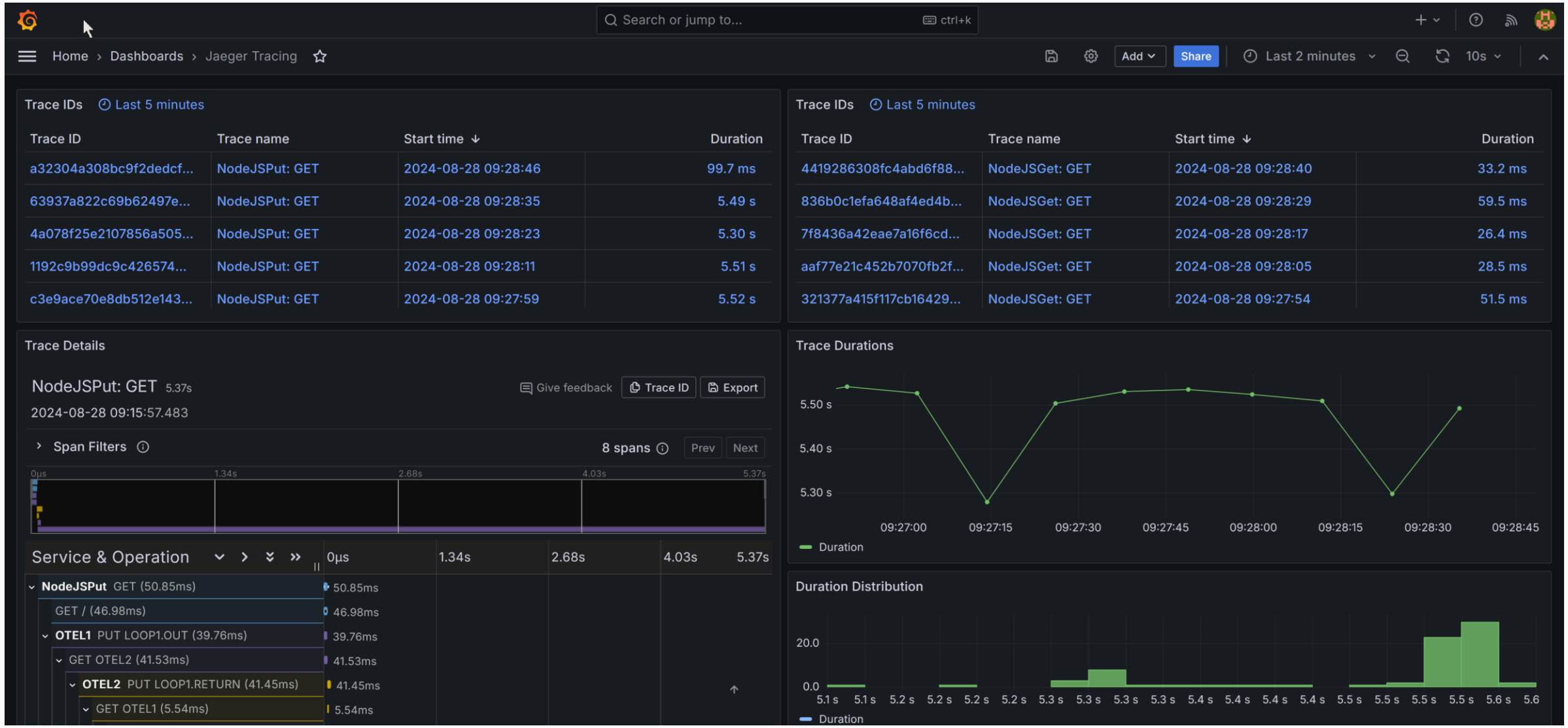
Setup


Grafana
Jaeger (OTLP)
QM:
OTEL1
Instana Exit
Delay Exit
QM:
OTEL2
Instana Exit
NodeJS Server
MQ PUT
NodeJS Server
MQ GET
curl GET
curl GET
QR
XQ
QR
QL
XQ

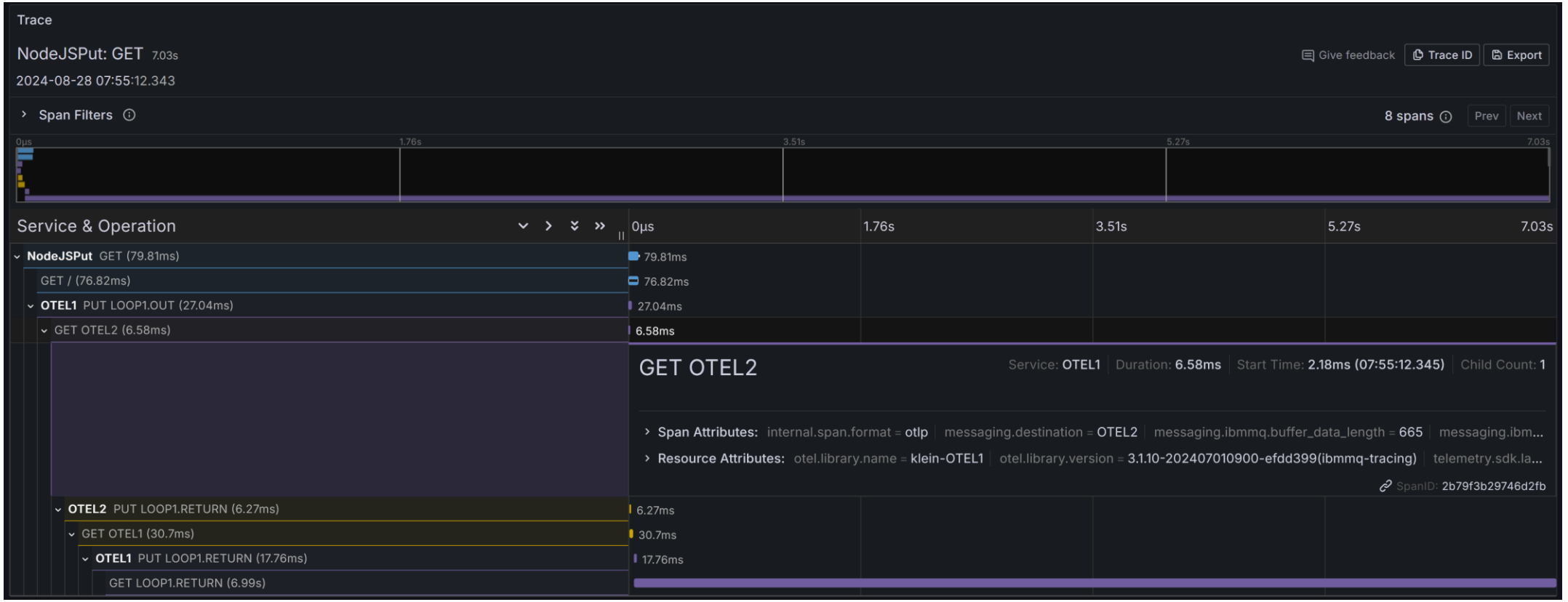
Setup


Grafana
Jaeger (OTLP)
QM:
OTEL1
Instana Exit
Delay Exit
QM:
OTEL2
Instana Exit
NodeJS Server
NodeJS Server
TRACE 1
TRACE 2
TRACE 1
TRACE 1

Grafana: Dashboard

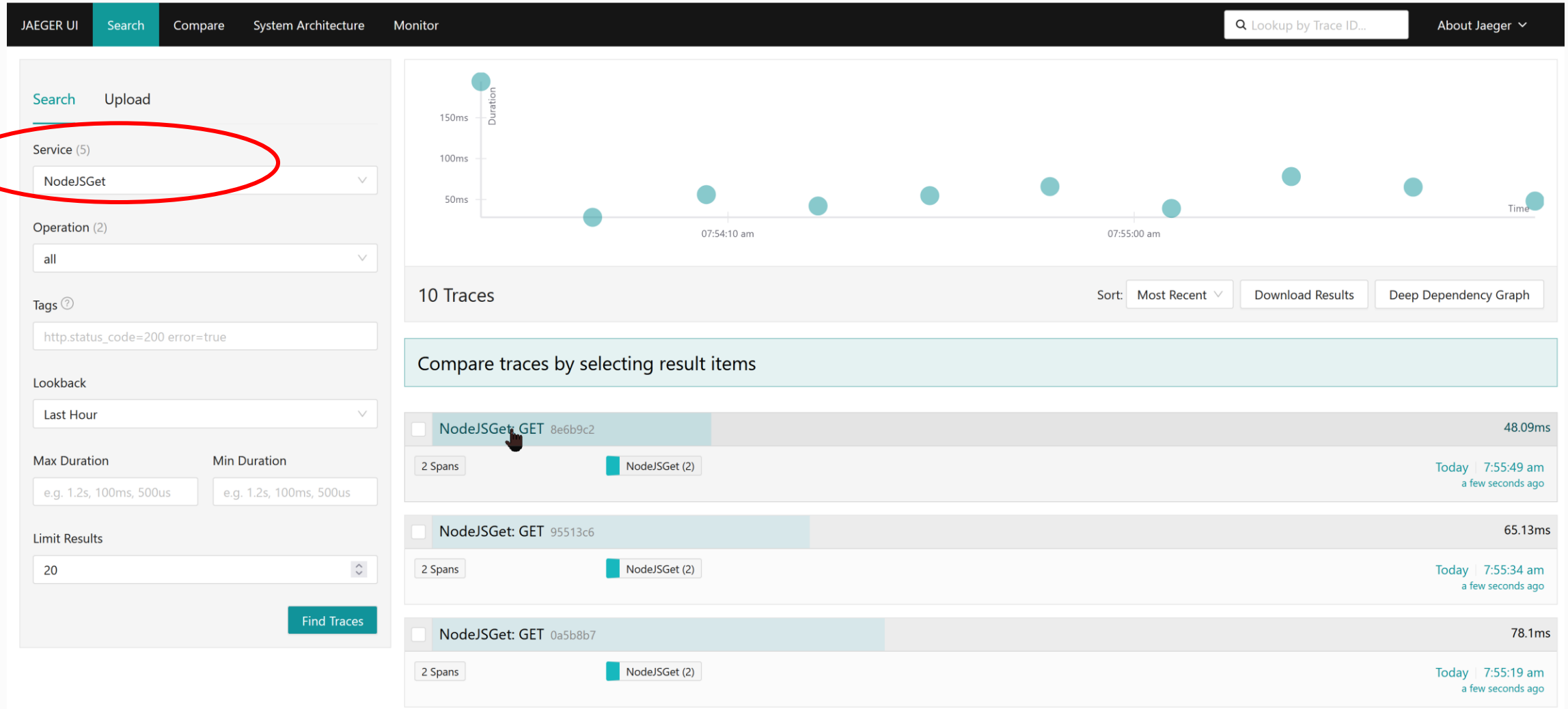


Grafana: PUT detailed trace

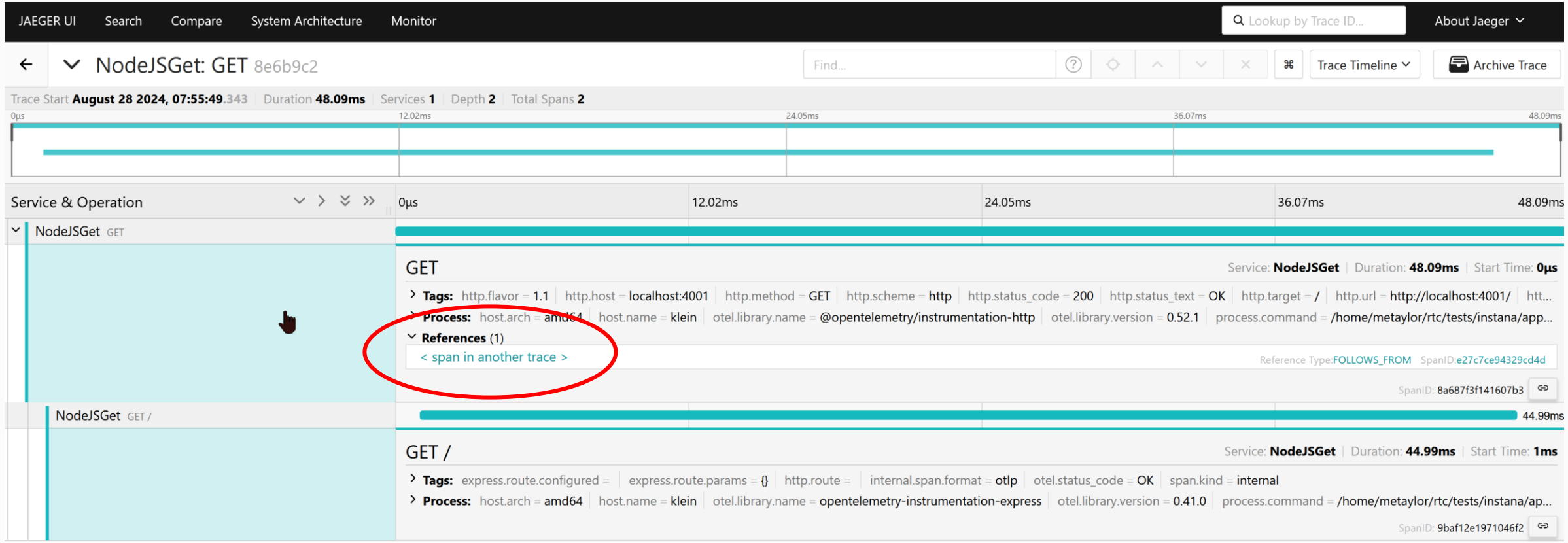


- Can see HTTP GET causing MQPUT through channels to msg finally removed
- “GET” used here for both MQGET and HTTP GET

Jaeger: Dashboard

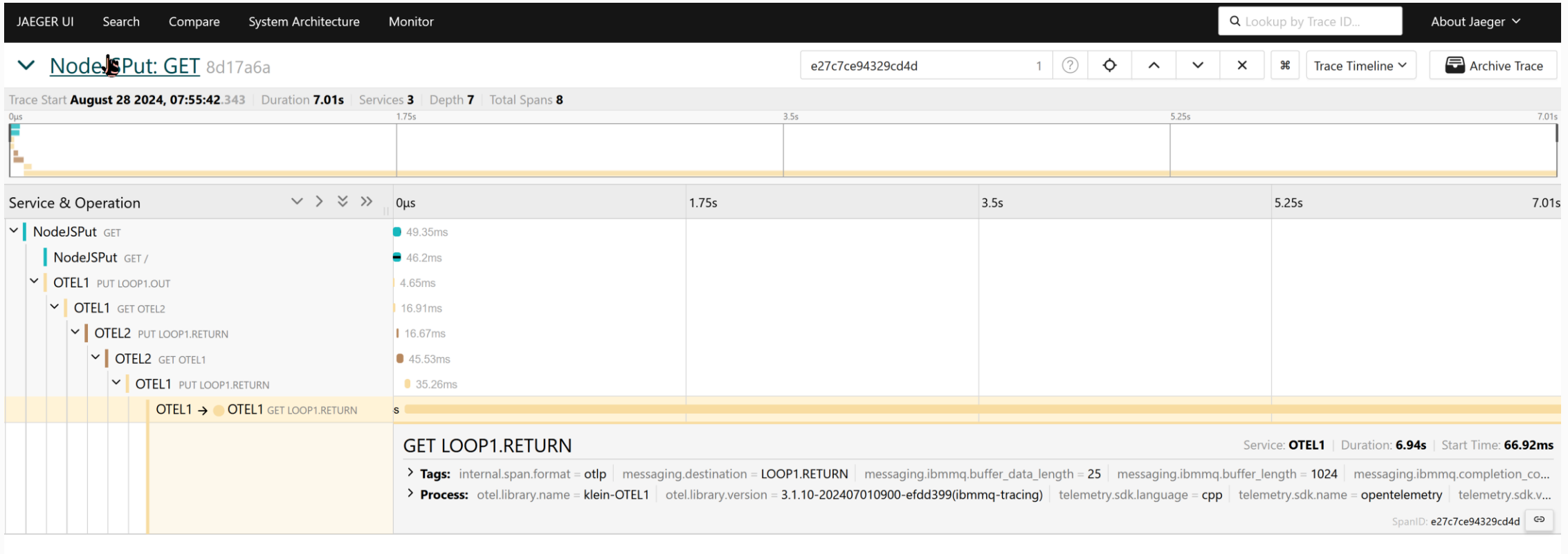


Jaeger: GET detailed trace



- Only see the HTTP GET here directly
- But a link to the MQPUT trace/span for the original operation

Jaeger: linked PUT detailed trace



Conclusions

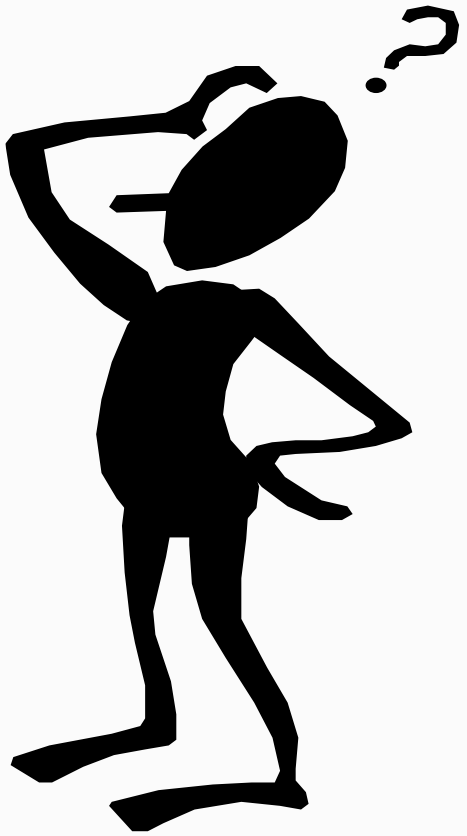
- Metrics and Logs are well-understood for MQ and OTel integration
- Tracing has made a good start
- Still work to do for more integrations
- Which applications/environments/languages are most important

Links

- GitHub source: https://github.com/ibm-messaging/mq-metric-samples
- OTel Registry: https://opentelemetry.io/ecosystem/registry/?s=mq

- Instana Tracing:
  - https://www.ibm.com/docs/en/instana-observability/current?topic=mq-tracing
- Articles:
  - https://marketaylor.synology.me/?p=1564 (Metrics)
  - https://marketaylor.synology.me/?p=1542 (Logs)
  - And search others on that site for older/generic information on the repo

- Aha Idea for formal support:
  - https://integration-development.ideas.ibm.com/ideas/MESNS-I-681



Any questions?