# Bringing Mainframe into the Future: OpenTelemetry for Db2 for z/OS

Christoph Theisen (ctheisen@rocketsoftware.com)

Credits to Joern Thyssen (jthyssen@rocketsoftware.com)

© Rocket Software, Inc. or its affiliates 1990 – 2025. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates.
© Copyright IBM Corporation 2023. IBM, the IBM logo, ibm.com, and Watson are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

# Learning Objectives

- Understand the basic concepts of OpenTelemetry and how this integrates with Db2 for z/OS
- Explain how Context Propagation works across z/OS subsystems
- Compare OpenTelemetry with traditional instrumentation approaches
- Recognize key considerations of Db2 for z/OS OpenTelemetry implementation



IBM | Rocket software

# Agenda

- **OpenTelemetry: Motivation and Basics**
- OpenTelemetry in IBM Z Context
- Db2 for z/OS Open Telemetry Implementation
- Conclusion and Lessons Learned

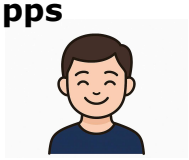


IBM | Rocket software

# Does that look familiar?

**Today's application landscape:**
**Mix of API-driven services**

**One big black box for users invoking services via mobile or web apps**



Mobile app

Web app

API Gateway

Database Service

Microservice 1

Microservice 2

Microservice 3

Cloud service A

Cloud service B

Mainframe service

MQ

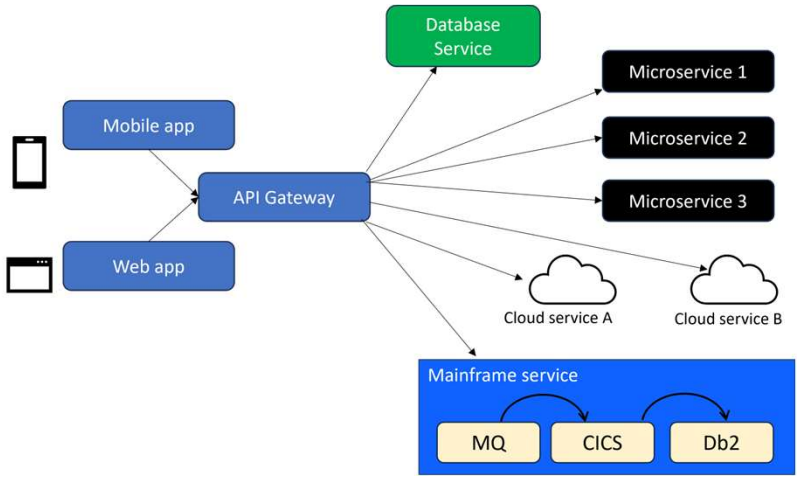
CICS

Db2

**Different platforms, operating systems, languages, architectural patterns, monitoring, tracing, ...**

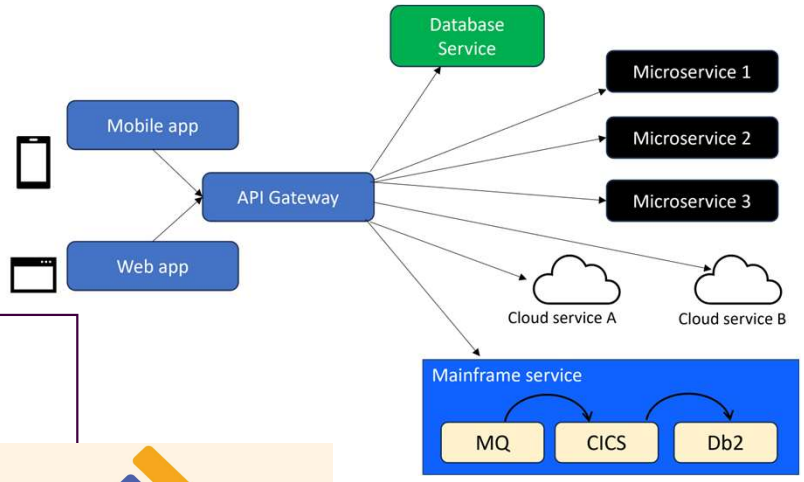
# Just Some Few Challenges...

- **Complexity – dozens of services can be involved in a single end user request**
- **Heterogenity of technology stack**
- **Different teams own and run different services**
- **Dynamic infrastructure – containers spin up and down**
- **No centralized visibility – every service can have its own concept of logging, tracing, monitoring, ...**
- **Hard to trace service requests**
- **Risk of blind spots and black boxes without any monitoring and tracing**



# Where OpenTelemetry enters the Game...

- **Complexity – dozens of services can be involved in a single end user request**
- **Heterogenity of technology stack**
- **Different teams own and run different services**
- **Dynamic infrastructure – containers spin up and down**
- **No centralized visibility – every service can have its own concept of logging, tracing, monitoring, ...**
- **Hard to trace service requests**
- **Risk of blind spots and black boxes without any monitoring and tracing**




OpenTelemetry

# What is OpenTelemetry?

- **Vendor-neutral framework for IT observability**
- Defines standards for collection and processing of **metrics**, **traces**, and **logs**
- Supports cloud, hybrid cloud, on-premise architectures
- Defines a protocol for exchange of telemetry data: **OTLP**
- Supported by Open Source community and (commercial) vendor ecosystem
- Defines APIs and provides SDKs to empower applications for OpenTelemetry



IBM | Rocket software

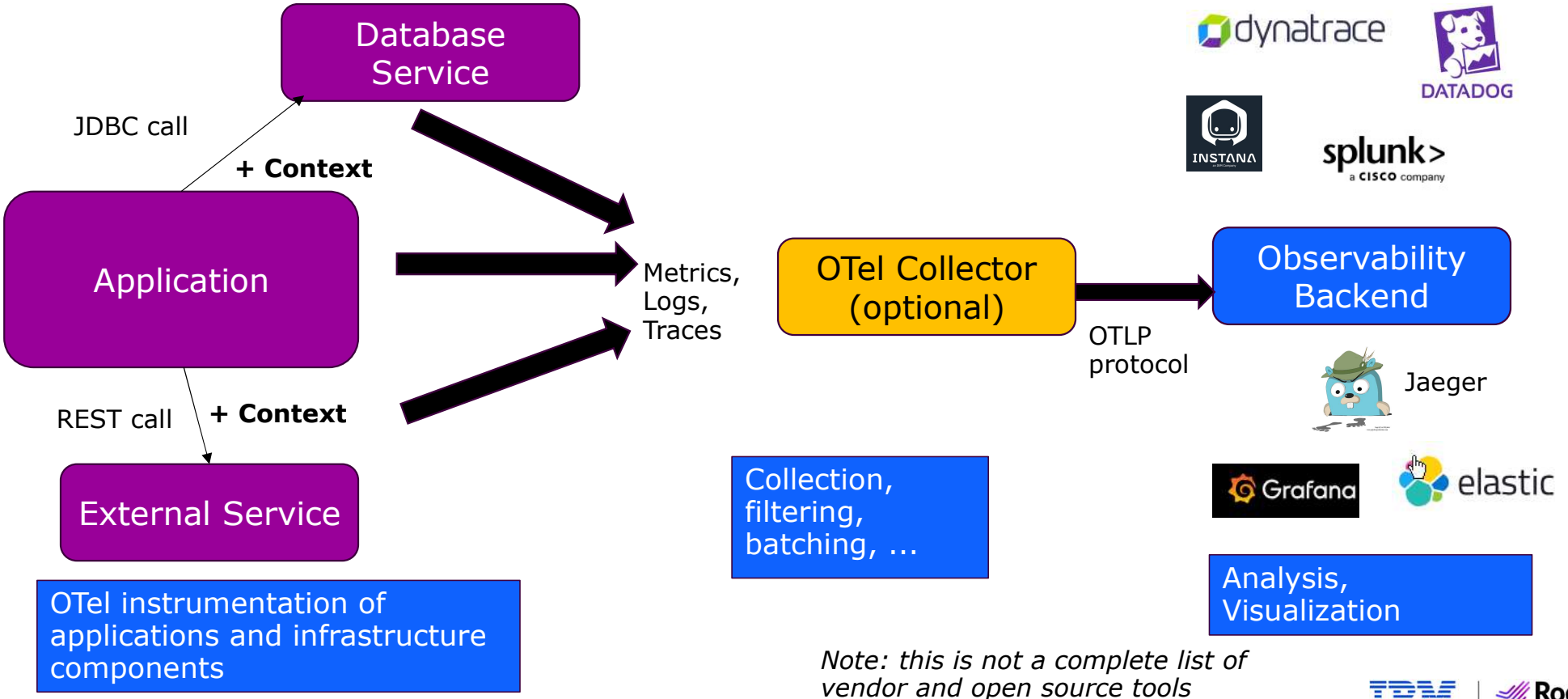
# Benefits of OpenTelemetry

- **Almost no coding required to enable existing applications for OpenTelemetry** (“Zero Code Instrumentation”)
- Moderate effort to add tailored instrumentation to an application
  - **SDK/API support for Java, .NET, Go, Python, PHP, Ruby,...**
- Choice of **open source and vendor tools** for collection, visualization and analysis
- OTel’s “Trace” concept allows **end-to-end tracing** and monitoring across different services and platforms
- Does not replace but **complements existing vendor solutions** for platform-specific monitoring and observability



IBM | Rocket software

# OpenTelemetry in Action




*Note: this is not a complete list of vendor and open source tools*

# OpenTelemetry – what can be observed?

**Metrics**

- Numerical data points collected over time
- Gathered from periodic snapshots
- OTel does NOT define which metrics are captured

**Logs**

- Text records, either structured or unstructed, which have a timestamp and optional metadata

**Traces**

- **Track the flow of an application** across distributed systems (or even inside an application)
- **Spans** generated by instrumented applications and components to support distributed tracing

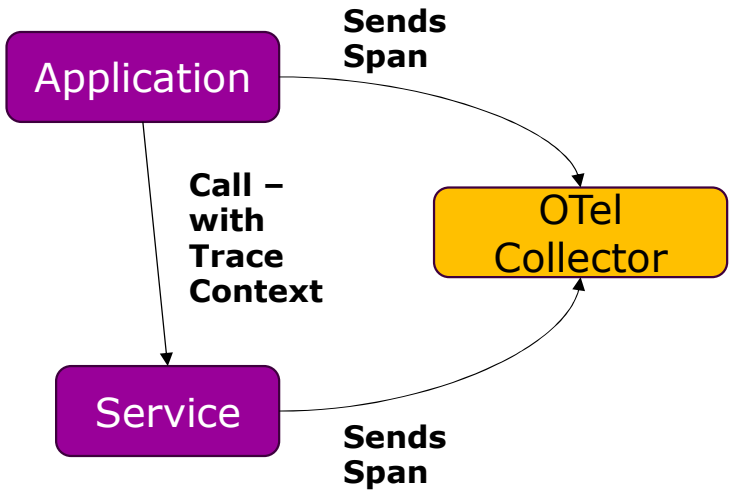
Focus in IBM Z context

**Baggage**

- Contextual information (key/value format) that can be passed across services (typically used in a trace context)

# Contexts, Traces, and Spans

- **Every OTel instrumented component sends its own data (metrics, logs, traces) to OTel collector**
- OTel collector correlates incoming records
- Every incoming record represents a piece of work – which is named „**Span**"
- Applications and infrastructure components (e.g database server) are responible for creating and terminating Spans
- Every service request has a unique **Trace ID**
- Every involved Span contains this **Trace ID**
- Trace context: information passed from an instrumented application to a service which contains **Trace ID and Parent Span ID**
- **Important: No Trace Context = No Correlation**



# Ingredients of a Span Record

## Sample Span – formatted by a visualization tool (Jaeger)

```
"traceID": "04db42ec0272e2862f508b53f577f448",
    "spanID": "e23fe8d4667ad30a",
    "operationName": "My Java Method",
    "references": [
        {
            "refType": "CHILD_OF",
            "traceID": "04db42ec0272e2862f508b53f577f448",
            "spanID": "24b8f5f1a981a9a5"
        }
    ],
    "startTime": 1782896589388537,
    "duration": 46750,
    "tags": [
        {
            "key": "otel.scope.name",
            "type": "string",
            "value": "io.opentelemetry.example"
        },
        {
            "key": "span.kind",
            "type": "string",
            "value": "internal"
        }
    ]
```

**Trace ID** – identifies a service call

**Span ID** – unique for a Span

Operation Name – choosen by instrumented application

Identification of **parent Span**

**Start time** of Span in Unix Epoch format (nano second precision)

**Duration** in microseconds (note: this is a derived value, OTel Span records contain endTime)

**Optional list of attributes in key / value format**

Set by instrumented application to deliver additional information

# Agenda

- OpenTelemetry: Motivation and Basics
- **OpenTelemetry in IBM Z Context**
- Db2 for z/OS Open Telemetry Implementation
- Conclusion and Lessons Learned

# OpenTelemetry – now available on IBM Z

- October 2025: Initial support in z/OS 3.2 and 3.1 (OA66345)
- SMF in-memory logstreams for new record types 1157 to 1161
- New Java-based “z/OS OTel Emitter” STC, part of z/OS Data Gatherer
- Very basic documentation (README file in USS)
- Reads OTel trace records – generated by Db2, IMS, CICS, MQ and written to SMF – and sends them via OTLP to an OTel Collector
- Each subsystem (such as Db2 or CICS) responsible for accurate generation of OTel trace records



IBM | Rocket software

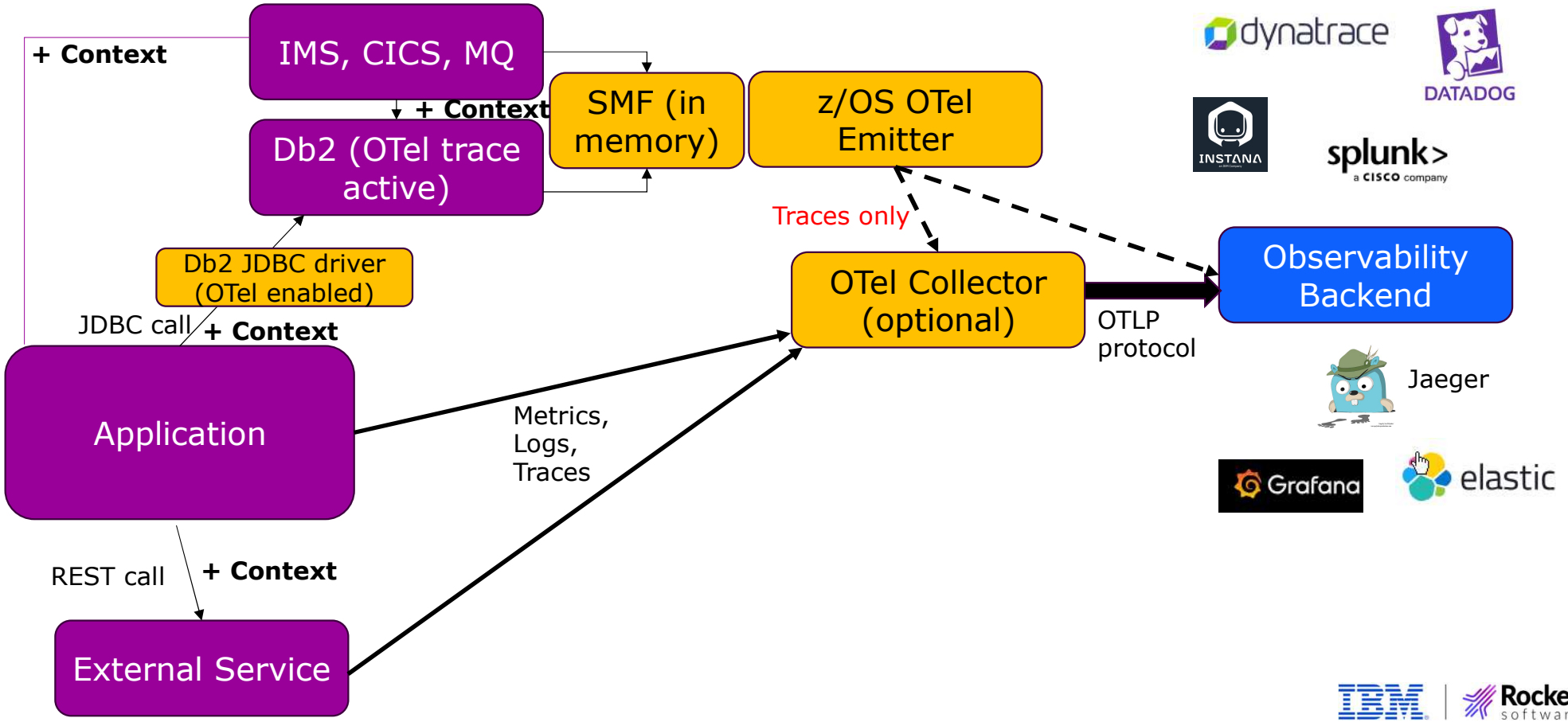
# Distributed Tracing across IBM Z Subsystems

- **IBM Z is no more "black box"** from distributed perspective
- CICS, CICS TG, IMS, MQ, Db2 can now generated their own OTel trace records
  - Requires PTFs or minimum version/release level for every type of subsystem
- Instrumented clients **must provide trace context** when calling z/OS applications and services
- z/OS Subsystems can **propagate trace context**
  - Example: MQ → CICS → Db2
- **End-to-end tracing** from distributed platforms to IBM Z and inside z/OS subsystems



IBM | Rocket software

# OpenTelemetry, IBM Z and Db2 for z/OS






IBM | Rocket software

# Agenda

- OpenTelemetry: Motivation and Basics
- OpenTelemetry in IBM Z Context
- **Db2 for z/OS Open Telemetry Implementation**
- Conclusion and Lessons Learned

# OpenTelemetry Support in Db2 for z/OS (1|2)

- **Db2 13 for z/OS requires PH67971 and PH68073 for OTel support**
  - Effective for all function levels
- New command “START OTEL EMIT(YES)” activates generation of OTel trace records to SMF
  - Does not start a Db2 trace like START TRACE does
- Records written when
  - OTel tracing is active **AND**
  - Applications provide Trace Context to Db2 **AND**
  - Unit of Work is committed

# OpenTelemetry Support in Db2 for z/OS (2|2)

- Accounting class 1 and 2 should be active (any destination)
  - Provides additional metrics in the Span record (see next slide)
- **Context propagation supported:**
  - JDBC T4 – requires Db2 Connect 12.1.3 or higher (JCC 4.36 or higher)
  - Db2 native REST
  - z/OS Connect REST
  - CICS-Db2 attach, CICS Liberty
  - IMS-Db2 attach, IMS-Db2 Java JDBC T2 environments
- See documentation for the individual products for minimum maintenance level for OTel support (context propagation)



IBM | Rocket software

# Example for a Db2 Span (1|3)


```
"traceID": "04db42ec0272e2862f508b53f577f448",
   "spanID": "e23fe8d4667ad30a",
   "operationName": "Db2 unit of work",
   "references": [
       {
           "refType": "CHILD_OF",
           "traceID": "04db42ec0272e2862f508b53f577f448",
           "spanID": "24b8f5f1a981a9a5"
       }
   ],
   "startTime": 1782896592360606,
   "duration": 10894409,
   "tags": [
       {
           "key": "db.ibm.db2.connection.id",
           "type": "string",
           "value": "SERVER"
       },
       {
           "key": "db.ibm.db2.connection.type",
           "type": "string",
           "value": "DIST"
       },
       {
           "key": "db.ibm.db2.correlation.id",
           "type": "string",
           "value": "SampleQueries_2"
       },
       {
           "key": "db.ibm.db2.cpu.gp",
           "type": "int64",
           "value": 518225
       },
```


**ID of Parent Span**

**Time spent in Db2**

**Db2-provided information (span attributes)**

**CPU time on GCP in microseconds**


```
{   "key": "db.ibm.db2.cpu.ziip",
    "type": "int64",
    "value": 588863
},
{
    "key": "db.ibm.db2.et",
    "type": "int64",
    "value": 6920390
},
{
    "key": "db.ibm.db2.group.name",
    "type": "string",
    "value": "IDS2"
},
{
    "key": "db.ibm.db2.location",
    "type": "string",
    "value": "RS01IDS2"
},
{
    "key": "db.ibm.db2.luwid",
    "type": "string",
    "value": "GA11E103.G409.E23FE8CFE0C7.0004"
},
{
    "key": "db.ibm.db2.member.name",
    "type": "string",
    "value": "I9A2"
},
{
    "key": "db.ibm.db2.planname",
    "type": "string",
    "value": "DISTSERV"
},...
```


**zIIP time in microseconds**

**Elapsed time in microseconds**

**Logical Unit of Work ID**

# Example for a Db2 Span (2|3)

```
"traceID": "04db42ec0272e2862f508b53f577f448",
   "spanID": "e23fe8d4667ad30a",
   "operationName": "Db2 unit of work",
   "references": [
       {
           "refType": "CHILD_OF",
           "traceID": "04db42ec0272e2862f508b53f577f448",
           "spanID": "24b8f5f1a981a9a5"
       }
   ],
   "startTime": 1782896592360606,
   "duration": 10894409,
   "tags": [
      ...
```

**ID of Parent Span**

**Time spent in Db2**

- Mandatory information to allow correlation with other Spans
- Trace Id and Parent Span Id delivered by calling application in Trace Context
- Db2 generates its own Span Id, operation name and start / end time
- Duration (in this example) calculated by visualization tool

# Example for a Db2 Span (3|3)

- Attributes („Tags“) added by Db2 as additional information (not enforced by OTel)
- Important: Elapsed time reported here can differ from „duration“ (depends on attach type, thread allocation, number of fetches, ...)

```
"tags": [
   {
        "key": "db.ibm.db2.connection.id",
        "type": "string",
        "value": "SERVER"
   },
   {
        "key": "db.ibm.db2.connection.type",
        "type": "string",
        "value": "DIST"
   },
   {
        "key": "db.ibm.db2.correlation.id",
        "type": "string",
        "value": "SampleQueries_2"
   },
   {
        "key": "db.ibm.db2.cpu.gp",
        "type": "int64",
        "value": 518225
   },
```

**CPU time on GCP in microseconds**

```
{    "key": "db.ibm.db2.cpu.ziip",
     "type": "int64",
     "value": 588863
},
{
     "key": "db.ibm.db2.et",
     "type": "int64",
     "value": 6920390
},
{
     "key": "db.ibm.db2.group.name",
     "type": "string",
     "value": "IDS2"
},
{
     "key": "db.ibm.db2.location",
     "type": "string",
     "value": "RS01IDS2"
},
{
     "key": "db.ibm.db2.luwid",
     "type": "string",
     "value": "GA11E103.G409.E23FE8CFE0C7.0004"
},
{
     "key": "db.ibm.db2.member.name",
     "type": "string",
     "value": "I9A2"
},
{
     "key": "db.ibm.db2.planname",
     "type": "string",
     "value": "DISTSERV"
},...
```

**zIIP time in microseconds**

**Elapsed time in microseconds**

**Logical Unit of Work ID**

# Enabling OTel on the Application Side

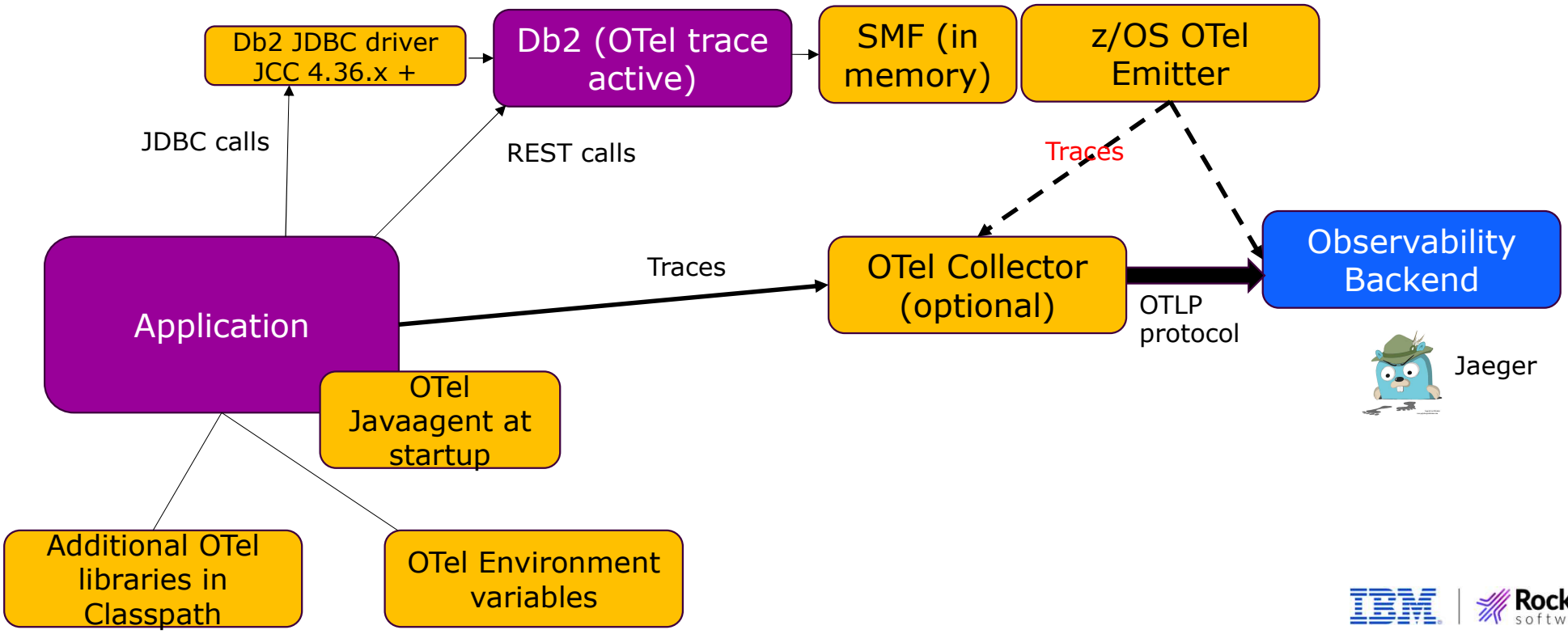
- **Applications must be enabled for:**
  - Creating Traces and Spans
  - Emitting Spans to OTel Collectors
  - Providing Trace Context to called applications and services
- **Zero Code Instrumentation**
  - Enabling applications without code change
  - Requires additional libraries plus configuration changes
  - Support for Java, JavaScript, Python, PHP, .NET, Go (other languages with limitations)
  - Automatic generation of Spans for service calls



IBM | Rocket software

# Zero-code Instrumentation – Java Example

**Consider an existing Java application with JDBC calls to Db2 for z/OS and native Db2 REST services**



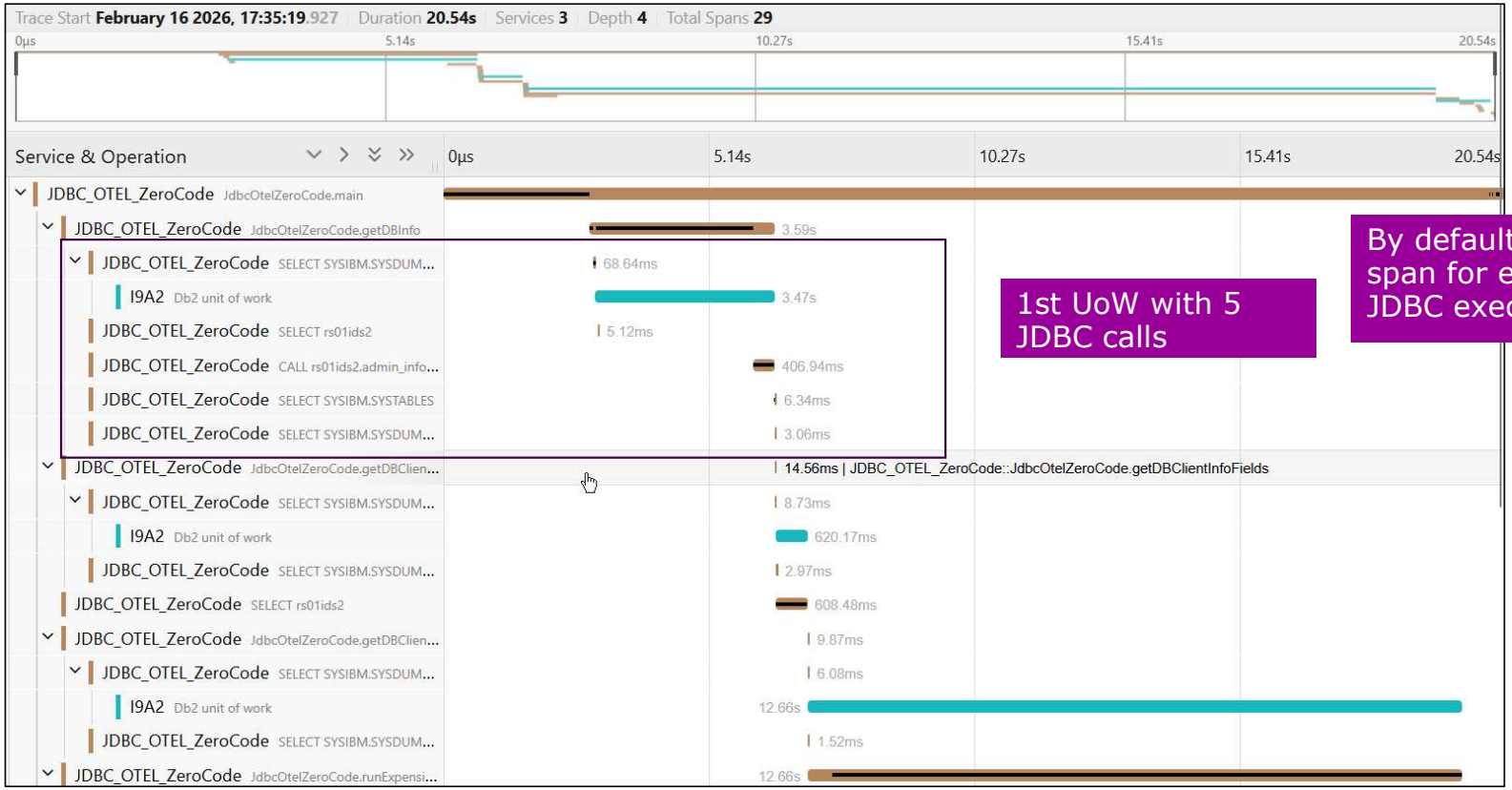
# Zero Code Instrumentation – Java Example

- Minimum set of OTel libraries required in Classpath
  - Downloadable from GitHub
- OTel Javaagent must be specified in JAVA_TOOL_OPTIONS
- Additional environment variables required:
  - OTEL_TRACES_EXPORTER: which protocol to use for emitting trace records (e.g. OTLP)
  - OTEL_EXPORTER_OTLP_ENDPOINT: hostname/port of collector or observability backend
  - OTEL_SERVICE_NAME: meaningful name to locate traces in observability backend
  - OTEL_INSTRUMENTATION_METHODS_INCLUDE: optional – list of Java methods in application for which OTel instrumentation should be enabled



IBM | Rocket software

# Trace Example with Zero Code Instrumentation




Every horizontal bar is a span

1st UoW with 5 JDBC calls

By default: one span for every JDBC execution

# Manual (Code-based) Instrumentation

- Requires OTel SDK for instrumented application (GitHub)
- Full control of Span creation and hierarchy
- Can provide application specific attributes to Span
- Allows alignment of Spans to Db2 Unit of Work
- MUST provide Trace Context before calling Db2 (or CICS, MQ, IMS...)

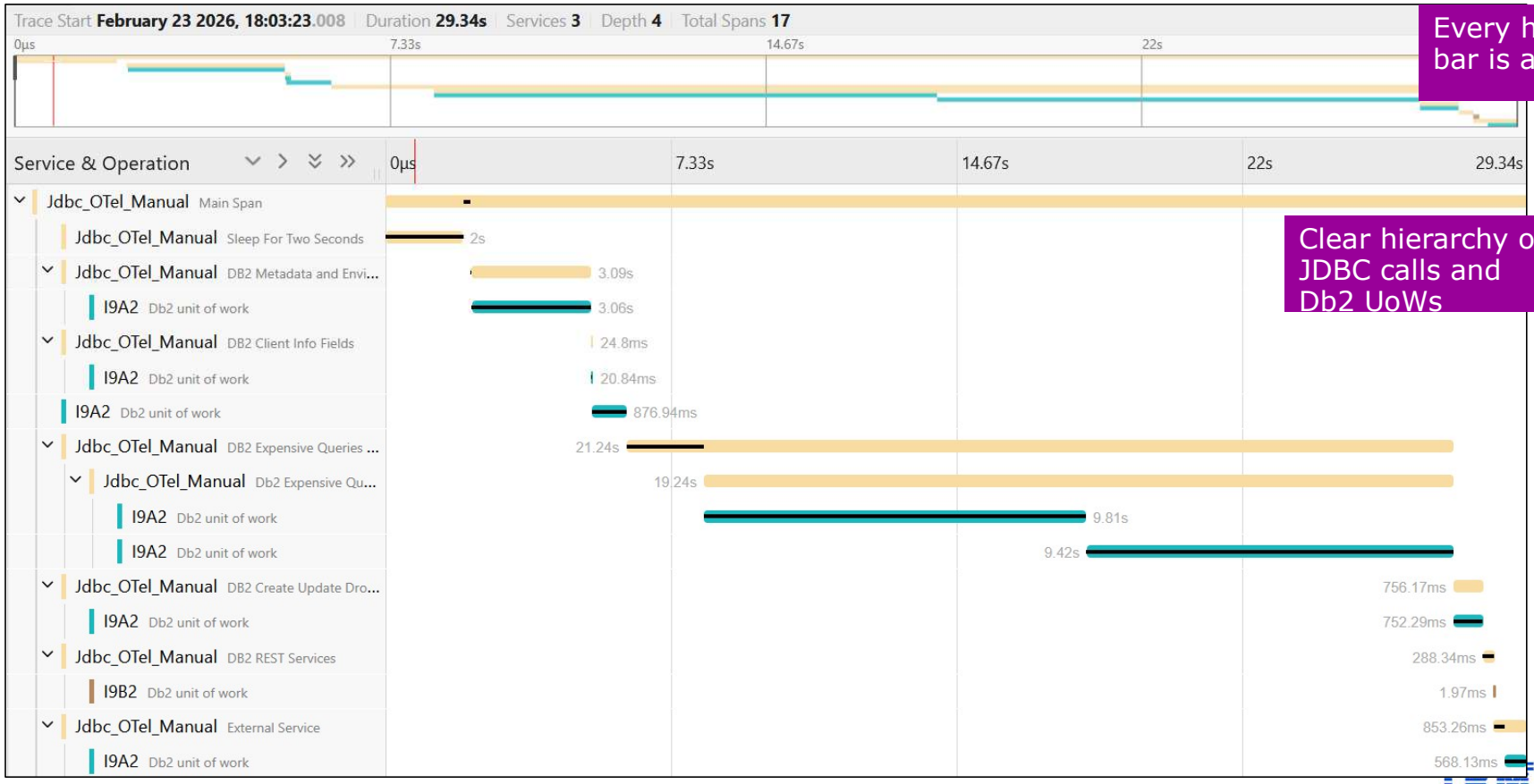
```
Span getDBClientInfoSpan = tracer.spanBuilder("DB2 Client Info
Fields")
                    .setSpanKind(SpanKind.INTERNAL)
                    .setParent(parentContext)
                    .startSpan();
            try (Scope dbScope =
getDBClientInfoSpan.makeCurrent()) {
                getDBClientInfoFields(con);
                con.commit();
            } catch (Exception e) {
            } finally {
                getDBClientInfoSpan.end();
            }
```

*Java example:*

*Method getDBClientInfoFields runs some Db2 queries*

*Dedicated span created for this method*

# Trace Example with Manual Instrumentation




Every horizontal bar is a span

Clear hierarchy of JDBC calls and Db2 UoWs

# Unbox the Black Box: CICS/Db2 Example

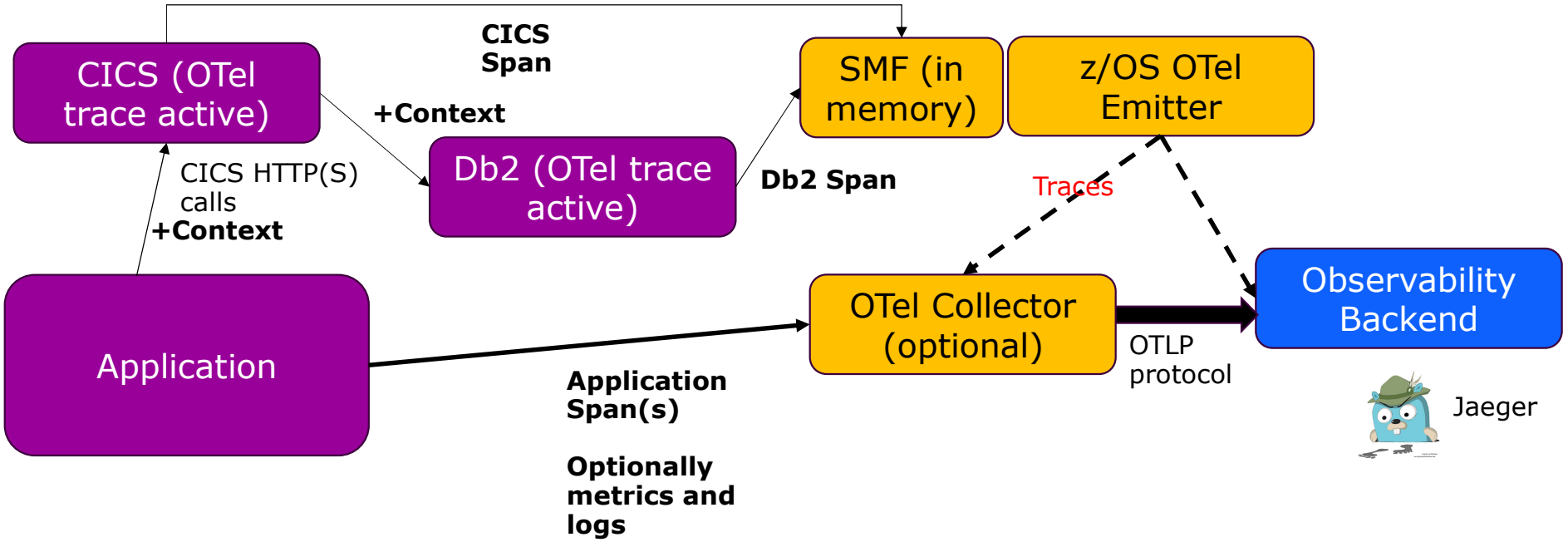
- **CICS TS 6.3: Support for OTel tracing + context propagation**
  - Allows tracing of service flows inside the mainframe
- Multiple entry points into CICS supported: CICS HTTP, CICS TG, z/OS Connect, ...
- Context propagation to Db2 via CICS-Db2 attach, CICS-Liberty attach (JDBC T2), ...
- More granular control options than Db2
  - Context propagation yes/no (towards other subsystems)
  - Emit yes/no (generating CICS Spans for z/OS OTel emitter
  - Can be set on region or transaction level



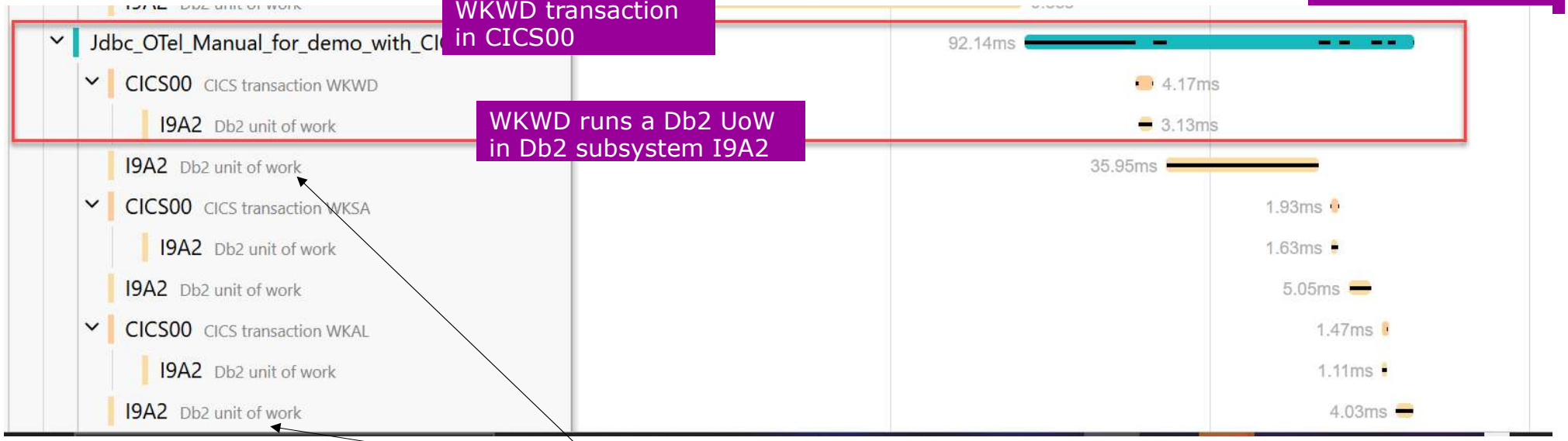
IBM | Rocket software

# CICS/Db2 Example

Consider a client application calling CICS via HTTP interface



# Result in Jaeger UI



# Agenda

- OpenTelemetry: Motivation and Basics
- OpenTelemetry in IBM Z Context
- Db2 for z/OS Open Telemetry Implementation
- **Conclusion and Lessons Learned**

# Traditional Db2 Monitoring vs. OpenTelemetry

## Db2 IFI Instrumentation

- Focused on **metrics**
- Proprietary format
- In-Db2 events only (e.g. Accounting record)
- Can trace SQL flow and related events (getpages, lock requests, I/Os,etc.) with overhead
- No distributed point of view
- Complexity of Db2 IFI handled by vendor products

## Db2 OTel Instrumentation

- OTel **traces (with few metrics embedded)**
- OTel trace format
- Contributes to distributed point of view
- Can provide higher level of granularity for basic metrics than accounting rollup records
- Leveraged by Open source tools or vendor products



IBM | Rocket software

# Lessons Learned (1|3)

- **OTel is no replacement for traditional Db2 monitoring (yet)**
  - Hundreds of Db2 metrics available from IFI calls + plus events (such as accounting records, deadlocks, timeouts, authorization failures, etc.
  - Db2's OTel implementation focusing on Traces and Spans
  - Metrics (in Span attributes) are complimentary – not necessarily as accurate as accounting records

- **Scope of Db2 Spans is Unit of Work**
  - Can provide more details than accounting record in case of accounting rollup
  - Application's COMMIT strategy drives Span generation
  - No end-to-end tracing of single SQL executions (unless UoW has one SQL only)



IBM | Rocket software

# Lessons Learned (2|3)

- **Currently only little control on Db2 side**
  - Once OTel tracing is on → every OTel instrumented application can cause Db2 Span generation
  - Better control on CICS side
- **Time synchronization of all instrumented application is crucial**
- **Amount of trace records can be challenging for high-volume applications**
  - Consider OTel sampling
- **Consider manual instrumentation or instrumentation via Annotations if Zero-Code is too inflexible**



IBM | Rocket software

# Lessons Learned (3|3)

- **Visualization with Zero-Code instrumentation can be misleading for JDBC calls to Db2**
  - One Span for every single JDBC call on the client side, but one Span for entire UoW on Db2 side
- **Not all connection types into Db2 currently supported**
  - For example, ODBC/CLI currently (mid 2026) missing
- **According to IBM, only very little overhead for Span generation**
- **OTel is a very dynamic topic, many activities currently going on (Open Source, vendors, also IBM Z ecosystem)**



IBM | Rocket software

# Resources

Db2 related blogs:

https://community.ibm.com/community/user/blogs/m-sueli-almeida/2025/09/29/opentelemetry-support-in-db2-for-zos

https://community.ibm.com/community/user/blogs/jrn-thyssen1/2025/12/16/practical-guide-to-opentelemetry-tracing-for-java1

IBM AIOps related podcase:

https://www.buzzsprout.com/2370536/episodes/17544480

OpenTelemetry documentation:

https://opentelemetry.io/



IBM | Rocket software

# Thank you.

rocketsoftware.com
zconcierge@rocketsoftware.com

*© Rocket Software, Inc. or its affiliates 1990 – 2025. All rights reserved. Rocket and the Rocket Software logos are registered trademarks of Rocket Software, Inc. Other product and service names might be trademarks of Rocket Software or its affiliates.*
© Copyright IBM Corporation 2023. IBM, the IBM logo, ibm.com, and Watson are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.