Event Based Distributed Tracing – Observability Ecosystem Design

Sathiyanarayanan Palani[1], Sanjay Tiwari[2]

1: Intel SRR4, Intel Technology India, Bellandur, Bengaluru, Karnataka
2: Intel AMR, US

Abstract: In the field of Semiconductor Product Development & Testing, Telemetry methodology plays a significant role in collecting data during test execution on hundreds of platforms, for holistic quantization of design level improvements. So, data collection tools & methodology must support not only three pillars of observability (Logs, Traces & Metrics), but also more custom data types like Test Sessions & File IO Objects.

The data models involved in the Framework, is designed to get more insights from lesser data to ensure low latency data transfer rates. while workflow collector built based on the principles of Open Telemetry, focuses to solve problems involved in Electronic / Semiconductor testing tasks, rather than focusing on Microservice stacks.

The components / Building Blocks involved in the data collection pipeline, has minimal dependencies on external tools, hence business logics can be altered as per need. It creates an advantage over OTLP, where certain components like fluentd are rigid executables that stick to standards.

Keywords: Event Based Distributed Tracing (EBDT), Workflow Collector, Open Telemetry (OTLP)

1. Data Model Design

The framework is implemented with support for logs, metrics & traces data as the primitive data models, which contains specific data describing the nature of execution. In addition to that, two more custom data models have been introduced for efficient workflow collection, termed as session & object.

Rules: Unique names of all these data models should not have (.) or (/) slash in it.

1.1 Session:

- Every session should have unique ID and an optional configuration dictionary that defines the session.
- It can access several unique metrics, trace, log & pin attributes.

1.2 Object:

- Pins an Object (file / folder) with parent class info & timestamp (UTC). Folder will be stored as zip, storage size of both limited to 100 MB per object.

1.3 Log:

- Logs a statement under one of the five categories (DEBUG, INFO, WARNING, ERROR, CRITICAL) with parent class info and timestamp (UTC).

1.4 Metric:

- A meter child class can be defined by the user with the list of functions to be invoked, recurring at a periodic interval in a separate process.

1.5 Trace:

- Every Trace should have a unique trace id, and trace can be used as a sub attribute of session or another parent trace.

2. Building Blocks

2.1 Collector

➢ EBDT-Client

The Collector Library is used for defining the workflow of the test script, that streams data model packets to Service API at runtime.

➢ EBDT-Service

Service API accepts Data Model Packets to store them. Additionally, Packets will be streamed, only if D2C mode of the platform is Enabled before the start of the test session.

2.2 Dashboard

➢ EBDT-File Server

File Server API listens for files as bytes, to store it on the server & serve it back to users during On-click download events initiated from Dashboard.

➢ EBDT-Dash

Web UI developed with Dash Framework, where one can search the test sessions based on platforms of interest & Execution Date, with drilldowns to each of the unique session pages.

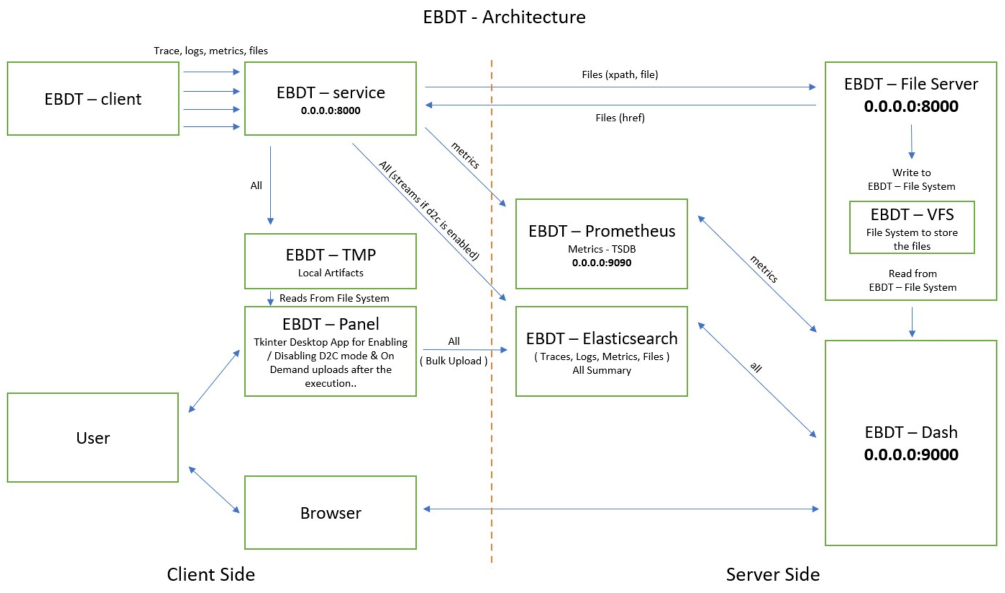

EBDT - Architecture
Trace, logs, metrics, files
EBDT – client
EBDT – service 0.0.0.0:8000
Files (xpath, file)
Files (href)
EBDT – File Server 0.0.0.0:8000
Write to EBDT – File System
EBDT – VFS File System to store the files
Read from EBDT – File System
metrics
All (streams if d2c is enabled)
All
EBDT – TMP Local Artifacts
Reads From File System
EBDT – Prometheus Metrics - TSDB 0.0.0.0:9090
metrics
EBDT – Panel Tkinter Desktop App for Enabling / Disabling D2C mode & On Demand uploads after the execution..
All (Bulk Upload)
EBDT – Elasticsearch (Traces, Logs, Metrics, Files) All Summary
all
User
EBDT – Dash 0.0.0.0:9000
Browser
Client Side
Server Side

Fig. 1. Architecture of the Observability Ecosystem

2.3 Panel

- EBDT-Panel

Tkinter based Desktop app facilitates users to control the functions of Service by manipulating D2C mode of the platform. Additionally, Same app can be used to upload new test run artifacts that is executed with disabled D2C mode as well as to open the session page of uploaded artifacts.

3. EBDT-Client Design

3.1 Traces

Traces records the start & end time of code snippet execution within a trace context, along with the meta data of the trace.

- Usage

```
import time
from ebdt.traces import tracer
with tracer('tracer-example') as trace:
    time.sleep(60)
```

- Schema

type : _trace
meta.pf : platform name
meta.run : trace name
meta.sn_st : trace start time
meta.sn_et : trace end time

3.2 Logs

Each Logging statements attributed to one among the five categories will record the data locally as well as in the cloud during the run based on D2C Mode.

- Usage

```
from ebdt.logs import logger
With logger('logger-example') as log:
    log.debug('Hello World')
    log.info('Hello World')
    log.warning('Hello World')
    log.error('Hello World')
    log.critical('Hello World')
```

- Schema

```
type : _log
log.lvl :
    one of (
    DEBUG /
    INFO /
    WARNING /
    ERROR /
    CRITICAL)
log.msg : user statement
meta.run : parent session / trace name
ts : timestamp (UTC)
```

3.3 Metrics

When a Meter is started, class methods are passed in as channels with an interval specified in seconds for recurring channel calls in an individual process.

- Usage

```
import time
from datetime import datetime, timezone
from ebdt.metrics import meter

class Time(meter) :

    def M_hour(self) :
        return datetime.now(timezone.utc).hour

    def M_minute(self) :
        return datetime.now(timezone.utc).minute

    def M_second(self) :
        return datetime.now(timezone.utc).second

with Time('meter-example') as metric:
      metric.open(
            channels=[
            'M_hour',
            'M_minute',
            'M_second'], interval=10)
     time.sleep(60)
```

- Schema

type : _metric

meta.pf : platform name

meta.run : metric name

meta.sn_st : meter start time

meta.sn_et : meter end time

3.4 Objects

Objects represent file System IO objects like uploaded file / folder in the form of meta data, that contains path of the file to be uploaded in the Virtual File System, managed by EBDT-file server, within listener service.

- Usage

```
import time
from ebdt.objs import uploader
with uploader('uploader-example') as obj :
      obj.file(
            desc='example readme file uploaded',
            path= 'examples/README.md')
      obj.folder(
            desc='example folder uploaded as zip',
            path='examples/chunks_test/
```

- Schema

type : _obj

meta.run : parent session / trace name

obj.desc : description of the attachment

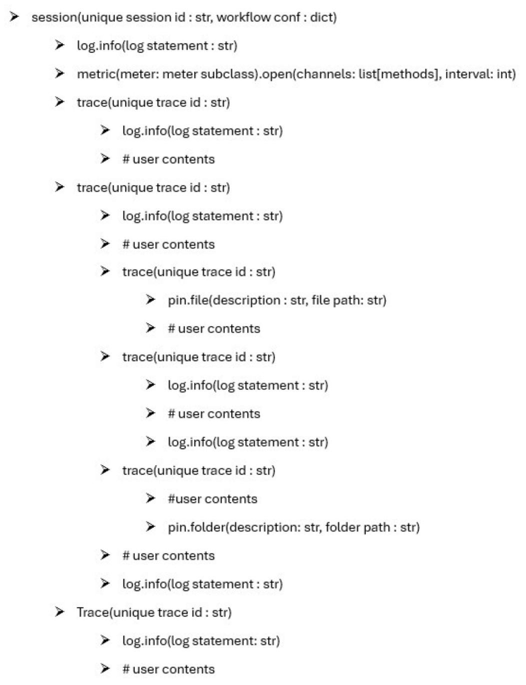
obj.path : Attachment file path in VFS

ts : timestamp (UTC)

3.5 Session

Session is the root trace that also records the pass / fail status of the run, with uncaught errors if any, as well as the platform info where the exécution is triggered and the unique session name with start & end time of the session.

- Psuedo Code



```
➢ session(unique session id : str, workflow conf : dict)
    ➢ log.info(log statement : str)
    ➢ metric(meter: meter subclass).open(channels: list[methods], interval: int)
    ➢ trace(unique trace id : str)
        ➢ log.info(log statement : str)
        ➢ # user contents
    ➢ trace(unique trace id : str)
        ➢ log.info(log statement : str)
        ➢ # user contents
        ➢ trace(unique trace id : str)
            ➢ pin.file(description : str, file path: str)
            ➢ # user contents
        ➢ trace(unique trace id : str)
            ➢ log.info(log statement : str)
            ➢ # user contents
            ➢ log.info(log statement : str)
        ➢ trace(unique trace id : str)
            ➢ #user contents
            ➢ pin.folder(description: str, folder path : str)
        ➢ # user contents
        ➢ log.info(log statement : str)
    ➢ Trace(unique trace id : str)
        ➢ log.info(log statement: str)
        ➢ # user contents
```

- Schema

type : _session

conf : info as dict from user at the start of the session

meta.run : unique name of the session

meta.pf : platform where the session is executed

meta.sn_st : start_time of the session

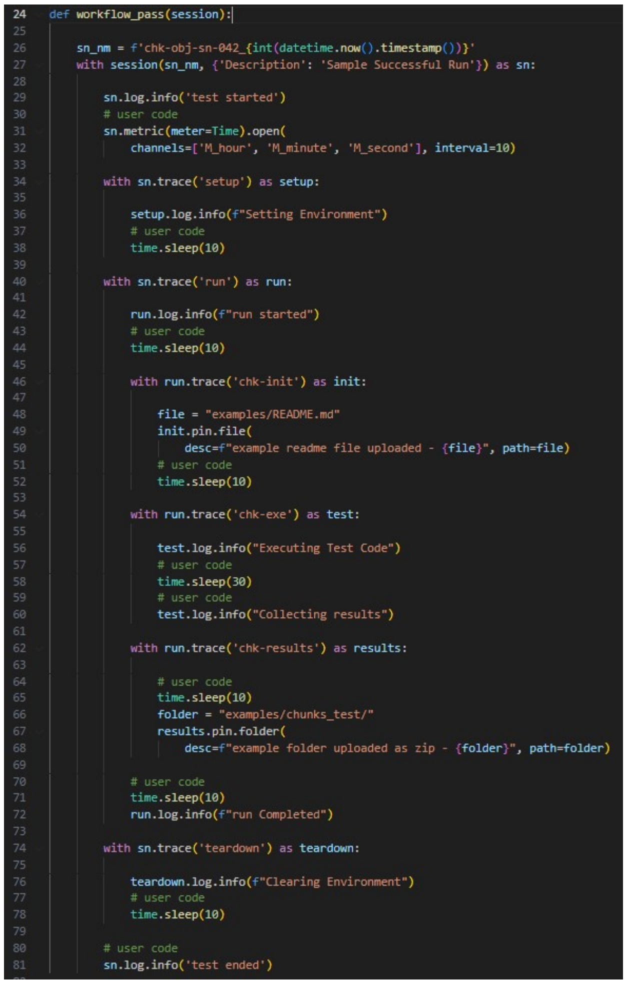
meta.sn_et : end time of the session

meta.sts : status of the run (success / failure)

meta.err : populated with uncaught errors, if any

4. Examples

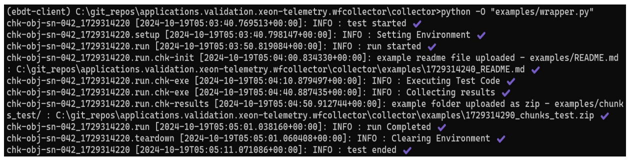
➢ Passing Script



```python
def workflow_pass(session):

    sn_nm = f'chk-obj-sn-042_{int(datetime.now().timestamp())}'
    with session(sn_nm, {'Description': 'Sample Successful Run'}) as sn:

        sn.log.info('test started')
        # user code
        sn.metric(meter=Time).open(
            channels=['M_hour', 'M_minute', 'M_second'], interval=10)

        with sn.trace('setup') as setup:

            setup.log.info(f"Setting Environment")
            # user code
            time.sleep(10)

        with sn.trace('run') as run:

            run.log.info(f"run started")
            # user code
            time.sleep(10)

            with run.trace('chk-init') as init:

                file = "examples/README.md"
                init.pin.file(
                    desc=f"example readme file uploaded - {file}", path=file)
                # user code
                time.sleep(10)

            with run.trace('chk-exe') as test:

                test.log.info("Executing Test Code")
                # user code
                time.sleep(30)
                # user code
                test.log.info("Collecting results")

            with run.trace('chk-results') as results:

                # user code
                time.sleep(10)
                folder = "examples/chunks_test/"
                results.pin.folder(
                    desc=f"example folder uploaded as zip - {folder}", path=folder)

            # user code
            time.sleep(10)
            run.log.info(f"run Completed")

        with sn.trace('teardown') as teardown:

            teardown.log.info(f"Clearing Environment")
            # user code
            time.sleep(10)

        # user code
        sn.log.info('test ended')
```

Fig. 2. Passing Script Example

➢ Passing Output



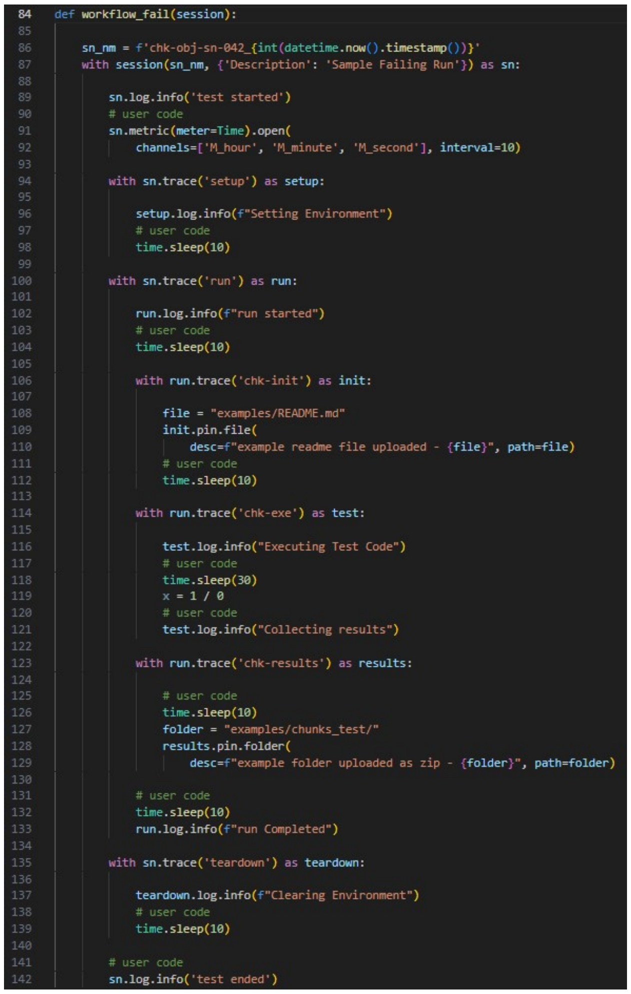
```
(ebdt-client) C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector>python -O "examples/wrapper.py"
chk-obj-sn-042_1729314220 [2024-10-19T05:03:40.769513+00:00]: INFO : test started ✓
chk-obj-sn-042_1729314220.setup [2024-10-19T05:03:40.798147+00:00]: INFO : Setting Environment ✓
chk-obj-sn-042_1729314220.run [2024-10-19T05:03:50.819084+00:00]: INFO : run started ✓
chk-obj-sn-042_1729314220.run.chk-init [2024-10-19T05:04:00.834330+00:00]: example readme file uploaded - examples/README.md
: C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\1729314240_README.md ✓
chk-obj-sn-042_1729314220.run.chk-exe [2024-10-19T05:04:10.879497+00:00]: INFO : Executing Test Code ✓
chk-obj-sn-042_1729314220.run.chk-exe [2024-10-19T05:04:40.887435+00:00]: INFO : Collecting results ✓
chk-obj-sn-042_1729314220.run.chk-results [2024-10-19T05:04:50.912744+00:00]: example folder uploaded as zip - examples/chunk
s_test/ : C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\1729314290_chunks_test.zip ✓
chk-obj-sn-042_1729314220.run [2024-10-19T05:05:01.038160+00:00]: INFO : run Completed ✓
chk-obj-sn-042_1729314220.teardown [2024-10-19T05:05:01.060408+00:00]: INFO : Clearing Environment ✓
chk-obj-sn-042_1729314220 [2024-10-19T05:05:11.071086+00:00]: INFO : test ended ✓
```

Fig. 3. Passing Script Output

Note :

The tick at the end of each line signifies that the data model is stored or streamed based on the D2C mode set in the platform.

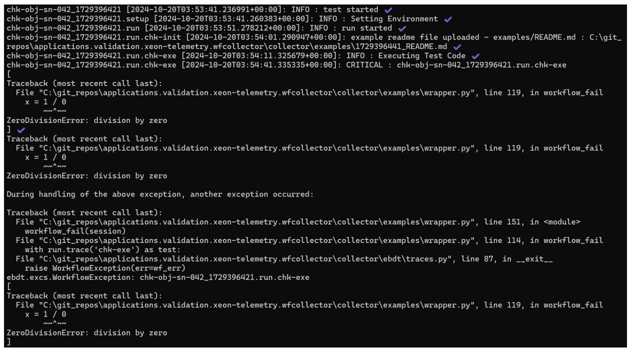
➢ Failing Script



```python
def workflow_fail(session):

    sn_nm = f'chk-obj-sn-042_{int(datetime.now().timestamp())}'
    with session(sn_nm, {'Description': 'Sample Failing Run'}) as sn:

        sn.log.info('test started')
        # user code
        sn.metric(meter=Time).open(
            channels=['M_hour', 'M_minute', 'M_second'], interval=10)

        with sn.trace('setup') as setup:

            setup.log.info(f"Setting Environment")
            # user code
            time.sleep(10)

        with sn.trace('run') as run:

            run.log.info(f"run started")
            # user code
            time.sleep(10)

            with run.trace('chk-init') as init:

                file = "examples/README.md"
                init.pin.file(
                    desc=f"example readme file uploaded - {file}", path=file)
                # user code
                time.sleep(10)

            with run.trace('chk-exe') as test:

                test.log.info("Executing Test Code")
                # user code
                time.sleep(30)
                x = 1 / 0
                # user code
                test.log.info("Collecting results")

            with run.trace('chk-results') as results:

                # user code
                time.sleep(10)
                folder = "examples/chunks_test/"
                results.pin.folder(
                    desc=f"example folder uploaded as zip - {folder}", path=folder)

            # user code
            time.sleep(10)
            run.log.info(f"run Completed")

        with sn.trace('teardown') as teardown:

            teardown.log.info(f"Clearing Environment")
            # user code
            time.sleep(10)

        # user code
        sn.log.info('test ended')
```

Fig. 4. Failing Script Example

➢ Failing Output



```
chk-obj-sn-042_1729396421 [2024-10-20T03:53:41.236991+00:00]: INFO : test started ✓
chk-obj-sn-042_1729396421.setup [2024-10-20T03:53:41.260383+00:00]: INFO : Setting Environment ✓
chk-obj-sn-042_1729396421.run [2024-10-20T03:53:51.278212+00:00]: INFO : run started ✓
chk-obj-sn-042_1729396421.run.chk-init [2024-10-20T03:54:01.290947+00:00]: example readme file uploaded - examples/README.md : C:\git_
repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\1729396441_README.md ✓
chk-obj-sn-042_1729396421.run.chk-exe [2024-10-20T03:54:11.325679+00:00]: INFO : Executing Test Code ✓
chk-obj-sn-042_1729396421.run.chk-exe [2024-10-20T03:54:41.335335+00:00]: CRITICAL : chk-obj-sn-042_1729396421.run.chk-exe
[
Traceback (most recent call last):
  File "C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\wrapper.py", line 119, in workflow_fail
    x = 1 / 0
        ~~^~~
ZeroDivisionError: division by zero
] ✓
Traceback (most recent call last):
  File "C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\wrapper.py", line 119, in workflow_fail
    x = 1 / 0
        ~~^~~
ZeroDivisionError: division by zero

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\wrapper.py", line 151, in <module>
    workflow_fail(session)
  File "C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\wrapper.py", line 114, in workflow_fail
    with run.trace('chk-exe') as test:
  File "C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\ebdt\traces.py", line 87, in __exit__
    raise WorkflowException(err=wf_err)
ebdt.excs.WorkflowException: chk-obj-sn-042_1729396421.run.chk-exe
[
Traceback (most recent call last):
  File "C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\wrapper.py", line 119, in workflow_fail
    x = 1 / 0
        ~~^~~
ZeroDivisionError: division by zero
]
```

Fig. 5. Failing Script Output

5. EBDT-Dash Design

5.1 Home Page

You can edit platform list & date range to compare test sessions from different platforms during different time frames, click refresh to update the Gantt chart visualization based on selected inputs. Sessions in green color are passed sessions & those in red are failed ones. Zoom in to interested session and click on it to navigate to the session page.

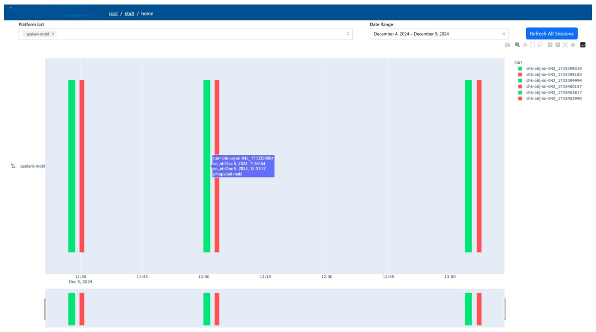
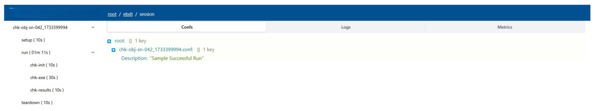


Fig. 6. Home page

5.2 Session Page

Session traces with their respective runtimes will be presented in hierarchical format in the left pane. With the conf file that is passed by the user at the start of the session, in the right pane.

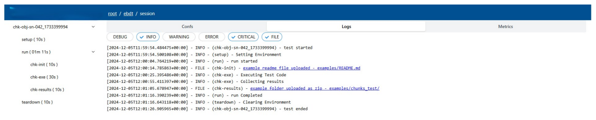


```
chk-obj-sn-042_1733399994
    setup ( 10s )
    run ( 01m 11s )
        chk-init ( 10s )
        chk-exe ( 30s )
        chk-results ( 10s )
    teardown ( 10s )
```

```
root:  {} 1 key
  chk-obj-sn-042_1733399994.conf:  {} 1 key
    Description:  "Sample Successful Run"
```

Fig. 7. Confs page with traces in left pane

Other details like logs and metrics can be found, by clicking on the Logs and metrics section, as shown in the page.

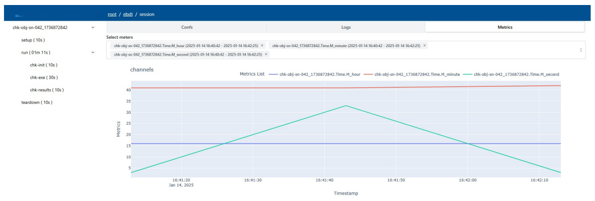


```
chk-obj-sn-042_1733399994
    setup ( 10s )
    run ( 01m 11s )
        chk-init ( 10s )
        chk-exe ( 30s )
        chk-results ( 10s )
    teardown ( 10s )
```

```
[2024-12-05T11:59:54.484475+00:00] - INFO - (chk-obj-sn-042_1733399994) - test started
[2024-12-05T11:59:54.500108+00:00] - INFO - (setup) - Setting Environment
[2024-12-05T12:00:04.764219+00:00] - INFO - (run) - run started
[2024-12-05T12:00:14.785863+00:00] - FILE - (chk-init) - example readme file uploaded - examples/README.md
[2024-12-05T12:00:25.395486+00:00] - INFO - (chk-exe) - Executing Test Code
[2024-12-05T12:00:55.411397+00:00] - INFO - (chk-exe) - Collecting results
[2024-12-05T12:01:05.678947+00:00] - FILE - (chk-results) - example folder uploaded as zip - examples/chunks_test/
[2024-12-05T12:01:16.390239+00:00] - INFO - (run) - run Completed
[2024-12-05T12:01:16.643118+00:00] - INFO - (teardown) - Clearing Environment
[2024-12-05T12:01:26.905965+00:00] - INFO - (chk-obj-sn-042_1733399994) - test ended
```

Fig. 8. Logs page with traces in left pane

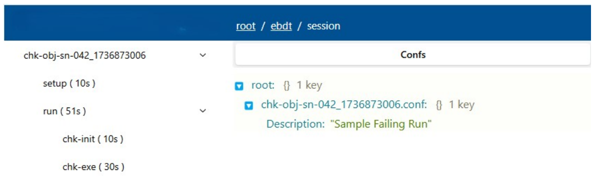
Log Lines can be selected based on categories to filter them, for convenience during troubleshooting. Also, the download links specified in File type log lines can be used for downloading the files from the test script.



```
chk-obj-sn-042_1736872842
    setup ( 10s )
    run ( 01m 11s )
        chk-init ( 10s )
        chk-exe ( 30s )
        chk-results ( 10s )
    teardown ( 10s )
```

Fig. 9. Metrics page with traces in left pane

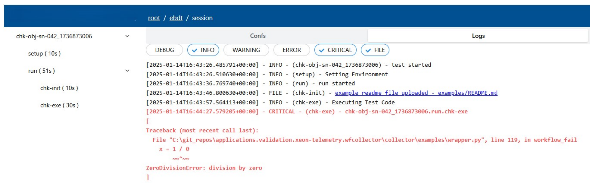
5.3 Failed Test Runs



```
chk-obj-sn-042_1736873006
    setup ( 10s )
    run ( 51s )
        chk-init ( 10s )
        chk-exe ( 30s )
```

```
root:  {} 1 key
  chk-obj-sn-042_1736873006.conf:  {} 1 key
    Description:  "Sample Failing Run"
```

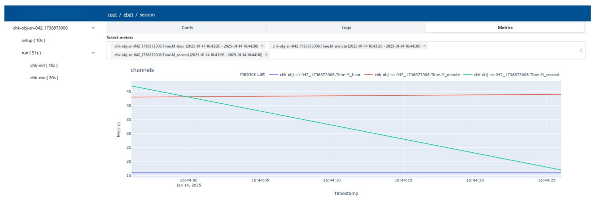
Fig. 10. Confs page with traces in left pane (failing case)



```
chk-obj-sn-042_1736873006
    setup ( 10s )
    run ( 51s )
        chk-init ( 10s )
        chk-exe ( 30s )
```

```
[2025-01-14T16:43:26.485791+00:00] - INFO - (chk-obj-sn-042_1736873006) - test started
[2025-01-14T16:43:26.510630+00:00] - INFO - (setup) - Setting Environment
[2025-01-14T16:43:36.769740+00:00] - INFO - (run) - run started
[2025-01-14T16:43:46.800630+00:00] - FILE - (chk-init) - example readme file uploaded - examples/README.md
[2025-01-14T16:43:57.564113+00:00] - INFO - (chk-exe) - Executing Test Code
[2025-01-14T16:44:27.579205+00:00] - CRITICAL - (chk-exe) - chk-obj-sn-042_1736873006.run.chk-exe
[
Traceback (most recent call last):
  File "C:\git_repos\applications.validation.xeon-telemetry.wfcollector\collector\examples\wrapper.py", line 119, in workflow_fail
    x = 1 / 0
        ~~^~~
ZeroDivisionError: division by zero
]
```

Fig. 11. Logs page with traces in left pane (failing case)



```
chk-obj-sn-042_1736873006
    setup ( 10s )
    run ( 51s )
        chk-init ( 10s )
        chk-exe ( 30s )
```

Fig. 12. Metrics page with traces in left pane (failing case)

6. EBDT-Panel & EBDT-Service

Panel can be used to enable or disable D2C mode of the running service.

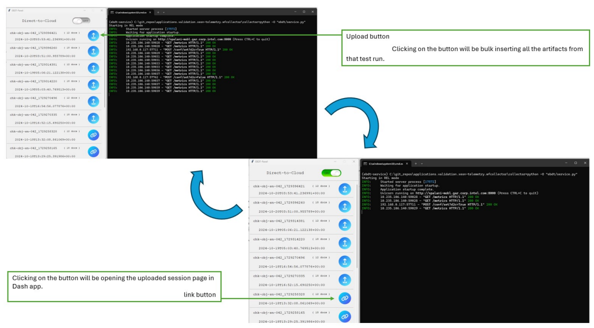
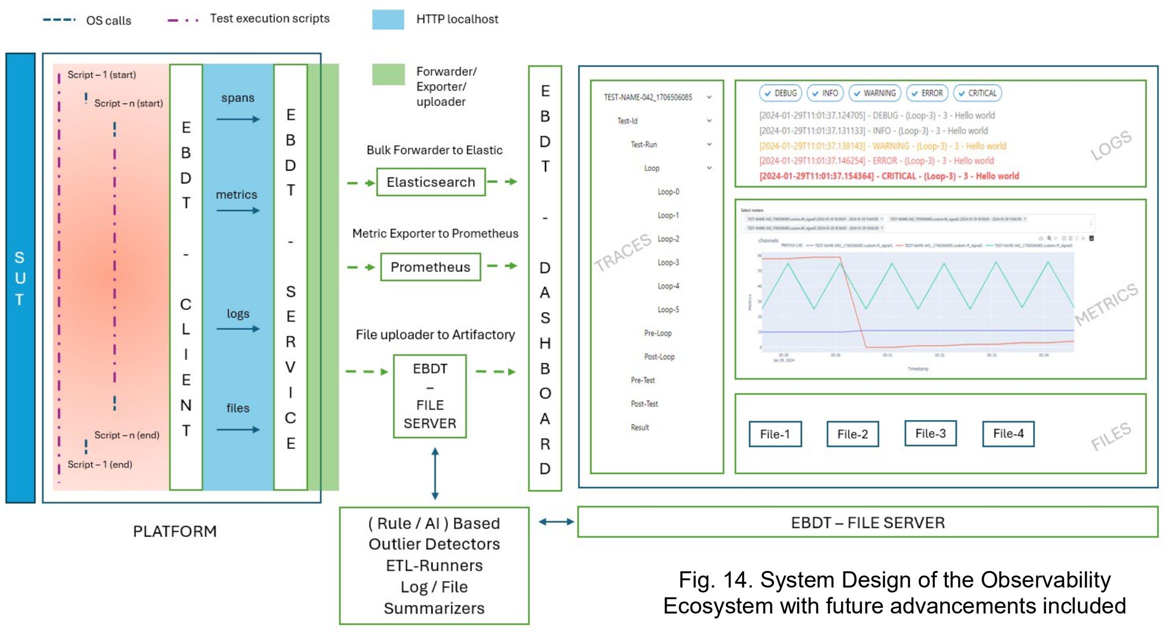

Direct-to-Cloud
Upload button
Clicking on the button will be bulk inserting all the artifacts from that test run.
Clicking on the button will be opening the uploaded session page in Dash app.
link button

Fig. 13. EBDT – Panel user interface usage instruction



```
[2024-01-29T11:01:37.124705] - DEBUG - (Loop-3) - 3 - Hello world
[2024-01-29T11:01:37.131133] - INFO - (Loop-3) - 3 - Hello world
[2024-01-29T11:01:37.138143] - WARNING - (Loop-3) - 3 - Hello world
[2024-01-29T11:01:37.146254] - ERROR - (Loop-3) - 3 - Hello world
[2024-01-29T11:01:37.154364] - CRITICAL - (Loop-3) - 3 - Hello world
```

Fig. 14. System Design of the Observability Ecosystem with future advancements included

7. Conclusion & Prospects

- The framework, methodology & suit of tools developed, makes data pipeline intact & debugging easy for the platform engineers, via dash web UI, thereby improving the pace of platform development & test execution.
- Trace durations-based aggregation across different platforms can be done for plotting gaussian distribution curve to implement outlier detections.
- Several ETL-runners can be deployed using ap_schedulers (i.e) cron jobs for processing the trace durations / log contents / files uploaded, based on specific needs to create static html reports using jinja 2 template engine to host in nginx server and send the links in mail channels.
- LLM based Log Analysers, File Analysers for summarization, etc can also be implemented based on need.

8. Acknowledgement

I'd like to acknowledge the time & energy spent by Sanjay Tiwari in providing valuable inputs, that have shaped the requirements of the project.

9. References

[1] Muhammed Usman, Simone Ferlin, Anna Brunstrom, Javid Taheri: "*A Survey on observability of Distributed Edge & Container Based Microservices* ", IEEE Access, 2022.

[2] Oleh V. Talaver, Tetiana A. Vakaliuk: "*Telemetry to solve dynamic analysis of a distributed system*", Journal of Edge Computing, 2024.

10. Glossary

EBDT:	Event Based Distributed Tracing
OTLP:	Open Telemetry Project