# HiQ - A Modern Observability System

*Release 1.0.0*

**Fuheng Wu**

**Apr 03, 2022**

HiQ is named after Henry Fuheng Wu, Ivan Davchev, Qian Jun.

Imagine there's no countries.

It isn't hard to do.

Nothing to kill or die for.

And no religion, too.

Imagine all the people.

Living life in peace...

--John Lennon

Thanks List: Kathan Patel, Wei Gao, Karunakar Chinnabathini, Kulbhushan Pachauri, Tiger Deng, Pingbo Zhang, Simo Lin, Jinguo Zhang

Special Thanks To Oleksandra For Reviewing The Project While Fighting The War And Helping Others.

Henry Fuheng Wu, Fremont CA, USA 2022

# TABLE OF CONTENTS

# CHAPTER 1

# HIQ BACKGROUND

HiQ is a library for software performance tracing, monitoring and optimization.

## 1.1 Monolithic Application vs. Distributed System and Microservice Architecture

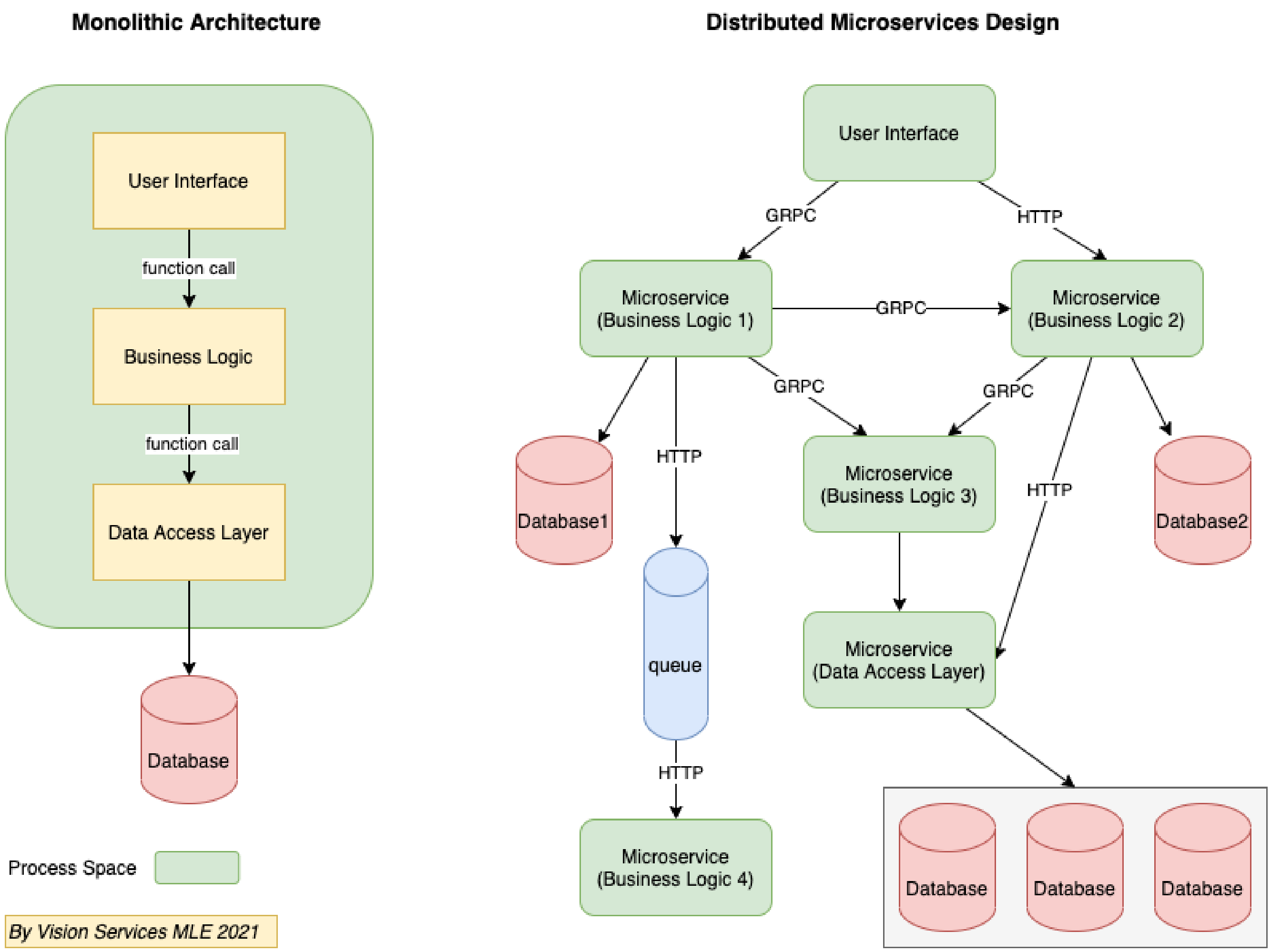
### 1.1.1 What is a monolithic architecture?

It’s a traditional approach to software development in which the entire system function is based on a single application as a single, autonomous unit. A helpful analogy here would be a large block of stone (a.k.a monolith). In software development, this single block would stand for a single platform.

In a monolithic app, all functions are managed and served in one place. Of course, an app has its inner structure consisting of a database, client-side interface, business logic, but it still remains an indivisible unit. Its components don’t require API to communicate.

### 1.1.2 What is a distributed/microservice architecture?

In a microservice architecture, business logic is broken down into lightweight, single-purpose self-sufficient services. As such, the infrastructure is akin to collection modules. Each service within this type of architecture is responsible for a specific business goal. In essence, the microservice architecture looks like a Lego construction, which can be decomposed into a number of modules. The interaction between the components of the system ensured by means of API.




## 1.2 Monitoring and Observability

Monitoring is tooling or a technical solution that allows teams to watch and understand the state of their systems. Monitoring is based on gathering predefined sets of metrics or logs.

Observability is tooling or a technical solution that allows teams to actively debug their system. Observability is based on exploring properties and patterns not defined in advance.

### 1.2.1 Blackbox monitoring

In a blackbox (or synthetic) monitoring system, input is sent to the system under examination in the same way a customer might. This might take the form of HTTP calls to a public API, or RPC calls to an exposed endpoint, or it might be calling for an entire web page to be rendered as a part of the monitoring process.

Blackbox monitoring is a `sampling-based method`. The same system that is responsible for user requests is monitored by the blackbox system. A blackbox system can also provide coverage of the target system's surface area. This could mean probing each external API method. You might also consider a representative mixture of requests to better mimic actual customer behavior. For example, you might perform 100 reads and only 1 write of a given API.

You can govern this process with a scheduling system, to ensure that these inputs are made at a sufficient rate in order to gain confidence in their sampling. Your system should also contain a validation engine, which can be as simple as checking response codes, or matching output with regular expressions, up to rendering a dynamic site in a headless browser and traversing its DOM tree, looking for specific elements. After a decision is made (pass, fail) on a given probe, you must store the result and metadata for reporting and alerting purposes. Examining a snapshot of a failure and its context can be invaluable for diagnosing an issue.

### 1.2.2 Whitebox Monitoring

Monitoring and observability rely on signals sent from the workload under scrutiny into the monitoring system. This can generally take the form of the three most common components: `metrics`, `logs`, and `traces`. Some monitoring systems also track and report events, which can represent user interactions with an entire system, or state changes within the system itself.

Metrics are simply measurements taken inside a system, representing the state of that system in a measurable way. These are almost always numeric and tend to take the form of counters, distributions, and gauges. There are some cases where string metrics make sense, but generally numeric metrics are used due to the need to perform mathematical calculations on them to form statistics and draw visualizations.

Logs can be thought of as append-only files that represent the state of a single thread of work at a single point in time. These logs can be a single string like "User pushed button X" or a structured log entry which includes metadata such as the time the event happened, what server was processing it, and other environmental elements. Sometimes a system which cannot write structured logs will produce a semi-structured string like `[timestamp] [server] message [code]` which can be parsed after the fact, as needed. Log processing can be a very reliable method of producing statistics that can be considered trustworthy, as they can be reprocessed based on immutable stored logs, even if the log processing system itself is buggy. Additionally, logs can be processed in real time to produce log-based metrics. In HiQ, LMK (LogMonkeyKing) is used to write the log entry.

Traces are often used in distributed system. Traces are composed of spans, which are used to follow an event or user action through a distributed system. A span can show the path of a request through one server, while another span might run in parallel, both having the same parent span. These together form a trace, which is often visualized in a waterfall graph similar to those used in profiling tools. This lets developers understand time taken in a system, across many servers, queues, and network hops. A common framework for this is OpenTelemetry, which was formed from both OpenCensus and OpenTracing. OpenTelemetry defines interface, but the implementations are in the specific software like `Zipkin`, `Jaeger`, or `Apache Skywalking`.

Metrics, logs, and traces can be reported to the monitoring system by the server under measurement, or by an adjacent agent that can witness or infer things about the system.

### 1.2.3 Instrumentation

To make use of a monitoring system, your system must be instrumented. In some cases, code need to be added to a system in order to expose its inner state. For example, if a simple program contains a pool of connections to another service, you might want to keep track of the size of that pool and the number of unused connections at any given time. In order to do so, a developer must write some code in the connection pool logic to keep track of when connections are formed or destroyed, when they are handed out, and when they are returned. This might take the form of log entries or events for each of these, or you might increment and decrement the metric for the size of the queue, or you might increment an absolute metric called connection_number each time a connection is created, or each time a pool is expanded. In other cases, like when you are using `HiQ`, you don't have to explicit instrument your code. HiQ will implicitly instrument your code without touching the target code.

## 1.3 Metrics

Metrics can be categorized into two types: business metrics and system metrics. Business metrics are quantified measures relavent to business logic and normally used to make business decision. System metrics are quantitative measures of the software system, such as latency, memory, CPU load, disk I/O, network I/O. HiQ is able to handle both metrics.

In monitoring and observability context, metrics, from another perspective, can be categorize into different types. Different software or organizations have different ways, for instance, GCP use 3 types way and they call it `Kind` instead of `type` , Prometheus uses 4 types. In HiQ, we only use two types only: abs metric and delta metric.

### 1.3.1 Abs

A abs metric, in which the value measures a specific instant in time. For example, metrics measuring CPU utilization are absolute metrics; each point records the CPU utilization at the time of measurement. Some other examples of a absolute metric are the current temperature, current time, and current memory resident set size.

### 1.3.2 Delta

A delta metric, aka relative metric, in which the value measures the change since it was last recorded. For example, metrics measuring request counts are delta metrics; each value records how many requests were received since the last data point was recorded. The delta is always the end value minus start value. Please be noted delta metric could be negative. Some other examples of a delta metric are the latency, memory cost, and network I/O traffic.

> Compared with Google and Prometheus' definition, HiQ abs metric is equivalent to Google and Prometheus's gauge metric, and HiQ's delta metric is equivalent to Google's delta and cumulative metrics and Prometheus's counter.

Ref:

- https://prometheus.io/docs/concepts/metric_types/

- https://cloud.google.com/monitoring/api/v3/kinds-and-types#metric-kinds

## 1.4 Application Performance Monitoring

APM (Application Performance Monitoring) provides a comprehensive set of features to monitor applications and diagnose performance issues. It has a very long history and covers very broad areas like including hardware performance monitoring. Although the name has word monitoring inside, it is more like an observability tool. It has become a profitable business for many companies and used frequently in sales and marketing context, like this one: Application Performance Monitoring Tools Reviews 2021 by Gartner. In early times, APM is more for monolitic applications, but now it has expanded to distributed systems.

## 1.5 Distributed Tracing

Distributed tracing, sometimes called distributed request tracing, is a method to monitor applications built on a microservices architecture.

IT and DevOps teams use distributed tracing to follow the course of a request or transaction as it travels through the application that is being monitored. This allows them to pinpoint bottlenecks, bugs, and other issues that impact the application's performance.

In 2010, Google put online a paper, Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, which starts the new era of distributed tracing. 2019 started with the merge of OpenTracing and OpenCensus into OpenTelemetry, so that the industry started to have a unified standard for distributed tracing. Now all APM vendors provide distributed tracing features.

# CHAPTER 2

# HIQ CORE CONCEPTS

## 2.1 Target Code

The main program which we want to collect information about. It could be a runnable python code or a module.

## 2.2 Driver Code

HiQ driver code is like agent in most APM applications, but there is a little difference. With agent, a runnable application is needed, so that the agent can attach to it. But driver code can work with modules too. For instance, you can write python function in driver code to call another target function in the target module.

## 2.3 HiQ Tracing Class/Object

HiQ provides two Tracing Class out of the box: `HiQLatency` for latency tracing and `HiQMemory` for memory tracing. You can derive from `HiQSimple` to have your own customized tracing. These classes are called `HiQ Tracing Class` and the object is called `HiQ Tracing Object`.

## 2.4 LumberJack/Jack

LumberJack is a process to collect traces, HiQ trees in this case, to send to HiQ server. To enable LumberJack, set environment variable `JACK` to 1.

## 2.5 Log Monkey King

Log Monkey King is a process to write traditional semi-structured, append-only log into log files. To enable Log Monkey King, set environment variable `LMK` to 1.

## 2.6 HiQ Tree

HiQ tree is a n-ary tree, plus a stack and dictionary/map. Different from the traditional BST, AVL, RB Tree, the tree is a strictly insertion-time-ordered tree from top to bottom and from left to right, so you can not switch the order of the nodes. The purpose of the tree is not for searching, or sorting. It is for visualizing program execution and facilitating code optimization. The values inserted into the tree doesn't need to be monotonically increasing.

Every node in an HiQ tree has a start value and a end value. `end` value minus `start` value is equal to the span of the node, or sometimes you can just call the node itself as a `span` to confirm with OpenTracing conventions.

HiQ tree has three `mode`s. When HiQ tree is in `concise` mode, which is the default mode, HiQ tree will not contain ZSP(zero-span node). When the mode is `verbose` mode, HiQ tree can have ZSP if there is no extra information in the node, like exception information. When the mode is `debug`, all the zero span node will be recorded as well.

## 2.7 HiQ Conf

HiQ conf could be a text configuration file to specify the functions you want to trace. It can be json or CSV file.

A sample json file is like:

```
[
    {
        "name": "f1",
        "module": "my_model2",
        "function": "func1",
        "class": ""
    },
    {
        "name": "f2",
        "module": "my_model2",
        "function": "func2",
```

(continues on next page)

(continued from previous page)

```
            "class": ""
        },
        {
            "name": "f3",
            "module": "my_model2",
            "function": "func3",
            "class": ""
        },
        {
            "name": "f4",
            "module": "my_model2",
            "function": "func4",
            "class": ""
        }
    ]
```

A sample csv file is like:

```
    "my_model2", "", "func1", "f1"
    "my_model2", "", "func2", "f2"
    "my_model2", "", "func3", "f3"
    "my_model2", "", "func4", "f4"
```

Also you can also use a list of list to represent it. For example, an equivalent representation of the above json and csv file is:

```
[
    ["my_model2", "", "func1", "f1"],
    ["my_model2", "", "func2", "f2"],
    ["my_model2", "", "func3", "f3"],
    ["my_model2", "", "func4", "f4"]
]
```

The inner list must have length of 4. They are: `[module_name, class_name, function_name, tag_name]`. The tag name will display in the HiQ as the tree node name.

The following example shows how to use HiQ conf.

Target Code:

```python
import time


def func1():
    time.sleep(1.5)
    print("func1")
    func2()


def func2():
    time.sleep(2.5)
    print("func2")
```

(continues on next page)

(continued from previous page)

```


def main():
    func1()


if __name__ == "__main__":
    main()
```

Driver Code:

```
import hiq
import os

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with hiq.HiQStatusContext(debug=True):
        with hiq.HiQLatency(f"{here}/hiq.conf") as driver:
            hiq.mod("main").main()
            driver.show()


if __name__ == "__main__":
    run_main()
```

HIQ Conf:

```
"main", "", "main", "main"
"main", "", "func1", "func1"
"main", "", "func2", "func2"
```

Run the driver code and you will get something like:

```
□   python hiq/examples/conf/main_driver.py
func1
func2
[2021-11-03 22:51:08.946615 - 22:51:12.951082]  [100.00%] □_root_time(4.0045)
                                                            [OH:191us]
[2021-11-03 22:51:08.946615 - 22:51:12.951082]  [100.00%]    l___main(4.0045)
[2021-11-03 22:51:08.946663 - 22:51:12.951069]  [100.00%]       l___func1(4.0044)
[2021-11-03 22:51:10.448407 - 22:51:12.951018]  [ 62.50%]          l___func2(2.
↪5026)
```

## 2.8 Latency Overhead

All runtime monitoring has overhead, no matter latency or memory, CPU. In most cases, we care about latency overhead. Different from all the open source projects in the community and the products in the market, HiQ provides transparent latency overhead information out of the box.

In the quick start example, we can see the latency overhead is printed out under the tree’s root node, which is 163us, and equivalent to 0.04% of the total running time.

```
python main_driver.py
func1
func2
[2021-11-01 21:54:18.222424 - 21:54:22.226879]  [100.00%] _root_time(4.0045)
                                                           [OH:163us]
[2021-11-01 21:54:18.222424 - 21:54:22.226879]  [100.00%]   l___main(4.0045)
[2021-11-01 21:54:18.222472 - 21:54:22.226868]  [100.00%]      l___func1(4.0044)
[2021-11-01 21:54:19.724213 - 21:54:22.226818]  [ 62.50%]         l___func2(2.5026)
```

# CHAPTER 3

# HIQ TRACING TUTORIAL

Latency tracing is always enabled as long as global HiQ status is on. Other than latency, HiQ provides memory, disk I/O, network I/O, and Exception tracing out of the box.

## 3.1 Global HiQ Status

Global HiQ status is a cross-process boolean value that decide if HiQ running in the current machine is enabled or not. There are two functions to get and set the global HiQ status. You can get them from:

```python
from hiq.hiq_utils import get_global_hiq_status, set_global_hiq_status
```

The following is the demo code:

```python
from hiq.hiq_utils import get_global_hiq_status, set_global_hiq_status

if __name__ == "__main__":
    set_global_hiq_status(True)
    b = get_global_hiq_status()
    print(b)

    set_global_hiq_status(False)
    b = get_global_hiq_status()
    print(b)
```

Run it and you will get:

```
□   python examples/hiq_global_status/demo.py
□ set global hiq to True
True
```

(continues on next page)

(continued from previous page)

```
 set global hiq to False
False
```

If global HiQ status is False, all the HiQ in the machine is disabled. If it is True, you can call `disable()` to disable a specific HiQ Object. This is so-called `dynamic tracing`.

---

Note: We assume global HiQ status is already set to True in this tutorial.

---

Normally you don't have to call them directly. Instead you use context manager `hiq.HiQStatusContext()` to make sure the HiQ status is on or off.

**class hiq.HiQStatusContext**(target_status_on=True, debug=False)

An HiQ context manager

Inside HiQStatusContext, HiQ status is always enabled unless the status is changed in other processes(HiQ status is not guarded).

```
>>> from hiq import HiQStatusContext
>>> with HiQStatusContext():
        # HiQ will be enabled inside the `with` block, and reverted␣
↪to original value out of the block
>>> with HiQStatusContext(target_status_on=False):
        # HiQ will be disabled inside the `with` block, and reverted␣
↪to original value out of the block
```

**__init__**(target_status_on=True, debug=False)

Constructor of HiQStatusContext

Parameters

- **target_status_on** (*bool, optional*) –set the target HiQ status you want to set in the context manager. Defaults to True.
- **debug** (*bool, optional*) –print more information when debug is True. Defaults to False.

---

Tip: hiq.HiQStatusContext() is the best practice to use whenever possible.

---

## 3.2 Dynamic Tracing

HiQ tracing is dynamic, which means you can enable and disable it as needed. The following is a simple example.

You can disable and enable HiQ tracing at run time.

```
import hiq
import time
```

(continues on next page)

(continued from previous page)

```python


def run_main():
    # create an `hiq.HiQLatency` object and HiQ is enabled by default
    with hiq.HiQStatusContext():
        driver = hiq.HiQLatency(
            hiq_table_or_path=[
                ["main", "", "main", "main"],
                ["main", "", "func1", "func1"],
                ["main", "", "func2", "func2"],
            ]
        )
        print("*" * 20, "HiQ is enabled", "*" * 20)
        start = time.time()
        hiq.mod("main").main()
        print(f"{time.time()-start} second")
        driver.show()

        # disable HiQ in `driver`
        print("*" * 20, "disable HiQ", "*" * 20)
        driver.disable_hiq(reset_trace=True)
        start = time.time()
        hiq.mod("main").main()
        print(f"{time.time()-start} second")
        driver.show()

        # enable HiQ in `driver` again
        print("*" * 20, "re-enable HiQ", "*" * 20)
        driver.enable_hiq(reset_trace=True)
        start = time.time()
        hiq.mod("main").main()
        print(f"{time.time()-start} second")
        driver.show()


if __name__ == "__main__":
    run_main()
```

With this code above, we disable and enable HiQ tracing, and run the `main()` function. The result is like:

```
HIQ_STATUS_CACHED=1 python examples/dynamic/main_driver.py
******************** HiQ is enabled ********************
func1
func2
4.004539489746094 second
[2021-11-03 00:32:52.871352 - 00:32:56.875782]  [100.00%] _root_time(4.0044)
                                                            [OH:279us]
[2021-11-03 00:32:52.871352 - 00:32:56.875782]  [100.00%]    l___main(4.0044)
[2021-11-03 00:32:52.871442 - 00:32:56.875764]  [100.00%]       l___func1(4.0043)
[2021-11-03 00:32:54.373086 - 00:32:56.875699]  [ 62.50%]          l___func2(2.5026)

******************** disable HiQ ********************
func1
func2
4.004141569137573 second
******************** re-enable HiQ ********************
func1
func2
4.004455804824829 second
[2021-11-03 00:33:00.881389 - 00:33:04.885762]  [100.00%] _root_time(4.0044)
                                                            [OH:192us]
[2021-11-03 00:33:00.881389 - 00:33:04.885762]  [100.00%]    l___main(4.0044)
[2021-11-03 00:33:00.881409 - 00:33:04.885745]  [100.00%]       l___func1(4.0043)
[2021-11-03 00:33:02.383059 - 00:33:04.885686]  [ 62.50%]          l___func2(2.5026)
```

The environment variable `HIQ_STATUS_CACHED` decide if the result is cached. If it is enabled, the result will be cached for `5 seconds`.

## 3.3 Metrics Customization

HiQ supports metrics customization. You can choose to trace different metrics in HiQ tree.

### 3.3.1 ExtraMetrics

Now HiQ supports 3 types of customized metrics: `ExtraMetrics.FILE`, `ExtraMetrics.FUNC`, `ExtraMetrics.ARGS`. You can pass them in a set object to `extra_metrics` in the constructor like below. And of course, different metrics have different latency overheads, which you can find in HiQ tree as well.

Target Code:

```python
import time


def func1(x, y):
    time.sleep(1.5)
    func2(y)


```

(continues on next page)

(continued from previous page)

```
def func2(y):
    time.sleep(2.5)


def main(x, y):
    func1(x, y)


if __name__ == "__main__":
    main(1, 2)
```

Driver Code:

```
import hiq
import os
from hiq.constants import ExtraMetrics

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with hiq.HiQStatusContext(debug=False):
        driver1 = hiq.HiQLatency(
            f"{here}/hiq.conf",
            extra_metrics={ExtraMetrics.FILE},
        )
        hiq.mod("main").main(1, 2)
        driver1.show()
        driver1.disable_hiq()

        driver2 = hiq.HiQLatency(
            f"{here}/hiq.conf",
            extra_metrics={ExtraMetrics.FUNC},
        )
        hiq.mod("main").main(1, 2)
        driver2.show()
        driver2.disable_hiq()

        driver3 = hiq.HiQLatency(
            f"{here}/hiq.conf",
            extra_metrics={ExtraMetrics.ARGS},
        )
        hiq.mod("main").main(1, 2)
        driver3.show()
        driver3.disable_hiq()

        driver4 = hiq.HiQLatency(
            f"{here}/hiq.conf",
            extra_metrics={
                ExtraMetrics.FILE,
                ExtraMetrics.FUNC,
```

(continues on next page)

(continued from previous page)

```
                ExtraMetrics.ARGS,
            },
        )
        hiq.mod("main").main(1, 2)
        driver4.show()


if __name__ == "__main__":
    run_main()
```

Note: If we create one more driver for the same target, we need to disable the previous driver by calling driver.disable_hiq(), otherwise an exception will be raised.

Run this file and the output will be like:

```
□   python examples/extra/simple/main_driver.py
[2021-11-07 19:46:47.464262 - 19:46:51.476240]  [100.00%] □_root_time(4.0120)
                                                            [OH:38767us]
[2021-11-07 19:46:47.464262 - 19:46:51.476240]  [100.00%]    l___main(4.0120) ({'file': 'examples/extra/simple/main_driver.py:10'})
[2021-11-07 19:46:47.466069 - 19:46:51.476229]  [ 99.95%]       l___func1(4.0102) ({'file': 'examples/extra/simple/main.py:14'})
[2021-11-07 19:46:48.973780 - 19:46:51.476187]  [ 62.37%]          l___func2(2.5024) ({'file': 'examples/extra/simple/main.py:6'})

[2021-11-07 19:46:51.478223 - 19:46:55.488515]  [100.00%] □_root_time(4.0103)
                                                            [OH:6217us]
[2021-11-07 19:46:51.478223 - 19:46:55.488515]  [100.00%]    l___main(4.0103) ({'function': 'run_main'})
[2021-11-07 19:46:51.480468 - 19:46:55.488504]  [ 99.94%]       l___func1(4.0080) ({'function': 'main'})
[2021-11-07 19:46:52.985900 - 19:46:55.488467]  [ 62.40%]          l___func2(2.5026) ({'function': 'func1'})

[2021-11-07 19:46:55.490225 - 19:46:59.494639]  [100.00%] □_root_time(4.0044)
                                                            [OH:212us]
[2021-11-07 19:46:55.490225 - 19:46:59.494639]  [100.00%]    l___main(4.0044) ({'args': '[int](1),[int](2)'})
[2021-11-07 19:46:55.490282 - 19:46:59.494629]  [100.00%]       l___func1(4.0043) ({'args': '[int](1),[int](2)'})
[2021-11-07 19:46:56.992013 - 19:46:59.494591]  [ 62.50%]          l___func2(2.5026) ({'args': '[int](2)'})

[2021-11-07 19:46:59.496759 - 19:47:03.512220]  [100.00%] □_root_time(4.0155)
                                                            [OH:9936us]
[2021-11-07 19:46:59.496759 - 19:47:03.512220]  [100.00%]    l___main(4.0155) ({'args': '[int](1),[int](2)', 'file': 'examples/extra/simple/main_driver.py:28', 'function': 'run_main'})
[2021-11-07 19:46:59.500252 - 19:47:03.512210]  [ 99.91%]       l___func1(4.0120) ({'args': '[int](1),[int](2)', 'file': 'examples/extra/simple/main.py:14', 'function': 'main'})
```

(continues on next page)

(continued from previous page)

```
[2021-11-07 19:47:01.010409 - 19:47:03.512170]  [ 62.30%]         l___func2(2.
↪5018) ({'args': '[int](2)', 'file': 'examples/extra/simple/main.py:6', 'function
↪': 'func1'})
```

We can see when we enable `ExtraMetrics.FILE`, the file path and number name will be attached to the tree node. When we enable `ExtraMetrics.FUNC`, the caller function name will be attached to the tree node. When we enable `ExtraMetrics.ARGS`, the function argument type and value will be attached to the tree node. If we enable all of them, all the information will be attached, but we got the largest latency overhead.

### 3.3.2 Complex Data Type

Target Code:

```python
import time


def func1(x, y, df):
    time.sleep(1.5)
    func2(y)


def func2(y):
    time.sleep(2.5)


def main(x, y, df, lst, bytes, *args, **kwargs):
    func1(x, y, df)
```

Driver Code:

```python
import hiq
import os
import numpy as np
import pandas as pd
import torch
from hiq.constants import ExtraMetrics

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    a = torch.rand(2000, 3)
    b = np.random.rand(3, 2000)
    df = pd.DataFrame(np.random.randint(0, 100, size=(100, 4)), columns=list("ABCD
↪"))
    series = pd.date_range(start="2016-01-01", end="2020-12-31", freq="D")

    with hiq.HiQStatusContext(debug=False):
```

(continues on next page)

(continued from previous page)

```python
        with hiq.HiQLatency(
            f"{here}/hiq.conf",
            extra_metrics={ExtraMetrics.ARGS},
        ) as driver:
            hiq.mod("main").main(
                a,
                b,
                df,
                [1, 2, 3],
                b"abc",
                st=set({5, 6, 7}),
                dt={"a": 1},
                pd_time=series,
            )
            driver.show()


if __name__ == "__main__":
    run_main()
```

Run this file and the output will be like:

```
□   python examples/extra/complex/main_driver.py
[2021-11-07 19:51:05.408034 - 19:51:09.412475]  [100.00%] □_root_time(4.0044)
                                                            [OH:260us]
[2021-11-07 19:51:05.408034 - 19:51:09.412475]  [100.00%]    l___main(4.0044) ({'args': '[tensor](torch.Size([2000, 3])),[ndarry]((3, 2000)),[pandas]((100, 4)),[list<int>](3),[bytes](3)', 'kwargs': '{\'st\': \'[set](3)\', \'dt\': "[dict]([\'a\'])", \'pd_time\': \'[DatetimeIndex](1827)\'}'})
[2021-11-07 19:51:05.408108 - 19:51:09.412463]  [100.00%]       l___func1(4.0044) ({'args': '[tensor](torch.Size([2000, 3])),[ndarry]((3, 2000)),[pandas]((100, 4))'})
[2021-11-07 19:51:06.909852 - 19:51:09.412425]  [ 62.49%]          l___func2(2.5026) ({'args': '[ndarry]((3, 2000))'})
```

HiQ can handle all python built in types and third-party module' types including Pytorch tensor, Numpy NDArray, Pandas DataFrame and Series.

### 3.3.3 Large Data Structure

Tracing large data structure like arrays could be a performance killer. It will take a lot of CPU and some memory as well, and slow down the program. So this section is only recommended for use case where performance requirement is not that critical.

By default, HiQ trace the type and value of function arguments. For composite data structures, it traces the type and `size` instead of value. But sometimes, you may really need to know the data no matter how big it is. In this case, you can pass your own function arguments handler When creating HiQ Tracing Object.

With the same target code as above, we can have this driver code to save large data to hard disk:

```python
import os
import pickle

import hiq
import numpy as np
import pandas as pd
import torch
from hiq.constants import ExtraMetrics
from hiq.utils import write_file

here = os.path.dirname(os.path.realpath(__file__))


def large_data_processor(x, func_name=None) -> str:
    if func_name == "__main":
        if isinstance(x, tuple):
            write_file("/tmp/main.args.log", x[2].to_string(), append=True)
        elif isinstance(x, dict):
            with open("/tmp/main.args.pkl", "wb") as handle:
                pickle.dump(x, handle, protocol=pickle.HIGHEST_PROTOCOL)
        return "..."
    else:
        return hiq.hiq_utils.func_args_handler(x, func_name)


def run_main():
    a = torch.rand(2000, 3)
    b = np.random.rand(3, 2000)
    df = pd.DataFrame(np.random.randint(0, 100, size=(100, 4)), columns=list("ABCD
↪"))
    series = pd.date_range(start="2016-01-01", end="2020-12-31", freq="D")

    with hiq.HiQStatusContext(debug=False):
        with hiq.HiQLatency(
            f"{here}/hiq.conf",
            extra_metrics={ExtraMetrics.ARGS},
            func_args_handler=large_data_processor,
        ) as driver:
            hiq.mod("main").main(
                a,
                b,
                df,
                [1, 2, 3],
                b"abc",
                st=set({5, 6, 7}),
                dt={"a": 1},
                pd_time=series,
            )
            driver.show()


if __name__ == "__main__":
    run_main()
```

Run the code and we' ll get something like:

```
[2021-11-08 00:17:23.378755 - 00:17:27.383362]  [100.00%] □_root_time(4.0046)
                                                            [OH:8027us]
[2021-11-08 00:17:23.378755 - 00:17:27.383362]  [100.00%]    l___main(4.0046) ({
↪'args': '...', 'kwargs': '...'})
[2021-11-08 00:17:23.378954 - 00:17:27.383350]  [ 99.99%]       l___func1(4.0044)␣
↪({'args': '[tensor](torch.Size([2000, 3])),[ndarry]((3, 2000)),[pandas]((100,␣
↪4))'})
[2021-11-08 00:17:24.880710 - 00:17:27.383292]  [ 62.49%]          l___func2(2.
↪5026) ({'args': '[ndarry]((3, 2000))'})
```

The argument `df` has been saved into a file. To verify it:

```
□   cat /tmp/main.args.log |wc -l
100
```

The output 100 matches the row number 100 in line 29.

The entire `kwargs` has been pickled into `/tmp/main.args.pkl`. To verify the values:

```
>>> import pickle
>>> x = pickle.load(open('/tmp/main.args.pkl','rb'))
>>> x
{'st': {5, 6, 7}, 'dt': {'a': 1}, 'pd_time': DatetimeIndex(['2016-01-01', '2016-
↪01-02', '2016-01-03', '2016-01-04',
               '2016-01-05', '2016-01-06', '2016-01-07', '2016-01-08',
               '2016-01-09', '2016-01-10',
               ...
               '2020-12-22', '2020-12-23', '2020-12-24', '2020-12-25',
               '2020-12-26', '2020-12-27', '2020-12-28', '2020-12-29',
               '2020-12-30', '2020-12-31'],
               dtype='datetime64[ns]', length=1827, freq='D')}
```

## 3.4 Memory Tracing

```python
import hiq
import os
from hiq.constants import KEY_MEMORY, FORMAT_DATETIME

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with hiq.HiQStatusContext():
        driver = hiq.HiQMemory(f"{here}/hiq.conf")
        hiq.mod("main").main()
        driver.get_metrics(metrics_key=KEY_MEMORY)[0].show()


```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    run_main()
```

Output:

```
□   python examples/memory/main_driver.py
func1
func2
[      19.457 -       19.461]  [100.00%] □_root_get_memory_mb(0.0039)
[      19.457 -       19.461]  [100.00%]    l___main(0.0039)
```

The memory here means RSS memory. From the example above, we can see the memory is increased from 19.457MB to 19.461MB before and after the main function invocation. And the two functions `func1` and `func2` don't consume extra memory because we don't see them in the output. The reason why we don't see them is they are `zero span node`.

### 3.4.1 Timestamp With Non-latency Metrics

Unlike the latency metrics, memory is not related to time, so we don't see any timestamp in above output, which is not convenient for our debugging. For non-latency metrics, to get timestamp in the output, we should add `attach_timestamp=True` in `hiq.HiQMemory`'s constructor.

Note: This works for all non-latency metrics like memory, disk I/O, network I/O etc.

```
import hiq
import os
from hiq.constants import KEY_MEMORY, FORMAT_DATETIME

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with hiq.HiQStatusContext():
        driver = hiq.HiQMemory(f"{here}/hiq.conf", attach_timestamp=True)
        hiq.mod("main").main()
        driver.get_metrics(metrics_key=KEY_MEMORY)[0].show()


if __name__ == "__main__":
    run_main()
```

The result becomes:

```
$ python examples/memory/main_driver2.py
func1
func2
```

(continues on next page)

(continued from previous page)

```
                                   [      219.582 -       219.590]  [100.00%] □_
↪root_get_memory_mb(0.0078)
[1636877696.769 - 1636877700.774]  [      219.582 -       219.590]  [100.00%]    l__
↪_main(0.0078)
```

We can change the date time format by specify `time_format=FORMAT_DATETIME` in the `show` function. The new driver code is like:

```python
import hiq
import os
from hiq.constants import KEY_MEMORY, FORMAT_DATETIME

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with hiq.HiQStatusContext():
        driver = hiq.HiQMemory(f"{here}/hiq.conf", attach_timestamp=True)
        hiq.mod("main").main()
        driver.get_metrics(metrics_key=KEY_MEMORY)[0].show(time_format=FORMAT_
↪DATETIME)


if __name__ == "__main__":
    run_main()
```

In the new output below, we can see the datetime time format has changed:

```
$ python examples/memory/main_driver3.py
func1
func2
                                                    [     219.500 -      219.508]  [100.
↪00%] □_root_get_memory_mb(0.0078)
[2021-11-14 08:18:02.419058 - 08:18:06.423343]  [     219.500 -      219.508]  [100.
↪00%]    l___main(0.0078)
```

## 3.5 Disk I/O Tracing

Target Code:

```python
import os, time
from hiq.utils import execute_cmd, random_str


def create_and_read(k=102400):
    time.sleep(2)
    _100mb_file = "/tmp/" + random_str() + ".bin"
    if not os.path.exists(_100mb_file):
        execute_cmd(
```

(continues on next page)

(continued from previous page)

```
            f"dd if=/dev/zero of={_100mb_file} bs=1024 count={k}", verbose=False
        )
    with open(_100mb_file) as f:
        s = f.read()
        print(f"□ read file size: {len(s)} bytes")


def fun1():
    time.sleep(2)
    create_and_read(k=3)
    fun2()


def fun2():
    time.sleep(1)
    create_and_read(k=2)


def main():
    fun1()


if __name__ == "__main__":
    main()
```

Driver Code:

```
import hiq
from hiq.constants import *


def run_main():
    with hiq.HiQStatusContext():
        driver = hiq.HiQLatency(
            hiq_table_or_path=[
                ["main", "", "main", "main"],
                ["main", "", "create_and_read", "cr"],
                ["main", "", "fun1", "f1"],
                ["main", "", "fun2", "f2"],
            ],
            extra_hiq_table=[TAU_TABLE_DIO_RD],
        )
        hiq.mod("main").main()
        driver.show()


if __name__ == "__main__":
    run_main()
```

Run the driver code and get the output:

```
□   python hiq/examples/io_disk/main_driver.py
□ read file size: 3072 bytes
□ read file size: 2048 bytes
[2021-11-03 22:45:37.416571 - 22:45:44.432328]  [100.00%] □_root_time(7.0158)
                                                            [OH:552us]
[2021-11-03 22:45:37.416571 - 22:45:44.432328]  [100.00%]    └___main(7.0158)
[2021-11-03 22:45:37.416641 - 22:45:44.432315]  [100.00%]       └___f1(7.0157)
[2021-11-03 22:45:39.418850 - 22:45:41.424977]  [ 28.59%]          |___cr(2.0061)
[2021-11-03 22:45:41.424852 - 22:45:41.424904]  [  0.00%]          |  └___dio_r(0.0001)
[2021-11-03 22:45:41.425046 - 22:45:44.432301]  [ 42.86%]          └___f2(3.0073)
[2021-11-03 22:45:42.426265 - 22:45:44.432281]  [ 28.59%]             └___cr(2.0060)
[2021-11-03 22:45:44.432160 - 22:45:44.432212]  [  0.00%]                └___dio_r(0.0001)

[   0.000 - 5120.000]  [100.00%] □_root_get_io_bytes_r(5120.0000)
[   0.000 - 5120.000]  [100.00%]    └___main(5120.0000)
[   0.000 - 5120.000]  [100.00%]       └___f1(5120.0000)
[   0.000 - 3072.000]  [ 60.00%]          |___cr(3072.0000)
[   0.000 - 3072.000]  [ 60.00%]          |  └___dio_r(3072.0000)
[3072.000 - 5120.000]  [ 40.00%]          └___f2(2048.0000)
[3072.000 - 5120.000]  [ 40.00%]             └___cr(2048.0000)
[3072.000 - 5120.000]  [ 40.00%]                └___dio_r(2048.0000)
```

## 3.6 System I/O Tracing

The following target code creates a 3KB file in `fun1()` and a 2KB file in `fun2()` and then use `os.read`, which invokes linux system call `read()`, to read 50 bytes through file descriptor. HiQ can trace the I/O traffic of linux system call `read()` and `write()`.

Target Code:

```python
import os, time
from hiq.utils import execute_cmd, random_str


def create_and_read(k=102400):
    time.sleep(2)
    _100mb_file = "/tmp/" + random_str() + ".bin"
    if not os.path.exists(_100mb_file):
        execute_cmd(
            f"dd if=/dev/zero of={_100mb_file} bs=1024 count={k}", verbose=False
        )
    fd = os.open(_100mb_file, os.O_RDONLY)
    readBytes = os.read(fd, 50)
    os.close(fd)


def fun1():
```

(continues on next page)

(continued from previous page)

```
    time.sleep(2)
    create_and_read(k=3)
    fun2()


def fun2():
    time.sleep(1)
    create_and_read(k=2)


def main():
    fun1()


if __name__ == "__main__":
    main()
```

We can trace the system I/O by adding `HIQ_TABLE_SIO_RD` for read or `HIQ_TABLE_SIO_WT` for write. The following is the driver code:

```
import hiq
from hiq.constants import HIQ_TABLE_SIO_RD


def run_main():
    with hiq.HiQStatusContext():
        driver = hiq.HiQLatency(
            hiq_table_or_path=[
                ["main", "", "main", "main"],
                ["main", "", "create_and_read", "cr"],
                ["main", "", "fun1", "f1"],
                ["main", "", "fun2", "f2"],
            ],
            extra_hiq_table=[HIQ_TABLE_SIO_RD],
        )
        hiq.mod("main").main()
        driver.show()


if __name__ == "__main__":
    run_main()
```

Run the driver code and get the output:

```
□   python examples/io_sys/main_driver.py
[2021-11-04 02:56:27.995306 - 02:56:35.008258]  [100.00%] □_root_time(7.0130)
                                                            [OH:896us]
[2021-11-04 02:56:27.995306 - 02:56:35.008258]  [100.00%]    l___main(7.0130)
[2021-11-04 02:56:27.995369 - 02:56:35.008245]  [100.00%]       l___f1(7.0129)
[2021-11-04 02:56:29.997583 - 02:56:32.002374]  [ 28.59%]          |___cr(2.0048)
[2021-11-04 02:56:32.001401 - 02:56:32.001539]  [  0.00%]          |   |___sio_r(0.
↪0001)
```

(continues on next page)

(continued from previous page)

```
[2021-11-04 02:56:32.002117 - 02:56:32.002136]  [  0.00%]          |  |___sio_r(0.
↪0000)
[2021-11-04 02:56:32.002340 - 02:56:32.002354]  [  0.00%]          |  └___sio_r(0.
↪0000)
[2021-11-04 02:56:32.002420 - 02:56:35.008234]  [ 42.86%]          └___f2(3.0058)
[2021-11-04 02:56:33.003664 - 02:56:35.008218]  [ 28.58%]             └___cr(2.
↪0046)
[2021-11-04 02:56:35.007247 - 02:56:35.007400]  [  0.00%]                |___sio_
↪r(0.0002)
[2021-11-04 02:56:35.007963 - 02:56:35.007983]  [  0.00%]                |___sio_
↪r(0.0000)
[2021-11-04 02:56:35.008180 - 02:56:35.008200]  [  0.00%]                └___sio_
↪r(0.0000)

[0.000  - 100.000]  [100.00%] □_root_get_sio_bytes_r(100.0000)
[0.000  - 100.000]  [100.00%]    └___main(100.0000)
[0.000  - 100.000]  [100.00%]       └___f1(100.0000)
[0.000  -  50.000]  [ 50.00%]          |___cr(50.0000)
[0.000  -  50.000]  [ 50.00%]          |  └___sio_r(50.0000)
[50.000 - 100.000]  [ 50.00%]          └___f2(50.0000)
[50.000 - 100.000]  [ 50.00%]             └___cr(50.0000)
[50.000 - 100.000]  [ 50.00%]                └___sio_r(50.0000)
```

## 3.7 Network I/O Tracing

Target Code:

```python
import os
import time
from hiq.utils import execute_cmd, random_str, download_from_http

count = 0

here = os.path.dirname(os.path.realpath(__file__))


def create_and_read(k=102400):
    _100mb_file = "/tmp/" + random_str() + ".bin"
    if not os.path.exists(_100mb_file):
        execute_cmd(f"dd if=/dev/zero of={_100mb_file} bs=1024 count={k}")
    with open(_100mb_file) as f:
        s = f.read()
        print(f"□ file size: {len(s)}, {s[len(s) // 2 - 1]}")


def func1():
    global count
    if count == 5:
        create_and_read(1024 * 10)
```

(continues on next page)

(continued from previous page)

```
        count += 1
        return
    elif count > 5:
        return
    count += 1
    func4()
    # print("func1")


def func2():
    # print("func2")
    time.sleep(0.1)
    func1()


def func3():
    # print("func3")
    time.sleep(0.12)
    func2()


def func4():
    # print("func4")
    if count == 0:
        create_and_read(1024 * 50)
    if count == 3:
        download_from_http(
            "https://www.gardeningknowhow.com/wp-content/uploads/2017/07/hardwood-
↪tree.jpg",
            "/tmp/tree.jpg",
        )
    time.sleep(0.2)
    func2()
    func3()


def func5():
    time.sleep(0.24)
    # print("let func5 raise exception")
    # raise Exception("o")


def fit(model="awesome_model", data="awesome_data"):
    print(f"{data=},{model=}")
    time.sleep(0.35)
    func4()


def predict(model="awesome_model", data="awesome_data"):
    print(f"{data=},{model=}")
    time.sleep(0.16)
    func5()
```

(continues on next page)

(continued from previous page)

```python


def main():
    fit(model="awesome_model_1", data="awesome_data_1")
    predict(model="awesome_model_2", data="awesome_data_2")


if __name__ == "__main__":
    main()
```

In `func4()`, when global variable `count` is equal to 3, it will download an image from the internet.

The image size is 199602 bytes as displayed below:

```
-rw-rw-r-- 1 ubuntu ubuntu 199602 May 13  2018 hardwood-tree.jpg
```

Driver Code:

```python
import hiq
from hiq.constants import *


def run_main():
```

(continues on next page)

(continued from previous page)

```
6      with hiq.HiQStatusContext():
7          driver = hiq.HiQLatency(
8              hiq_table_or_path=[
9                  ["main", "", "main", "main"],
10                 ["main", "", "func1", "func1"],
11                 ["main", "", "func2", "func2"],
12                 ["main", "", "func3", "func3"],
13                 ["main", "", "func4", "func4"],
14                 ["main", "", "func5", "func5"],
15             ],
16             extra_hiq_table=[TAU_TABLE_NIO_GET],
17         )
18         hiq.mod("main").main()
19         driver.show()
20
21
22 if __name__ == "__main__":
23     run_main()
```

Notice at line 15, we added a new line to track network ingress I/O. To track the egress traffic, you just need to replace `TAU_TABLE_NIO_GET` with `TAU_TABLE_NIO_WRT`.

Output:

```
[2021-11-03 08:25:53.510876 - 08:25:57.561308]  [100.00%] □_root_time(4.0504)
                                                            [OH:2074us]
[2021-11-03 08:25:53.510876 - 08:25:57.561308]  [100.00%]    └___main(4.0504)
[2021-11-03 08:25:53.861402 - 08:25:57.160576]  [ 81.45%]        |___func4(3.2992)
[2021-11-03 08:25:54.183760 - 08:25:56.940055]  [ 68.05%]        |   |___func2(2.
↪7563)
[2021-11-03 08:25:54.283967 - 08:25:56.940045]  [ 65.58%]        |   |  └___func1(2.
↪6561)
[2021-11-03 08:25:54.284018 - 08:25:56.940032]  [ 65.57%]        |   |     └___
↪func4(2.6560)
[2021-11-03 08:25:54.484393 - 08:25:56.719469]  [ 55.18%]        |   |        |___
↪func2(2.2351)
[2021-11-03 08:25:54.584729 - 08:25:56.719449]  [ 52.70%]        |   |        |  └__
↪_func1(2.1347)
[2021-11-03 08:25:54.584799 - 08:25:56.719430]  [ 52.70%]        |   |        |    ␣
↪└___func4(2.1346)
[2021-11-03 08:25:54.785170 - 08:25:56.498725]  [ 42.31%]        |   |        |    ␣
↪   |___func2(1.7136)
[2021-11-03 08:25:54.885402 - 08:25:56.498709]  [ 39.83%]        |   |        |    ␣
↪   |  └___func1(1.6133)
[2021-11-03 08:25:54.885453 - 08:25:56.498696]  [ 39.83%]        |   |        |    ␣
↪   |     └___func4(1.6132)
[2021-11-03 08:25:54.885522 - 08:25:54.906254]  [  0.51%]        |   |        |    ␣
↪   |        |___nio_get(0.0207)
[2021-11-03 08:25:55.106743 - 08:25:56.278137]  [ 28.92%]        |   |        |    ␣
↪   |        |___func2(1.1714)
[2021-11-03 08:25:55.206995 - 08:25:56.278122]  [ 26.44%]        |   |        |    ␣
↪   |        |  └___func1(1.0711)
```

(continues on next page)

(continued from previous page)

```
[2021-11-03 08:25:55.207054 - 08:25:56.278107]  [ 26.44%]       |  |        |    ␣
↪    |        |     l___func4(1.0711)
[2021-11-03 08:25:55.407383 - 08:25:56.057437]  [ 16.05%]       |  |        |    ␣
↪    |        |        |___func2(0.6501)
[2021-11-03 08:25:55.507616 - 08:25:56.057420]  [ 13.57%]       |  |        |    ␣
↪    |        |        |  l___func1(0.5498)
[2021-11-03 08:25:55.507676 - 08:25:56.057403]  [ 13.57%]       |  |        |    ␣
↪    |        |        |     l___func4(0.5497)
[2021-11-03 08:25:55.708037 - 08:25:55.836658]  [  3.18%]       |  |        |    ␣
↪    |        |        |        |___func2(0.1286)
[2021-11-03 08:25:55.808240 - 08:25:55.836573]  [  0.70%]       |  |        |    ␣
↪    |        |        |        |  l___func1(0.0283)
[2021-11-03 08:25:55.836824 - 08:25:56.057387]  [  5.45%]       |  |        |    ␣
↪    |        |        |        l___func3(0.2206)
[2021-11-03 08:25:55.957082 - 08:25:56.057372]  [  2.48%]       |  |        |    ␣
↪    |        |        |           l___func2(0.1003)
[2021-11-03 08:25:56.057324 - 08:25:56.057343]  [  0.00%]       |  |        |    ␣
↪    |        |        |              l___func1(0.0000)
[2021-11-03 08:25:56.057490 - 08:25:56.278093]  [  5.45%]       |  |        |    ␣
↪    |        |        l___func3(0.2206)
[2021-11-03 08:25:56.177766 - 08:25:56.278078]  [  2.48%]       |  |        |    ␣
↪    |        |           l___func2(0.1003)
[2021-11-03 08:25:56.278022 - 08:25:56.278050]  [  0.00%]       |  |        |    ␣
↪    |        |              l___func1(0.0000)
[2021-11-03 08:25:56.278194 - 08:25:56.498680]  [  5.44%]       |  |        |    ␣
↪    |        l___func3(0.2205)
[2021-11-03 08:25:56.398421 - 08:25:56.498667]  [  2.47%]       |  |        |    ␣
↪    |           l___func2(0.1002)
[2021-11-03 08:25:56.498626 - 08:25:56.498643]  [  0.00%]       |  |        |    ␣
↪    |              l___func1(0.0000)
[2021-11-03 08:25:56.498771 - 08:25:56.719408]  [  5.45%]       |  |        |    ␣
↪    l___func3(0.2206)
[2021-11-03 08:25:56.619078 - 08:25:56.719386]  [  2.48%]       |  |        |    ␣
↪       l___func2(0.1003)
[2021-11-03 08:25:56.719326 - 08:25:56.719350]  [  0.00%]       |  |        |    ␣
↪          l___func1(0.0000)
[2021-11-03 08:25:56.719543 - 08:25:56.940020]  [  5.44%]       |  |        l___
↪func3(0.2205)
[2021-11-03 08:25:56.839789 - 08:25:56.940008]  [  2.47%]       |  |           l__
↪_func2(0.1002)
[2021-11-03 08:25:56.939970 - 08:25:56.939986]  [  0.00%]       |  |             ␣
↪l___func1(0.0000)
[2021-11-03 08:25:56.940103 - 08:25:57.160563]  [  5.44%]       |  l___func3(0.
↪2205)
[2021-11-03 08:25:57.060336 - 08:25:57.160552]  [  2.47%]       |     l___func2(0.
↪1002)
[2021-11-03 08:25:57.160520 - 08:25:57.160533]  [  0.00%]       |        l___
↪func1(0.0000)
[2021-11-03 08:25:57.320969 - 08:25:57.561265]  [  5.93%]       l___func5(0.2403)

[   0.000 - 199602.000]  [100.00%] ⌷_root_get_nio_bytes_r(199602.0000)
[   0.000 - 199602.000]  [100.00%]    l___main(199602.0000)
```

(continues on next page)

(continued from previous page)

```
[   0.000 - 199602.000]  [100.00%]       l___func4(199602.0000)
[   0.000 - 199602.000]  [100.00%]          l___func2(199602.0000)
[   0.000 - 199602.000]  [100.00%]             l___func1(199602.0000)
[   0.000 - 199602.000]  [100.00%]                l___func4(199602.0000)
[   0.000 - 199602.000]  [100.00%]                   l___func2(199602.0000)
[   0.000 - 199602.000]  [100.00%]                      l___func1(199602.0000)
[   0.000 - 199602.000]  [100.00%]                         l___func4(199602.0000)
[   0.000 - 199602.000]  [100.00%]                            l___func2(199602.
↪0000)
[   0.000 - 199602.000]  [100.00%]                               l___func1(199602.
↪0000)
[   0.000 - 199602.000]  [100.00%]                                  l___
↪func4(199602.0000)
[   0.000 - 199602.000]  [100.00%]                                     l___nio_
↪get(199602.0000)
```

We can see from the HiQ tree, network I/O get function `nio_get()` is called by called `func4` and the network traffic is 199602 bytes, and the downloading took 20.7 milliseconds.

## 3.8 Exception Tracing

HiQ provides exception tracing out of the box. By default, HiQ will populate the exception out until you catch it.

Target Code:

```python
import time


def func1():
    time.sleep(1.5)
    print("func1")
    func2()


def func2():
    time.sleep(2.5)
    print("func2")
    raise ValueError("an_exception")
    func3()


def func3():
    time.sleep(2.5)
    print("func3")


def main():
    func1()
```

(continues on next page)

(continued from previous page)

```


if __name__ == "__main__":
    main()
```

Driver Code 1:

```
import hiq
import os

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with hiq.HiQStatusContext():
        driver = hiq.HiQLatency(f"{here}/hiq.conf")
        try:
            hiq.mod("main").main()
        except Exception as e:
            print(e)
        driver.show()


if __name__ == "__main__":
    run_main()
```

Output:

```
   python examples/exception/main_driver.py
func1
func2
an_exception
[2021-11-03 17:17:03.547380 - 17:17:07.551894]  [100.00%] _root_time(4.0045)
                                                            [OH:121us]
[2021-11-03 17:17:03.547380 - 17:17:07.551894]  [100.00%]    l___main(4.0045) ({
↪'exception_summary': ValueError('an_exception')})
[2021-11-03 17:17:03.547442 - 17:17:07.551874]  [100.00%]       l___func1(4.0044)␣
↪({'exception_summary': ValueError('an_exception')})
[2021-11-03 17:17:05.049179 - 17:17:07.551824]  [ 62.50%]          l___func2(2.
↪5026) ({'exception_summary': ValueError('an_exception')})
```

You can also specify `fast_fail=False` when creating the HiQ object like `hiq.HiQLatency`, so that the exception will be silent and you get a concise HiQ tree.

Driver Code 2:

```
import hiq
import os

here = os.path.dirname(os.path.realpath(__file__))

```

(continues on next page)

(continued from previous page)

```python

def run_main():
    with hiq.HiQStatusContext():
        driver = hiq.HiQLatency(f"{here}/hiq.conf", fast_fail=False)
        hiq.mod("main").main()
        driver.show()


if __name__ == "__main__":
    run_main()
```

Output:

```
□   python examples/exception/main_driver2.py
func1
func2
[2021-11-03 17:22:18.648640 - 17:22:22.652281]  [100.00%] □_root_time(4.0036)
                                                           [OH:193us]
[2021-11-03 17:22:18.648640 - 17:22:22.652281]  [100.00%]    l___main(4.0036)
[2021-11-03 17:22:18.648686 - 17:22:22.652268]  [100.00%]       l___func1(4.0036)
[2021-11-03 17:22:20.150435 - 17:22:22.652231]  [ 62.49%]          l___func2(2.
↪5018)
```

## 3.9 Multiple Tracing

When HiQ is enabled and we call the target code more than one times, we will get multiple tracing results.

Target Code:

```python
import os
import time

from hiq.utils import download_from_http, execute_cmd, random_str


count = 0


def create_and_read(k=102400):
    _100mb_file = "/tmp/" + random_str() + ".bin"
    if not os.path.exists(_100mb_file):
        execute_cmd(
            f"dd if=/dev/zero of={_100mb_file} bs=1024 count={k}", verbose=False
        )
    with open(_100mb_file) as f:
        s = f.read()


def func1():
```

(continues on next page)

(continued from previous page)

```
    global count
    if count == 5:
        create_and_read(1024 * 10)
        count += 1
        return
    elif count > 5:
        return
    count += 1
    func4()


def func2():
    time.sleep(0.1)
    func1()


def func3():
    time.sleep(0.12)
    func2()


def func4():
    if count == 0:
        create_and_read(1024 * 5)
    if count == 3:
        download_from_http(
            "https://www.gardeningknowhow.com/wp-content/uploads/2017/07/hardwood-tree.jpg",
            "/tmp/tree.jpg",
        )
    time.sleep(0.2)
    func2()
    func3()


def func5():
    time.sleep(0.24)


def fit(model="awesome_model", data="awesome_data"):
    time.sleep(0.35)
    func4()


def predict(model="awesome_model", data="awesome_data"):
    time.sleep(0.16)
    func5()


def main():
    for i in range(4):
        fit(data={}, model=[i])
```

(continues on next page)

(continued from previous page)

```
        predict(model=f"awesome_model_{i}", data=i)


if __name__ == "__main__":
    main()
```

Driver Code:

```
1  import os
2  import hiq
3  import traceback, sys
4  from hiq.hiq_utils import get_global_hiq_status, set_global_hiq_status, HiQIdGenerator
5  from unittest.mock import MagicMock
6  
7  here = os.path.dirname(os.path.realpath(__file__))
8  
9  
10 def run_main():
11     _g_driver_original = get_global_hiq_status()
12     set_global_hiq_status(True)
13     driver = hiq.HiQLatency(
14         hiq_table_or_path=f"{here}/hiq.conf",
15         max_hiq_size=4,
16     )
17 
18     for i in range(3):
19         driver.get_tau_id = HiQIdGenerator()
20         try:
21             hiq.mod("main").fit(data={}, model=[i])
22         except Exception as e:
23             traceback.print_exc(file=sys.stdout)
24     driver.show(show_key=True)
25 
26     driver.disable_hiq()
27     print("-^" * 20, "disable HiQ", "-^" * 20)
28     hiq.mod("main").fit(data={}, model=[i])
29     set_global_hiq_status(_g_driver_original)
30 
31 
32 if __name__ == "__main__":
33     run_main()
```

From line 1 to 5: import necessary modules and functions. `get_global_hiq_status` and `set_global_hiq_status` are used to get and set the global hiq status. If the status is on, HiQ will function; if off, HiQ will stop working but you can still run the program.

Line 7: get the current directory path.

Line 10: define a function called `run_main`.

Line 11 to 12: back up the original HiQ status and set it to True

Line 13 to 16: create an object `driver` which has a type of class `hiq.HiQLatency`. `hiq.HiQLatency` is for latency tracking. We have `hiq.HiQMemory` to track both latency and memory. Users can also inherit `hiq.HiQSimple` to customize the metrics they want to track, but that is an advanced topics. For now, in this case, we just need `hiq.HiQLatency` to track latency.

Line 18 to 20: run the target code main.py's function `fit()` for 3 times.

Line 21: print the latency traces as trees.

Line 23: disable HiQ

Line 25: run target code main.py's function `fit()` once again.

Line 26: set the global hiq status back to what it was before this run

Run the driver code, you can get result like:

```
  python examples/multi-tracing/main_driver.py
 set global hiq to True
 k0: 0,  k1: time
[2021-11-03 19:38:38.528194 - 19:38:41.750020]  [100.00%] _root_time(3.2218)
                                                            [OH:3242us]
[2021-11-03 19:38:38.528194 - 19:38:41.750020]  [100.00%]    l___f4(3.2218)
[2021-11-03 19:38:38.745514 - 19:38:41.529168]  [ 86.40%]       |___f2(2.7837)
[2021-11-03 19:38:38.845949 - 19:38:41.529151]  [ 83.28%]       |  l___f1(2.6832)
[2021-11-03 19:38:38.846075 - 19:38:41.529131]  [ 83.28%]       |     l___f4(2.
↪6831)
[2021-11-03 19:38:39.046535 - 19:38:41.308501]  [ 70.21%]       |        |___f2(2.
↪2620)
[2021-11-03 19:38:39.146943 - 19:38:41.308482]  [ 67.09%]       |        |  l___
↪f1(2.1615)
[2021-11-03 19:38:39.147045 - 19:38:41.308462]  [ 67.09%]       |        |     l__
↪_f4(2.1614)
[2021-11-03 19:38:39.347496 - 19:38:41.087750]  [ 54.01%]       |        |        ␣
↪|___f2(1.7403)
[2021-11-03 19:38:39.447848 - 19:38:41.087731]  [ 50.90%]       |        |        ␣
↪|  l___f1(1.6399)
[2021-11-03 19:38:39.447920 - 19:38:41.087709]  [ 50.90%]       |        |        ␣
↪|     l___f4(1.6398)
[2021-11-03 19:38:39.694061 - 19:38:40.866945]  [ 36.40%]       |        |        ␣
↪|        |___f2(1.1729)
[2021-11-03 19:38:39.794394 - 19:38:40.866924]  [ 33.29%]       |        |        ␣
↪|        |  l___f1(1.0725)
[2021-11-03 19:38:39.794470 - 19:38:40.866904]  [ 33.29%]       |        |        ␣
↪|        |     l___f4(1.0724)
[2021-11-03 19:38:39.994862 - 19:38:40.646264]  [ 20.22%]       |        |        ␣
↪|        |        |___f2(0.6514)
[2021-11-03 19:38:40.095094 - 19:38:40.646241]  [ 17.11%]       |        |        ␣
↪|        |        |  l___f1(0.5511)
[2021-11-03 19:38:40.095146 - 19:38:40.646216]  [ 17.10%]       |        |        ␣
↪|        |        |     l___f4(0.5511)
[2021-11-03 19:38:40.295571 - 19:38:40.425075]  [  4.02%]       |        |        ␣
↪|        |        |        |___f2(0.1295)
[2021-11-03 19:38:40.395917 - 19:38:40.424979]  [  0.90%]       |        |        ␣
↪|        |        |        |  l___f1(0.0291)
```

(continues on next page)

(continued from previous page)

```
[2021-11-03 19:38:40.425275 - 19:38:40.646194]  [  6.86%]       |        |       ␣
↪|        |        |        l___f3(0.2209)
[2021-11-03 19:38:40.545736 - 19:38:40.646169]  [  3.12%]       |        |       ␣
↪|        |        |           l___f2(0.1004)
[2021-11-03 19:38:40.646097 - 19:38:40.646127]  [  0.00%]       |        |       ␣
↪|        |        |              l___f1(0.0000)
[2021-11-03 19:38:40.646342 - 19:38:40.866882]  [  6.85%]       |        |       ␣
↪|        |        l___f3(0.2205)
[2021-11-03 19:38:40.766563 - 19:38:40.866861]  [  3.11%]       |        |       ␣
↪|        |           l___f2(0.1003)
[2021-11-03 19:38:40.866809 - 19:38:40.866829]  [  0.00%]       |        |       ␣
↪|        |              l___f1(0.0000)
[2021-11-03 19:38:40.867007 - 19:38:41.087684]  [  6.85%]       |        |       ␣
↪|        l___f3(0.2207)
[2021-11-03 19:38:40.987283 - 19:38:41.087656]  [  3.12%]       |        |       ␣
↪|           l___f2(0.1004)
[2021-11-03 19:38:41.087594 - 19:38:41.087622]  [  0.00%]       |        |       ␣
↪|              l___f1(0.0000)
[2021-11-03 19:38:41.087820 - 19:38:41.308437]  [  6.85%]       |        |       ␣
↪l___f3(0.2206)
[2021-11-03 19:38:41.208148 - 19:38:41.308417]  [  3.11%]       |        |       ␣
↪   l___f2(0.1003)
[2021-11-03 19:38:41.308362 - 19:38:41.308385]  [  0.00%]       |        |       ␣
↪      l___f1(0.0000)
[2021-11-03 19:38:41.308562 - 19:38:41.529113]  [  6.85%]       |        l___f3(0.
↪2206)
[2021-11-03 19:38:41.428829 - 19:38:41.529091]  [  3.11%]       |           l___
↪f2(0.1003)
[2021-11-03 19:38:41.529046 - 19:38:41.529062]  [  0.00%]       |              l__
↪_f1(0.0000)
[2021-11-03 19:38:41.529225 - 19:38:41.749990]  [  6.85%]       l___f3(0.2208)
[2021-11-03 19:38:41.649575 - 19:38:41.749968]  [  3.12%]          l___f2(0.1004)
[2021-11-03 19:38:41.749906 - 19:38:41.749931]  [  0.00%]             l___f1(0.
↪0000)

 k0: 1,  k1: time
[2021-11-03 19:38:42.101156 - 19:38:42.622677]  [100.00%] _root_time(0.5215)
                                                            [OH:546us]
[2021-11-03 19:38:42.101156 - 19:38:42.622677]  [100.00%]    l___f4(0.5215)
[2021-11-03 19:38:42.301611 - 19:38:42.401940]  [ 19.24%]       |___f2(0.1003)
[2021-11-03 19:38:42.401887 - 19:38:42.401910]  [  0.00%]       |  l___f1(0.0000)
[2021-11-03 19:38:42.402009 - 19:38:42.622652]  [ 42.31%]       l___f3(0.2206)
[2021-11-03 19:38:42.522245 - 19:38:42.622633]  [ 19.25%]          l___f2(0.1004)
[2021-11-03 19:38:42.622575 - 19:38:42.622600]  [  0.00%]             l___f1(0.
↪0000)

 k0: 2,  k1: time
[2021-11-03 19:38:42.973617 - 19:38:43.495121]  [100.00%] _root_time(0.5215)
                                                            [OH:527us]
[2021-11-03 19:38:42.973617 - 19:38:43.495121]  [100.00%]    l___f4(0.5215)
[2021-11-03 19:38:43.173992 - 19:38:43.274265]  [ 19.23%]       |___f2(0.1003)
[2021-11-03 19:38:43.274217 - 19:38:43.274235]  [  0.00%]       |  l___f1(0.0000)
```

(continues on next page)

(continued from previous page)

```
[2021-11-03 19:38:43.274325 - 19:38:43.495096]  [ 42.33%]       l___f3(0.2208)
[2021-11-03 19:38:43.394686 - 19:38:43.495076]  [ 19.25%]          l___f2(0.1004)
[2021-11-03 19:38:43.495018 - 19:38:43.495043]  [  0.00%]             l___f1(0.
↪0000)

-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^ disable HiQ -^-^-^-^-^-^-^-^-^-^-^-^-^-^-
↪^-^-^-^-^-^
□ set global hiq to True
```

Note at line 16 above, we mocked `driver.get_tau_id`'s return value. In production or a more realistic setup, you don't have to do the mock, because `HiQLatency` will generate id for every instantiation automatically. The driver code will be like this:

```python
import os
import hiq
import traceback, sys
from hiq.hiq_utils import (
    HiQIdGenerator,
    HiQStatusContext,
)

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with HiQStatusContext():
        for i in range(3):
            with hiq.HiQLatency(hiq_table_or_path=f"{here}/hiq.conf") as driver:
                try:
                    hiq.mod("main").fit(data={}, model=[i])
                except Exception as e:
                    traceback.print_exc(file=sys.stdout)
                finally:
                    driver.show(show_key=True)


if __name__ == "__main__":
    run_main()
```

Tip: Using HiQLatency in a with statement is recommended, because this way you don't have to manually call driver.disable_hiq().

Run the code and the result is like:

```
□   python examples/multi-tracing/main_driver_real.py
□ k0: 16363948034575320, □ k1: time
[2021-11-08 18:06:43.809487 - 18:06:47.034452]  [100.00%] □_root_time(3.2250)
                                                            [OH:1309us]
[2021-11-08 18:06:43.809487 - 18:06:47.034452]  [100.00%]    l___f4(3.2250)
```

(continues on next page)

(continued from previous page)

```
[2021-11-08 18:06:44.028084 - 18:06:46.813921]  [ 86.38%]        |___f2(2.7858)
[2021-11-08 18:06:44.128282 - 18:06:46.813914]  [ 83.28%]        |  l___f1(2.6856)
[2021-11-08 18:06:44.128330 - 18:06:46.813906]  [ 83.27%]        |      l___f4(2.
↪6856)
[2021-11-08 18:06:44.328635 - 18:06:46.593395]  [ 70.23%]        |         |___f2(2.
↪2648)
[2021-11-08 18:06:44.428813 - 18:06:46.593385]  [ 67.12%]        |         |  l___
↪f1(2.1646)
[2021-11-08 18:06:44.428851 - 18:06:46.593379]  [ 67.12%]        |         |     l__
↪_f4(2.1645)
[2021-11-08 18:06:44.629144 - 18:06:46.372974]  [ 54.07%]        |         |        ␣
↪|___f2(1.7438)
[2021-11-08 18:06:44.729428 - 18:06:46.372967]  [ 50.96%]        |         |        ␣
↪|  l___f1(1.6435)
[2021-11-08 18:06:44.729487 - 18:06:46.372956]  [ 50.96%]        |         |        ␣
↪|     l___f4(1.6435)
[2021-11-08 18:06:44.972807 - 18:06:46.152537]  [ 36.58%]        |         |        ␣
↪|        |___f2(1.1797)
[2021-11-08 18:06:45.073035 - 18:06:46.152529]  [ 33.47%]        |         |        ␣
↪|        |  l___f1(1.0795)
[2021-11-08 18:06:45.073101 - 18:06:46.152517]  [ 33.47%]        |         |        ␣
↪|        |     l___f4(1.0794)
[2021-11-08 18:06:45.273425 - 18:06:45.931942]  [ 20.42%]        |         |        ␣
↪|        |        |___f2(0.6585)
[2021-11-08 18:06:45.373683 - 18:06:45.931931]  [ 17.31%]        |         |        ␣
↪|        |        |  l___f1(0.5582)
[2021-11-08 18:06:45.373748 - 18:06:45.931919]  [ 17.31%]        |         |        ␣
↪|        |        |     l___f4(0.5582)
[2021-11-08 18:06:45.574080 - 18:06:45.711033]  [  4.25%]        |         |        ␣
↪|        |        |        |___f2(0.1370)
[2021-11-08 18:06:45.674290 - 18:06:45.710935]  [  1.14%]        |         |        ␣
↪|        |        |        |  l___f1(0.0366)
[2021-11-08 18:06:45.711257 - 18:06:45.931909]  [  6.84%]        |         |        ␣
↪|        |        |        l___f3(0.2207)
[2021-11-08 18:06:45.831599 - 18:06:45.931898]  [  3.11%]        |         |        ␣
↪|        |        |           l___f2(0.1003)
[2021-11-08 18:06:45.931852 - 18:06:45.931873]  [  0.00%]        |         |        ␣
↪|        |        |              l___f1(0.0000)
[2021-11-08 18:06:45.931988 - 18:06:46.152507]  [  6.84%]        |         |        ␣
↪|        |        l___f3(0.2205)
[2021-11-08 18:06:46.052282 - 18:06:46.152497]  [  3.11%]        |         |        ␣
↪|        |           l___f2(0.1002)
[2021-11-08 18:06:46.152464 - 18:06:46.152475]  [  0.00%]        |         |        ␣
↪|        |              l___f1(0.0000)
[2021-11-08 18:06:46.152581 - 18:06:46.372947]  [  6.83%]        |         |        ␣
↪|        l___f3(0.2204)
[2021-11-08 18:06:46.272759 - 18:06:46.372938]  [  3.11%]        |         |        ␣
↪|           l___f2(0.1002)
[2021-11-08 18:06:46.372916 - 18:06:46.372924]  [  0.00%]        |         |        ␣
↪|              l___f1(0.0000)
[2021-11-08 18:06:46.373011 - 18:06:46.593367]  [  6.83%]        |         |        ␣
↪l___f3(0.2204)
```

(continues on next page)

(continued from previous page)

```
[2021-11-08 18:06:46.493184 - 18:06:46.593360]  [  3.11%]       |         |       ␣
↪   l___f2(0.1002)
[2021-11-08 18:06:46.593342 - 18:06:46.593349]  [  0.00%]       |         |       ␣
↪      l___f1(0.0000)
[2021-11-08 18:06:46.593431 - 18:06:46.813896]  [  6.84%]       |         l___f3(0.
↪2205)
[2021-11-08 18:06:46.713605 - 18:06:46.813889]  [  3.11%]       |            l___
↪f2(0.1003)
[2021-11-08 18:06:46.813858 - 18:06:46.813873]  [  0.00%]       |               l__
↪_f1(0.0000)
[2021-11-08 18:06:46.813971 - 18:06:47.034441]  [  6.84%]       l___f3(0.2205)
[2021-11-08 18:06:46.934184 - 18:06:47.034435]  [  3.11%]          l___f2(0.1003)
[2021-11-08 18:06:47.034407 - 18:06:47.034421]  [  0.00%]             l___f1(0.
↪0000)

 k0: 16363948070358521,  k1: time
[2021-11-08 18:06:47.387733 - 18:06:47.908781]  [100.00%] _root_time(0.5210)
                                                            [OH:229us]
[2021-11-08 18:06:47.387733 - 18:06:47.908781]  [100.00%]    l___f4(0.5210)
[2021-11-08 18:06:47.588017 - 18:06:47.688221]  [ 19.23%]       |___f2(0.1002)
[2021-11-08 18:06:47.688196 - 18:06:47.688206]  [  0.00%]       |   l___f1(0.0000)
[2021-11-08 18:06:47.688260 - 18:06:47.908773]  [ 42.32%]       l___f3(0.2205)
[2021-11-08 18:06:47.808428 - 18:06:47.908765]  [ 19.26%]          l___f2(0.1003)
[2021-11-08 18:06:47.908721 - 18:06:47.908746]  [  0.00%]             l___f1(0.
↪0000)

 k0: 16363948079093882,  k1: time
[2021-11-08 18:06:48.261303 - 18:06:48.782447]  [100.00%] _root_time(0.5211)
                                                            [OH:238us]
[2021-11-08 18:06:48.261303 - 18:06:48.782447]  [100.00%]    l___f4(0.5211)
[2021-11-08 18:06:48.461619 - 18:06:48.561838]  [ 19.23%]       |___f2(0.1002)
[2021-11-08 18:06:48.561810 - 18:06:48.561821]  [  0.00%]       |   l___f1(0.0000)
[2021-11-08 18:06:48.561881 - 18:06:48.782439]  [ 42.32%]       l___f3(0.2206)
[2021-11-08 18:06:48.682091 - 18:06:48.782432]  [ 19.25%]          l___f2(0.1003)
[2021-11-08 18:06:48.782395 - 18:06:48.782414]  [  0.00%]             l___f1(0.
↪0000)
```

Another way to replace the mock is to use:

```
driver.get_tau_id = HiQIdGenerator()
```

This will allow you to create only one `hiq.HiQLatency` object but will generate the same result as above.

# CHAPTER 4

# HIQ ADVANCED TOPICS

The metrics described in the previous chapter are enough for most of the use cases for system metrics. To gain more insights on business metrics, you need to customize HiQ.

## 4.1 Customized Tracing

HiQ is flexible so that you can customize it to trace other non-built-in metrics, such as business metrics. In order to customize it, you need to create your own class inheriting class `hiq.HiQSimple` and implement two functions `def custom(self)` and `def custom_disable(self)`.

### 4.1.1 Log Metrics and Information to stdio

The following is a code example to demo how to log information, including business metrics, into terminal. The target code is a call chain from `main()`-> `func1()` -> `func2()`. The arguments for the main function are two dictionaries: `model` and `data`. We know the data input has two keys `img_path` and `size`, and we want to log the values corresponding to the keys.

Target Code:

```python
import time


def func1(model: dict, data: dict) -> int:
    time.sleep(1.5)
    r2 = func2(model, data)
    return r2 * 2


```

(continues on next page)

(continued from previous page)

```python
def func2(model: dict, data: dict) -> int:
    time.sleep(2.5)
    return len(data["img_path"])


def main(model: dict, data: dict) -> int:
    r = func1(model, data)
    return r


if __name__ == "__main__":
    res = main(model={"data": "abc"}, data={"img_path": "/tmp/hiq.jpg", "size": 1024})
    print(res)
```

Driver Code:

```python
import os
import hiq
from inspect import currentframe as cf
from hiq.constants import *


class MyHiQ(hiq.HiQSimple):
    def custom(self):
        @self.inserter
        def __my_main(data={}, model={}) -> int:
            if "img_path" in data:
                print(f"□ print log for img_path: {data['img_path']}")
            if "img_size" in data:
                print(f"□ print log for img_size: {data['img_size']}")
            return self.o_main(data=data, model=model)

        self.o_main = hiq.mod("main").main
        hiq.mod("main").main = __my_main

    def custom_disable(self):
        hiq.mod("main").main = self.o_main


def run_main():
    with hiq.HiQStatusContext():
        _ = MyHiQ()
        hiq.mod("main").main(
            model={"data": "abc"}, data={"img_path": "/tmp/hello.jpg", "img_size": 1024}
        )


if __name__ == "__main__":
    run_main()
```

In the `custom()` function, we define a new function called `__my_main` which has the same signature of the target code's `main` function, and assign the target code's `main` to `self.o_main`, assign `__my_main` to the target code' s `main`.

Inside the `__my_main` function, we check if there is img_path in the data argument. If there is, we log it. Finally we call `self.o_main` and return the result.

Run the driver code and get the output:

```
□   python examples/custom/stdio/main_driver.py
□ print log for img_path: /tmp/hello.jpg
□ print log for img_size: 1024
```

Without touching the target code, we logged one line of message into standard io console. This is useful for debugging purposes. We can also trace the information in HiQ Tree.

### 4.1.2 Trace Metrics and Information In HiQ Tree

The target code will be the same as above. The difference here is we extract the information inside `__my_main` and define a function with decorator `@self._inserter_with_extra(extra={})`. `extra` will contain the information we want to trace. In this case, they are the image path and size.

Driver Code:

```python
import os
import hiq
from inspect import currentframe as cf
from hiq.constants import *


class MyHiQ(hiq.HiQSimple):
    def custom(self):
        def __my_main(data={}, model={}, *args, **kwargs) -> int:
            img_path = data["img_path"] if "img_path" in data else None
            img_size = data["img_size"] if "img_size" in data else None

            @self.inserter_with_extra(extra={"img": img_path, "size": img_size})
            def __z(data, model):
                return self.o_main(data=data, model=model)

            return __z(data, model)

        self.o_main = hiq.mod("main").main
        hiq.mod("main").main = __my_main

    def custom_disable(self):
        hiq.mod("main").main = self.o_main


def run_main():
    with hiq.HiQStatusContext():
        driver = MyHiQ()
```

(continues on next page)

(continued from previous page)

```
        hiq.mod("main").main(
            model={"data": "abc"},
            data={"img_path": "/tmp/hello.jpg", "from": "driver", "img_size":␣
↪1024},
        )
        driver.show()


if __name__ == "__main__":
    run_main()
```

Run the driver code and get the output:

```
□   python examples/custom/hiqtree/main_driver.py
[2021-11-05 05:05:39.910686 - 05:05:43.914784]  [100.00%] □_root_time(4.0041)
                                                            [{'img': '/tmp/hello.
↪jpg', 'size': 1024}]
                                                            [OH:104us]
[2021-11-05 05:05:39.910686 - 05:05:43.914784]  [100.00%]    l___z(4.0041)
```

Under the tree’ s root node, we can see the image path information and image size metric.

## 4.2 Log Monkey King

LMK is a separate high performance logging system of HiQ. Sometimes we don’ t need the structural information of the trace, we just need to log data into a file in the disk. In this case, we can use LMK. To use LMK, an environment variable `LMK` must be enabled.

### 4.2.1 Log Metrics and Information to stdio

Without extra setup, LMK will print out logging information in stdio.

Target Code:

```python
import time


def func1():
    time.sleep(1.5)
    func2()


def func2():
    time.sleep(2.5)


def main():
    func1()


if __name__ == "__main__":
    main()
```

Driver Code:

```python
1  import os
2  import hiq
3
4  here = os.path.dirname(os.path.realpath(__file__))
5
6
7  def run_main():
8      _ = hiq.HiQLatency(f"{here}/hiq.conf")
9      hiq.mod("main").main()
10
11
12 if __name__ == "__main__":
13     import time
14
15     os.environ["LMK"] = "1"
16     run_main()
17     time.sleep(2)
```

At line 15, we set `LMK` equals to `1`, which enables log monkey king. Run the code and we can get:

```
□   python examples/lmk/stdio/main_driver.py
2021-11-05 07:45:42.019567 - [time] [□ 2418220] □ [main]
2021-11-05 07:45:42.020127 - [time] [□ 2418220] □ [func1]
2021-11-05 07:45:43.521903 - [time] [□ 2418220] □ [func2]
2021-11-05 07:45:46.024517 - [time] [□ 2418220] □ [func2]
2021-11-05 07:45:46.024616 - [time] [□ 2418220] □ [func1]
2021-11-05 07:45:46.024635 - [time] [□ 2418220] □ [main]
```

The default log format is:

```
time_stamp - [metric name] [process id] monkey [function name] [extra information]
```

🐵 means function call is started, and 🙈 means function call is completed.

### 4.2.2 Log Metrics and Information to file

We can easily log the metrics and information into a file with LMK. LMK supports Python's built-in `logging` module and third party logging module like `loguru`.

#### 4.2.2.1 Python built-in `logging` module

Target Code:

```python
import time


def func1():
    time.sleep(1.5)
    func2()


def func2():
    time.sleep(2.5)


def main():
    func1()


if __name__ == "__main__":
    main()
```

Driver Code:

```python
import logging
import os

import hiq

```

(continues on next page)

(continued from previous page)

```python
6  here = os.path.dirname(os.path.realpath(__file__))
7
8
9  LOG_FORMAT = "%(levelname)s %(asctime)s - %(message)s"
10
11 logging.basicConfig(
12     filename="/tmp/lmk.log", filemode="w", format=LOG_FORMAT, level=logging.INFO
13 )
14
15 logger = logging.getLogger()
16
17
18 def run_main():
19     _ = hiq.HiQLatency(f"{here}/hiq.conf", lmk_logger=logger)
20     hiq.mod("main").main()
21
22
23 if __name__ == "__main__":
24     import time
25
26     os.environ["LMK"] = "1"
27     run_main()
28     time.sleep(2)
```

- Explanation

Line 9-15: set up logging format, log file path and name

Line 19: pass `logger` as `lmk_logger` when constructing HiQLatency Object

Run the driver code, then you can see the log has been written into file `/tmp/lmk.log`:

```
□   python examples/lmk/logging/main_driver.py
□   cat /tmp/lmk.log
INFO 2021-11-05 17:03:57,581 - 2021-11-05 17:03:57.580419 - [time] [□ 3568910] □ [main]
INFO 2021-11-05 17:03:57,581 - 2021-11-05 17:03:57.581022 - [time] [□ 3568910] □ [func1]
INFO 2021-11-05 17:03:59,083 - 2021-11-05 17:03:59.082735 - [time] [□ 3568910] □ [func2]
INFO 2021-11-05 17:04:01,585 - 2021-11-05 17:04:01.585346 - [time] [□ 3568910] □ [func2]
INFO 2021-11-05 17:04:01,585 - 2021-11-05 17:04:01.585472 - [time] [□ 3568910] □ [func1]
INFO 2021-11-05 17:04:01,585 - 2021-11-05 17:04:01.585492 - [time] [□ 3568910] □ [main]
```

#### 4.2.2.2 Third-party Logging Library Support

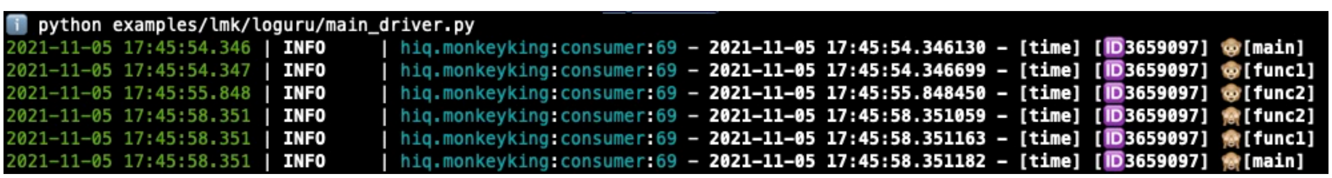
LMK supports third-party logging libraries which conforms to the standard logging protocol. One example is `loguru`. `loguru` is an easy-to-use, asynchronous, thread-safe, multiprocess-safe logging library. You can install it by running:

```
pip install loguru
```

The target code is the same as above. This is the driver Code:

```python
import os

import hiq
from loguru import logger

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    _ = hiq.HiQLatency(
        f"{here}/hiq.conf", lmk_logger=logger, lmk_path="/tmp/lmk_guru.log"
    )
    hiq.mod("main").main()


if __name__ == "__main__":
    import time

    os.environ["LMK"] = "1"
    run_main()
    time.sleep(2)
```

Run the driver code, you can see the information is printed in the terminal:




The same information is also stored in the log file:

```
□   cat /tmp/lmk_guru.log
2021-11-05 17:45:54.346 | INFO     | hiq.monkeyking:consumer:69 - 2021-11-05 17:45:54.346130 - [time] [□ 3659097] □ [main]
2021-11-05 17:45:54.347 | INFO     | hiq.monkeyking:consumer:69 - 2021-11-05 17:45:54.346699 - [time] [□ 3659097] □ [func1]
2021-11-05 17:45:55.848 | INFO     | hiq.monkeyking:consumer:69 - 2021-11-05 17:45:55.848450 - [time] [□ 3659097] □ [func2]
2021-11-05 17:45:58.351 | INFO     | hiq.monkeyking:consumer:69 - 2021-11-05 17:45:58.351059 - [time] [□ 3659097] □ [func2]
2021-11-05 17:45:58.351 | INFO     | hiq.monkeyking:consumer:69 - 2021-11-05 17:45:58.351163 - [time] [□ 3659097] □ [func1]
```

(continues on next page)

(continued from previous page)

```
2021-11-05 17:45:58.351 | INFO     | hiq.monkeyking:consumer:69 - 2021-11-05␣
↪17:45:58.351182 - [time] [ 3659097]  [main]
```

## 4.3 LumberJack

Different from LMK, which writes log entry for each span, LumberJack is to handle an entire HiQ tree. For simplicity, we call it Jack. Jack is very useful in use cases where the overhead for processing metrics is so big that you cannot process each entry one by one. Kafaka is one Exmaple. Due to message encoding, network latency and response validation, a call to a Kafaka producer's `send_message` can easily take more than 1 second. Jack is a good way to handle Kafka message. We can send metrics tree to Kafka and process it later with an analytics server. This will be described in details in section Integration with OCI Streaming.

Jack also writes a 500MB-rotated log in `~/.hiq/log_jack.log` unless you set environmental variable `NO_JACK_LOG`.

```
$ tail -n3 ~/.hiq/log_jack.log
time,v2,0,{"None":1637008247.9725869,1637008251.9771237,{"__main":1637008247.
↪9725869,1637008251.9771237,{"__func1":1637008247.972686,1637008251.9771047,{"__
↪func2":1637008249.4744177,1637008251.977021,}}}}
time,v2,0,{"None":1637008251.9785185,1637008255.9829764,{"__main":1637008251.
↪9785185,1637008255.9829764,{"__func1":1637008251.978641,1637008255.982966,{"__
↪func2":1637008253.480345,1637008255.9829247,}}}}
time,v2,0,{"None":1637008255.983492,1637008259.9854834,{"__main":1637008255.
↪983492,1637008259.9854834,{"__func1":1637008255.9836354,1637008259.9854727,{"__
↪func2":1637008257.485351,1637008259.9854305,}}}}
```

## 4.4 Async and Multiprocessing in Python

- TODO

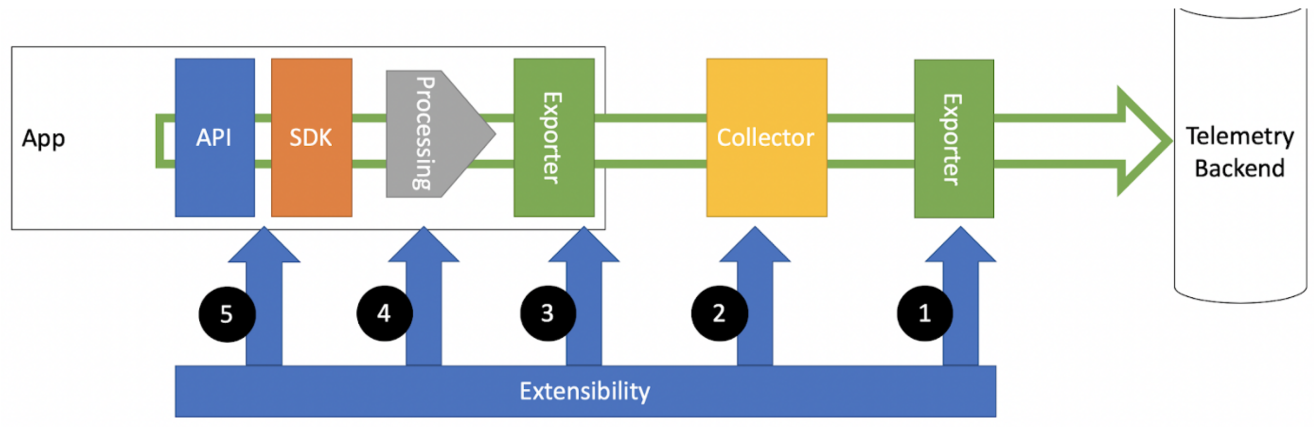
# CHAPTER 5

# HIQ DISTRIBUTED TRACING

Distributed tracing is the capability for a tracing solution to track and observe service requests as they flow through distributed systems by collecting data as the requests go from one service to another. The trace data helps you understand the flow of requests through your microservices environment and pinpoint where failures or performance issues are occurring in the system—and why.

## 5.1 OpenTelemetry

OpenTelemetry is a set of APIs, SDKs, tooling and integrations that are designed for the creation and management of telemetry data such as traces, metrics, and logs. It is vendor neutral, so it doesn’ t specify implementation details like Jaeger or Zipkin. OpenTelemetry provides default implementations for all the tracing backends and vendors, while allowing users to choose a different implementation for vendor specific features.




HiQ supports OpenTelemetry out of the box by context manager `HiQOpenTelemetryContext`.

To get OpenTelemetry and the code examples in this chapter working, install both the opentelemetry API and SDK:

```
pip install opentelemetry-api
pip install opentelemetry-sdk
```

The API package provides the interfaces required by the application owner, as well as some helper logic to load implementations. The SDK provides an implementation of those interfaces. The implementation is designed to be generic and extensible enough that in many situations, the SDK is sufficient. You won’ t use them directly but it is needed by HiQ.

## 5.2 Jaeger

Jaeger, inspired by Dapper and OpenZipkin, is a distributed tracing platform created by Uber Technologies and donated to Cloud Native Computing Foundation. It can be used for monitoring microservices-based distributed systems:

- Distributed context propagation
- Distributed transaction monitoring
- Root cause analysis
- Service dependency analysis
- Performance / latency optimization

https://www.jaegertracing.io/

HiQ supports Jaeger out of the box too.

### 5.2.1 Set Up

The following is an example which assume you have jaeger server/agent running locally. If you don't have, you can run the command to start a docker instance for jaeger server:

```
docker run --rm --name hiq_jaeger \
  -e COLLECTOR_ZIPKIN_HOST_PORT=:9411 \
  -p 5775:5775/udp \
  -p 6831:6831/udp \
  -p 6832:6832/udp \
  -p 5778:5778 \
  -p 16686:16686 \
  -p 14268:14268 \
  -p 14250:14250 \
  -p 9411:9411 \
  jaegertracing/all-in-one
```

The target code is the same as before:

```python
import time


def func1():
    time.sleep(1.5)
    print("func1")
    func2()


def func2():
    time.sleep(2.5)
    print("func2")


def main():
    func1()


if __name__ == "__main__":
    main()
```

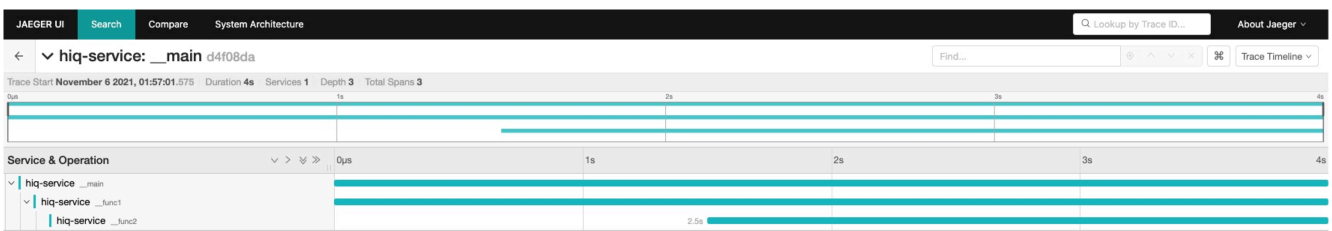
Jeager supports two protocols: `thrift` and `protobuf`.

### 5.2.2 Thrift + HiQ

Below is the driver code for `thrift`. You can see the only change is line 4 and 10. You only need to add a context manager `hiq.distributed.HiQOpenTelemetryContext` to get the jaeger tracing working.

```
1  import os
2  
3  import hiq
4  from hiq.distributed import HiQOpenTelemetryContext, OtmExporterType
5  
6  here = os.path.dirname(os.path.realpath(__file__))
7  
8  
9  def run_main():
10     with HiQOpenTelemetryContext(exporter_type=OtmExporterType.JAEGER_THRIFT):
11         driver = hiq.HiQLatency(f"{here}/hiq.conf")
12         hiq.mod("main").main()
13         driver.show()
14 
15 
16 if __name__ == "__main__":
17     run_main()
```

Run the driver code and check Jaeger UI at `http://localhost:16686`, you can see the traces have been recorded:

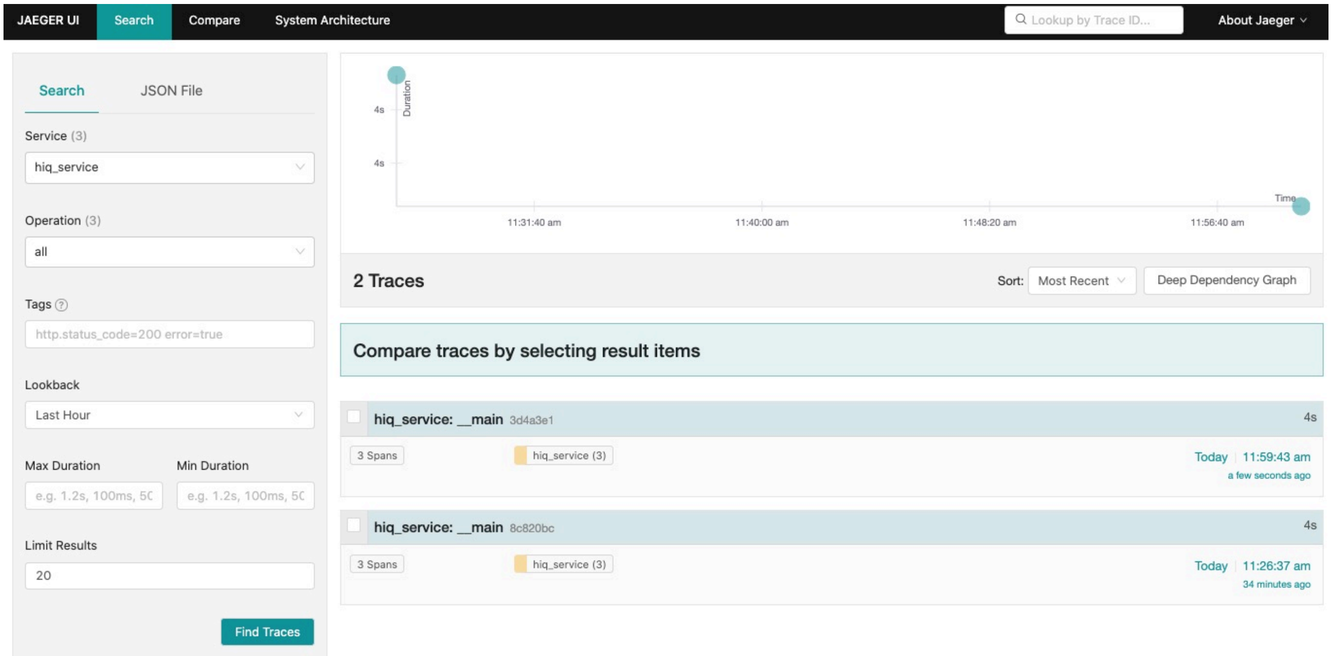



### 5.2.3 Protobuf + HiQ

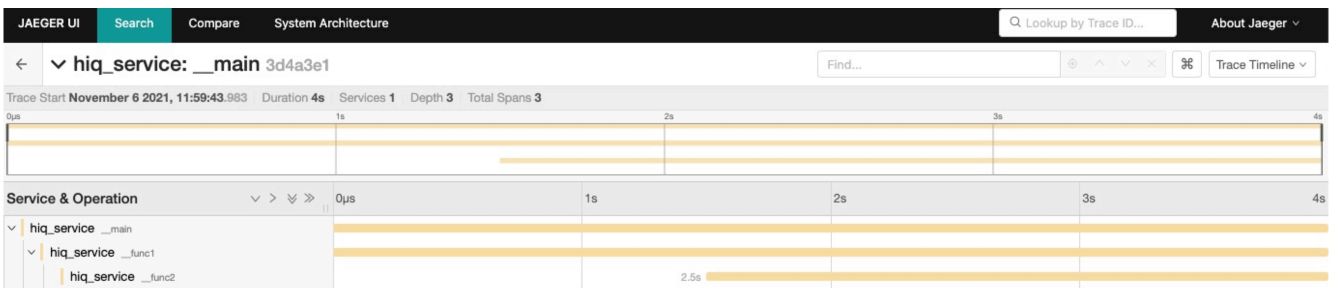
`Protobuf` works the same way. You just need to replace `OtmExporterType.JAEGER_THRIFT` with `OtmExporterType.JAEGER_PROTOBUF`. This exporter always sends traces to the configured agent using Protobuf via gRPC.

```
def run_main():
    with HiQOpenTelemetryContext(exporter_type=OtmExporterType.JAEGER_PROTOBUF):
        driver = hiq.HiQLatency(f"{here}/hiq.conf")
        hiq.mod("main").main()
        driver.show()
```

Run the driver code, and refresh Jaeger UI. We can see a new trace appears in Jaeger UI:




Click the new trace and we can see:




## 5.3 ZipKin

HiQ allows exporting of OpenTelemetry traces to Zipkin. This sends traces to the configured Zipkin collector endpoint using:

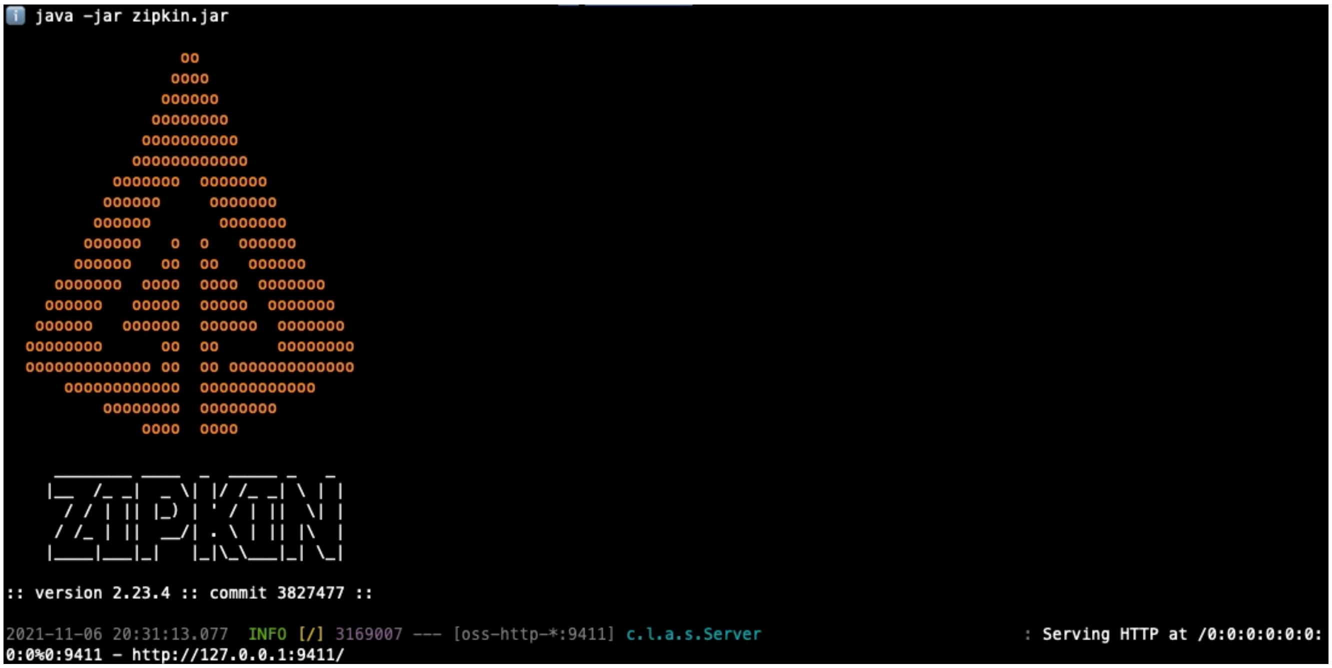
- JSON over HTTP with support of multiple versions (v1, v2)
- HTTP with support of v2 protobuf

### 5.3.1 Set Up

The quickest way to start a Zipkin server is to fetch the latest released server as a self-contained executable jar. Note that the Zipkin server requires minimum JRE 8. For example:

```
$ curl -sSL https://zipkin.io/quickstart.sh | bash -s
$ java -jar zipkin.jar
```

If everything is fine, you should see a Zipkin logo like:




Note: You can use the Jaeger server (port 9411) we launched too. But according to my test, it only works for JSON + HTTP mode, not Protobuf mode. However, the official Zipkin server works for both modes. Get the latest version at: https://github.com/openzipkin/zipkin.

The target code is the same as before.

### 5.3.2 JSON + HTTP + HiQ

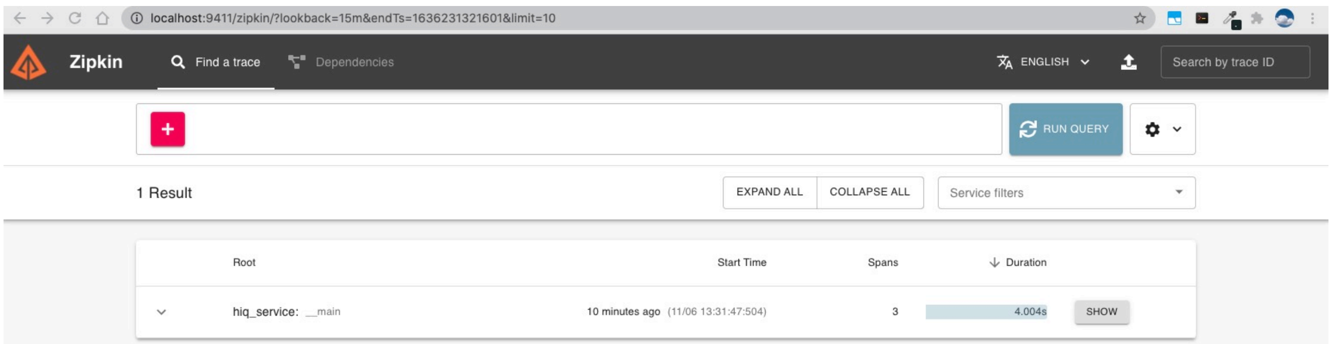
```python
import os

import hiq
from hiq.distributed import HiQOpenTelemetryContext, OtmExporterType

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
```

(continues on next page)

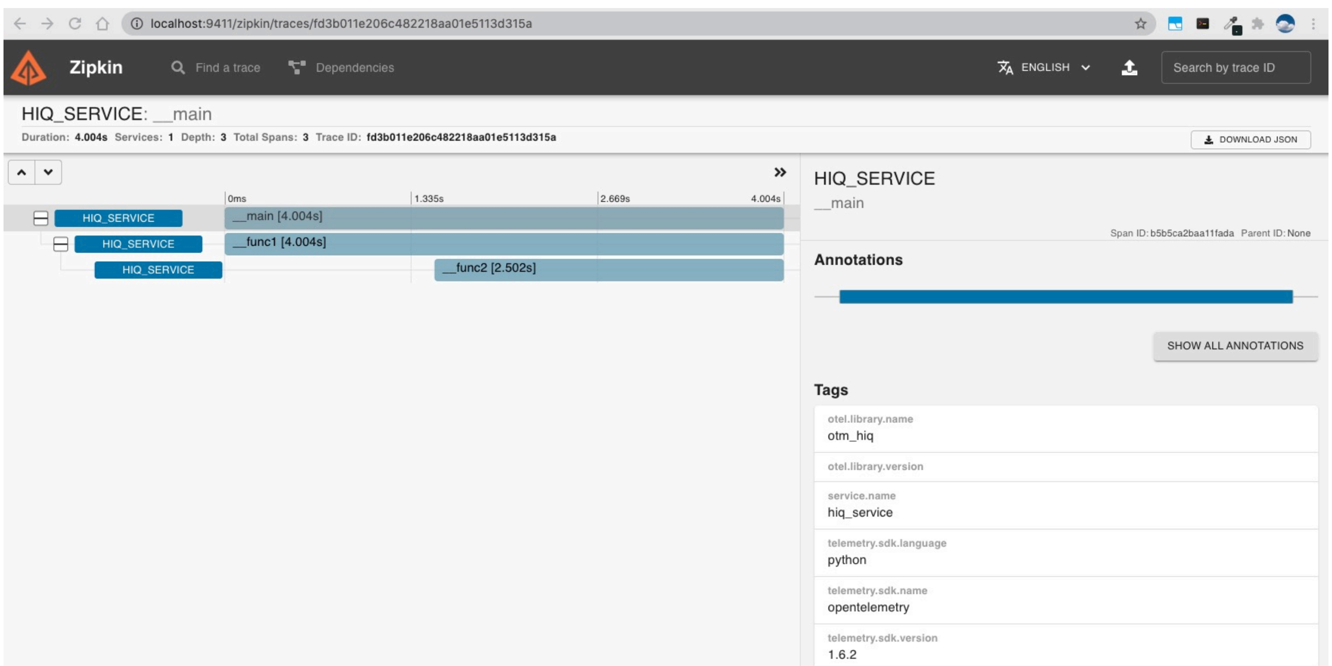
(continued from previous page)

```
    with HiQOpenTelemetryContext(exporter_type=OtmExporterType.ZIPKIN_JSON):
        driver = hiq.HiQLatency(f"{here}/hiq.conf")
        hiq.mod("main").main()
        driver.show()


if __name__ == "__main__":
    run_main()
```

Run the driver code and check the Zipkin web UI.




Click the `SHOW` button and we can see:

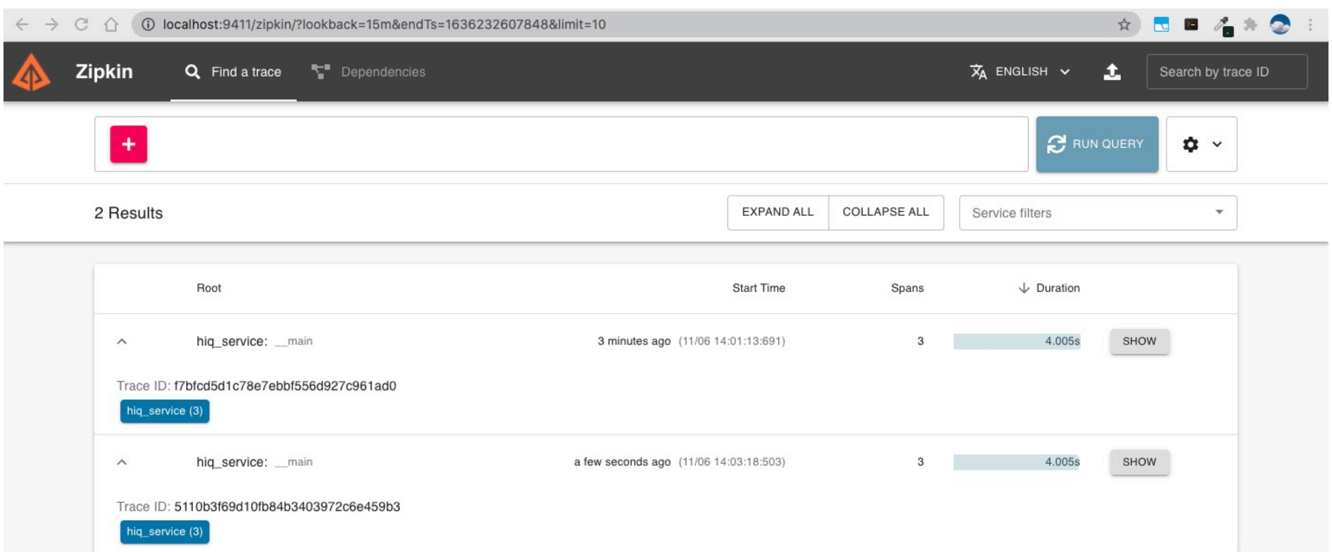



The default endpoint is `http://localhost:9411/api/v2/spans`. If there is a different endpoint `xxx`, you should add `endpoint='xxx'` as one of `HiQOpenTelemetryContext`' s arguments in the constructor.

### 5.3.3 Protobuf + HiQ

```python
import os

import hiq
from hiq.distributed import HiQOpenTelemetryContext, OtmExporterType

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with HiQOpenTelemetryContext(exporter_type=OtmExporterType.ZIPKIN_PROTOBUF):
        driver = hiq.HiQLatency(f"{here}/hiq.conf")
        hiq.mod("main").main()
        driver.show()


if __name__ == "__main__":
    run_main()
```

Run the driver code and check the Zipkin web UI. We can see a new trace has been recorded.




## 5.4 Ray

- Installation

```
pip install ray
```

## 5.5 Dask

- Installation

```
pip install dask
```

# CHAPTER 6

# HIQ VENDOR INTEGRATION

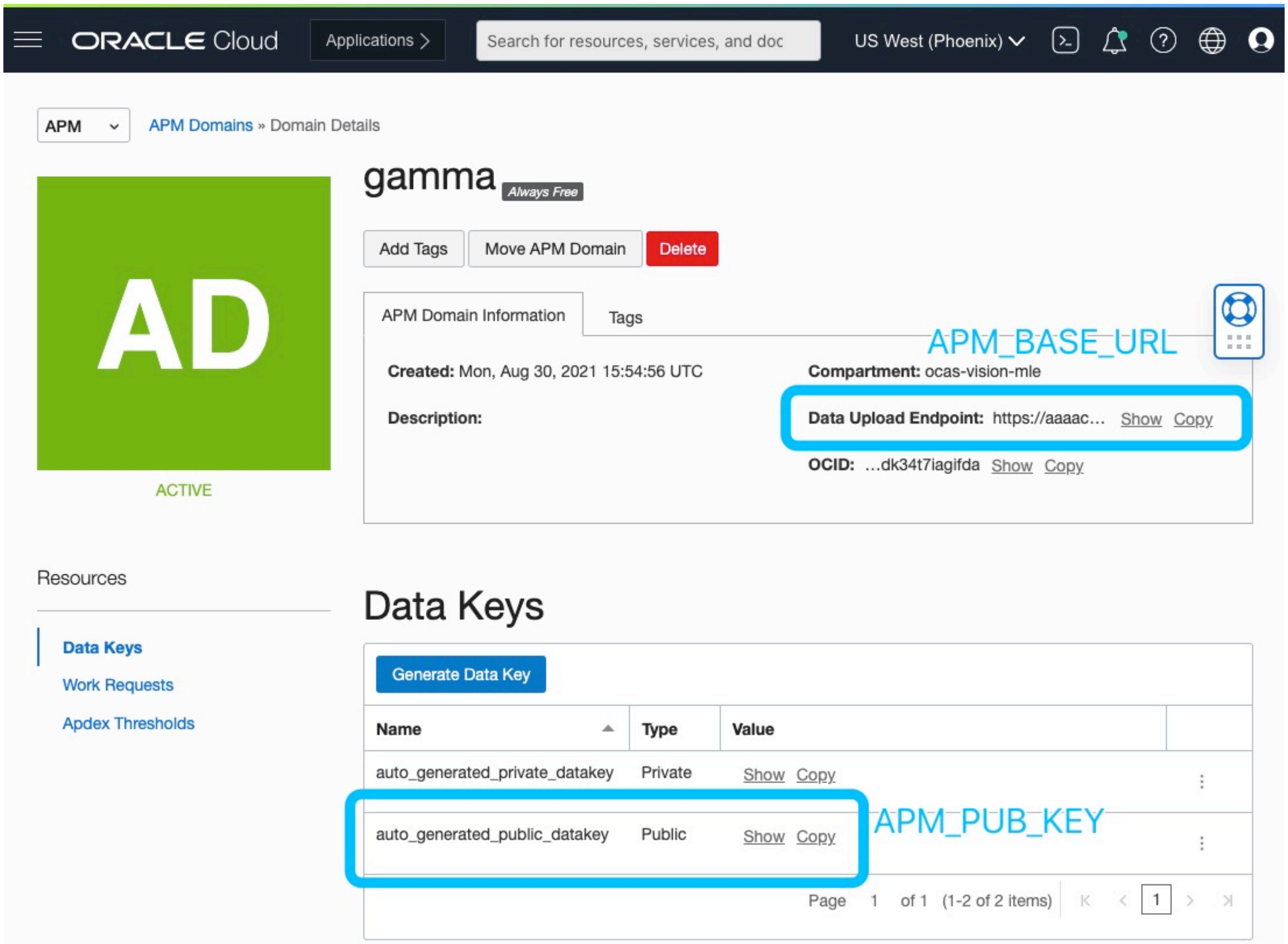
## 6.1 OCI APM

OCI Application Performance Monitoring (APM) is a service that provides deep visibility into the performance of applications and enables DevOps professionals to diagnose issues quickly in order to deliver a consistent level of service.

HiQ supports OCI APM out of the box.

### 6.1.1 Get APM Endpoint and Environments Setup

To use Oracle APM, we need to have the APM server's endpoint. To get the endpoint, you should copy your own `APM_BASE_URL` and `APM_PUB_KEY` from OCI web console and set them as environment variables.




`APM_BASE_URL` is the `Data Upload Endpoint` in `APM Domains` page; `APM_PUB_KEY` is the public key named `auto_generated_public_datakey` in the same page. You can just click the word `show` to copy them.

> Warning: The values below are fake and for demo purposes only. You should replace them with your own APM_BASE_URL and APM_PUB_KEY.

Then you can set them in the terminal like:

```
export APM_BASE_URL="https://aaaac64xyvkaiaaaxxxxxxxxxx.apm-agt.us-phoenix-1.oci.
↪oraclecloud.com"
export APM_PUB_KEY="JL6DVW2YBYYPA6G53UG3ZNAJSHSBSHSN"
```

> Tip: "The public key and public channel supposed to be used by something like a browser in which any end user may see the key. For server side instrumentation you should use the private data key. Changing this will make no difference in any way. The idea is that you may want/need to change the public key more often."

–Avi Huber

You can also set it in your python code programmatically with `os.environ` like what we have done in previous chapter.

There are two ways to use OCI APM in HiQ. The legacy way is to use `HiQOciApmContext` which uses `py_zipkin` under the hood. The modern way is to use `HiQOpenTelemetryContext`, which uses the new `OpenTelemetry` api.

### 6.1.2 HiQOciApmContext

The first way to send data to OCI APM is to use `HiQOciApmContext`. To use `HiQOciApmContext`, you need to install `py_zipkin`:

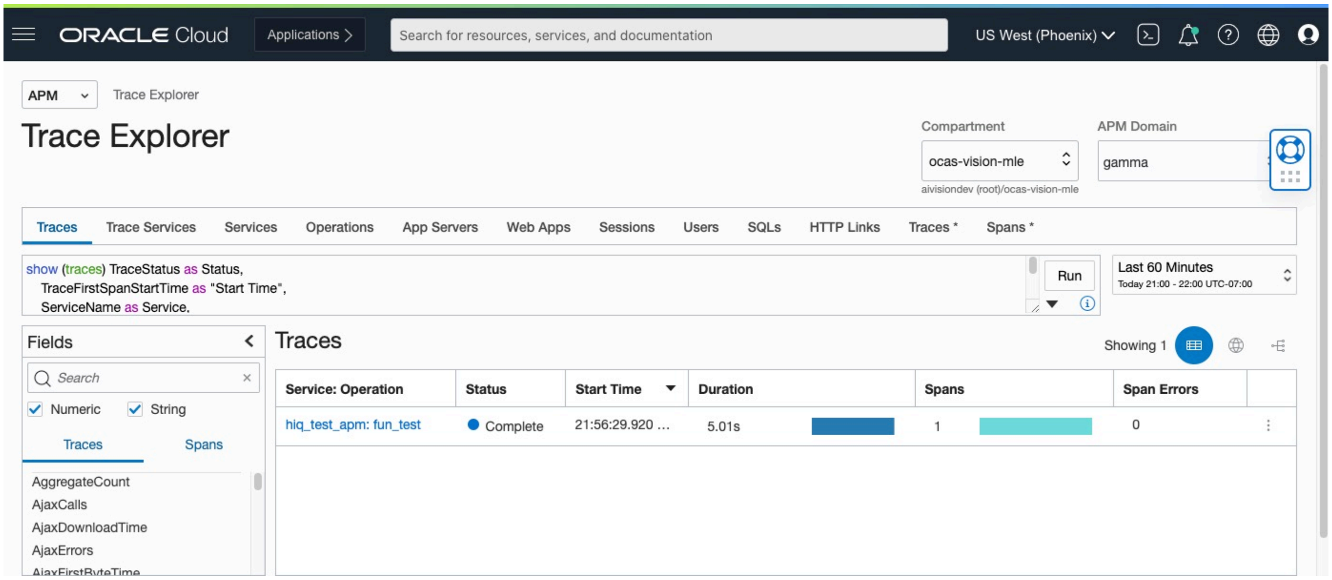
```
pip install py_zipkin
```

#### 6.1.2.1 A Quick Start Demo

With the two environment variables set, we can write the following code:

```python
import os
import time

from hiq.vendor_oci_apm import HiQOciApmContext


def fun():
    with HiQOciApmContext(
        service_name="hiq_test_apm",
        span_name="fun_test",
    ):
        time.sleep(5)
        print("hello")


if __name__ == "__main__":
    os.environ["TRACE_TYPE"] = "oci-apm"
    fun()
```

Run this code you can see the result in APM trace explorer.




### 6.1.2.2 Monolithic Application Performance Monitoring

Just like before, we have the same target code.

```python
import time


def func1():
    time.sleep(1.5)
    print("func1")
    func2()


def func2():
    time.sleep(2.5)
    print("func2")


def main():
    func1()


if __name__ == "__main__":
    main()
```

This is the driver code:

```python
import hiq
import os

from hiq.vendor_oci_apm import HiQOciApmContext

here = os.path.dirname(os.path.realpath(__file__))
```

(continues on next page)

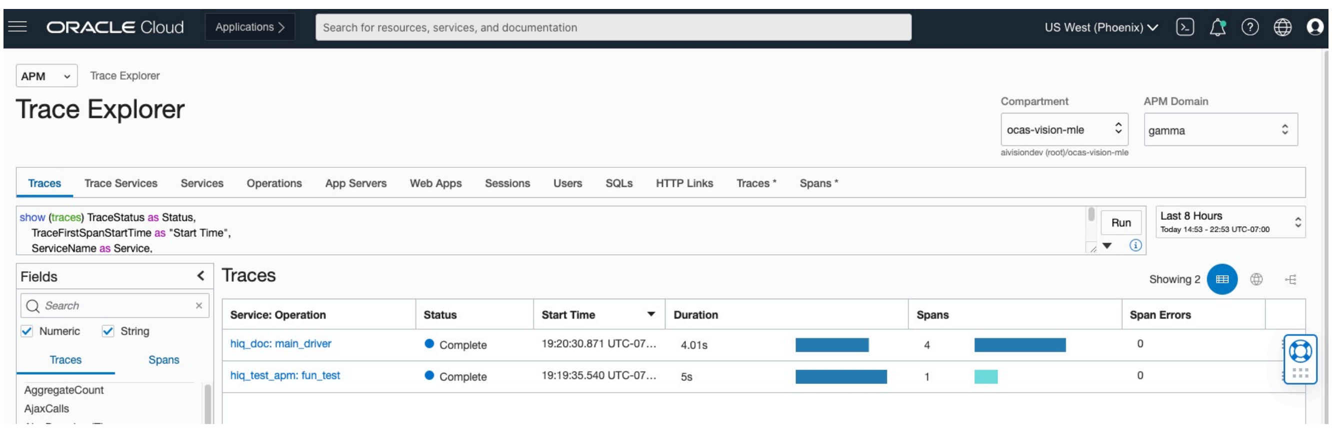
(continued from previous page)

```
7
8
9  def run_main():
10     with HiQOciApmContext(
11         service_name="hiq_doc",
12         span_name="main_driver",
13     ):
14         _ = hiq.HiQLatency(f"{here}/hiq.conf")
15         hiq.mod("main").main()
16
17
18 if __name__ == "__main__":
19     os.environ["TRACE_TYPE"] = "oci-apm"
20     run_main()
```

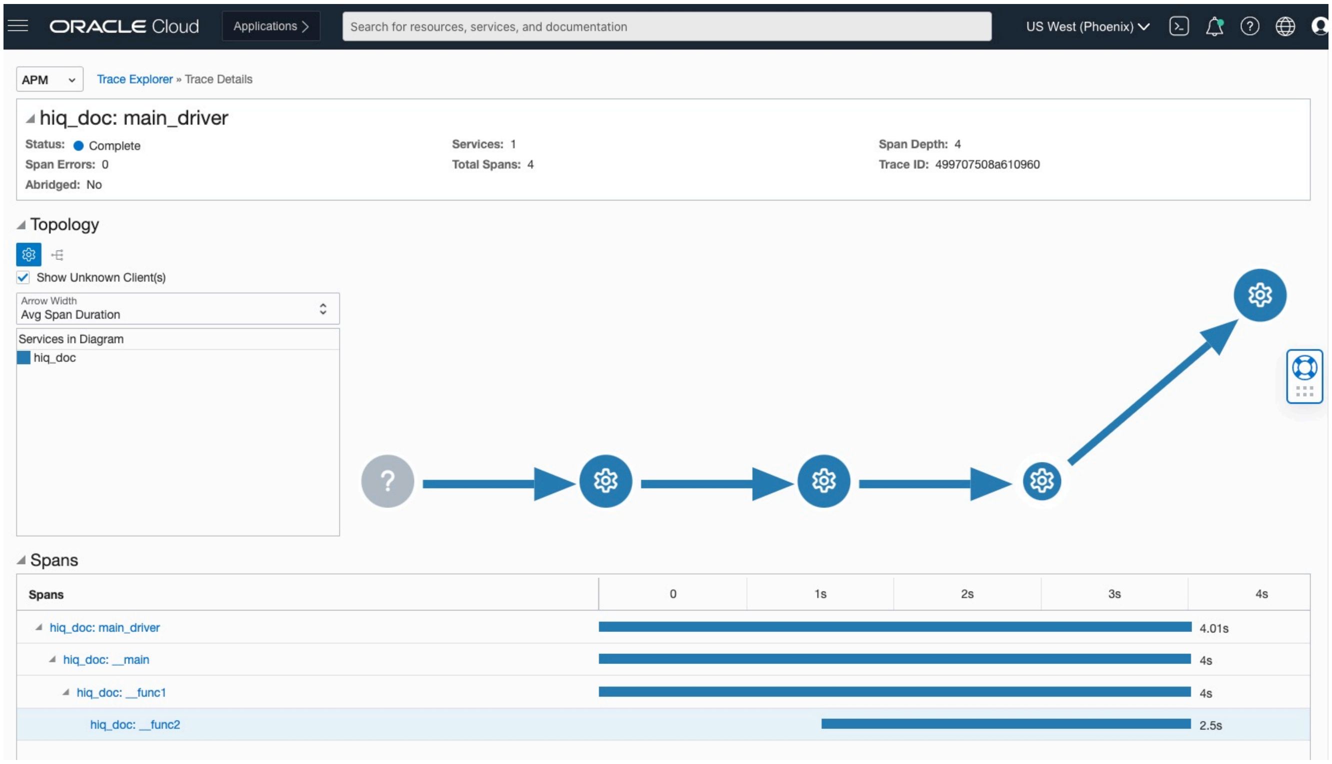
To view the performance in Oracle APM with HiQ, you just need to:

- Set environment variable `TRACE_TYPE` equal to `oci-apm` (Line 18)
- Create a `HiQOciApmContext` object using `with` clause and put everything under its scope (Line 10-12)

Run this code and check APM trace explorer in the web console.




We got a 4-span trace! Click `hiq_doc: main_driver` and we can see `Trace Details` page:




### 6.1.2.3 HiQ with Flask and OCI APM

HiQ can integrate with Flask and OCI APM by class `FlaskWithOciApm` in a non-intrusive way. This can be used in distributed tracing.

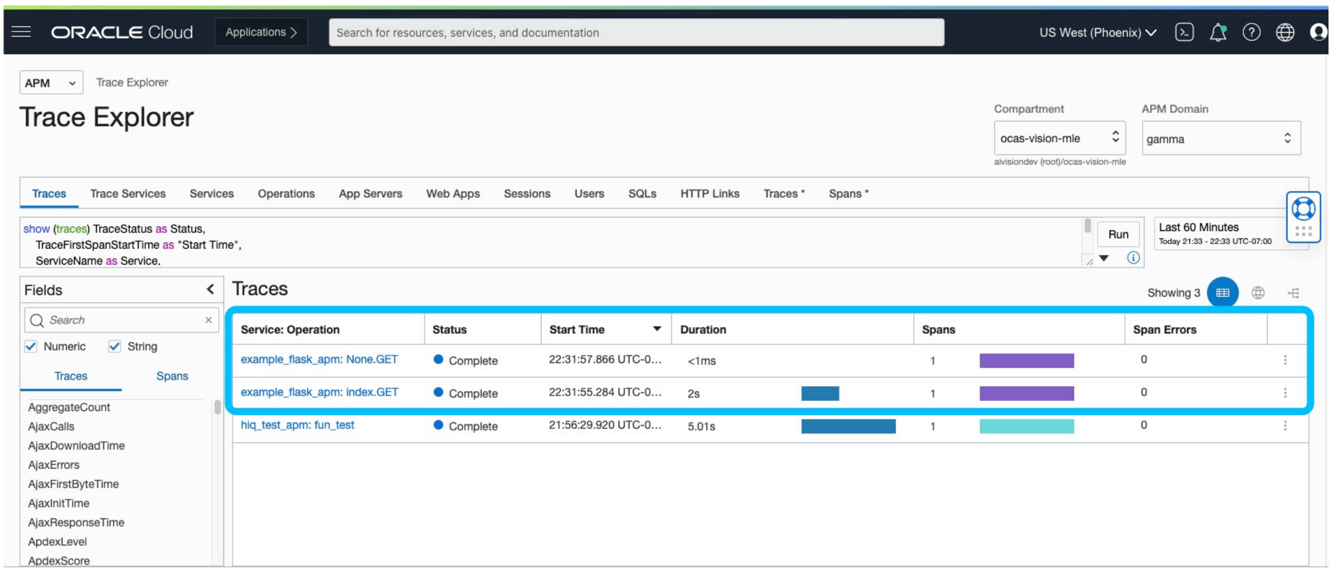
```python
import os
import time

from flask import Flask
from flask_request_id_header.middleware import RequestID
from hiq.server_flask_with_oci_apm import FlaskWithOciApm


def create_app():
    app = Flask(__name__)
    app.config["REQUEST_ID_UNIQUE_VALUE_PREFIX"] = "hiq-"
    RequestID(app)
    return app


app = create_app()

amp = FlaskWithOciApm()
amp.init_app(app)


@app.route("/", methods=["GET"])
```

(continues on next page)

(continued from previous page)

```python
def index():
    time.sleep(2)
    return "OK"


@app.route("/predict", methods=["GET"])
def predict():
    time.sleep(1)
    return "OK"


if __name__ == "__main__":
    host = "0.0.0.0"
    port = int(os.getenv("PORT", "8080"))
    debug = False
    app.run(host=host, port=port, debug=debug)
```

All the endpoints requests information will be recorded and available for analysis in APM.




### 6.1.3 HiQOpenTelemetryContext

The second way to send data to OCI APM is to use `HiQOpenTelemetryContext`, which leverage Open-Telemetry api under the hood.

For the same target code, the driver code is like:

```python
import hiq
import os

from hiq.distributed import HiQOpenTelemetryContext, OtmExporterType

here = os.path.dirname(os.path.realpath(__file__))
```

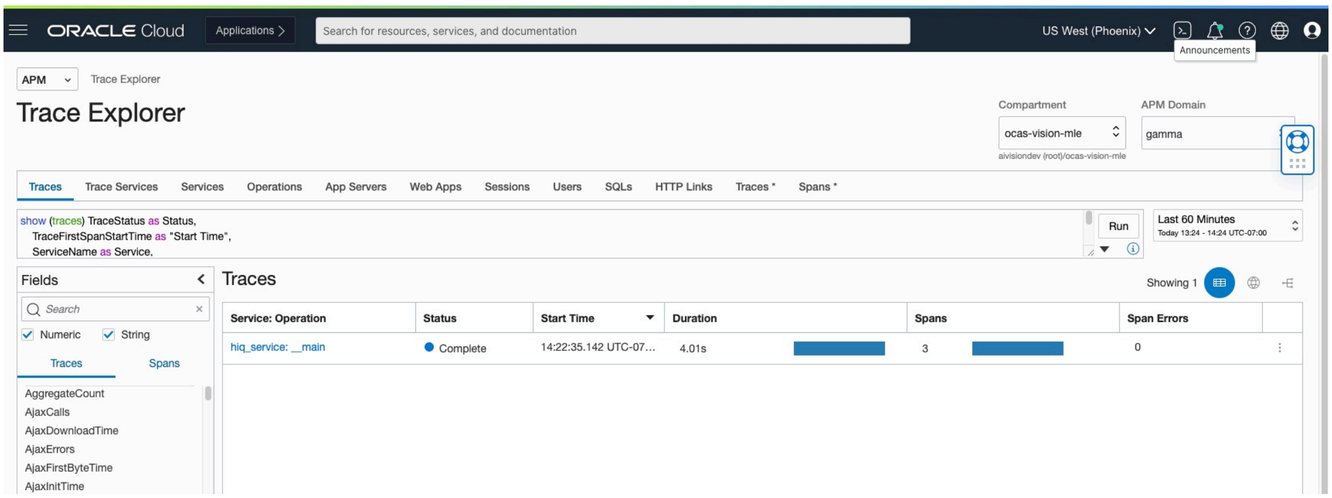
(continues on next page)

(continued from previous page)

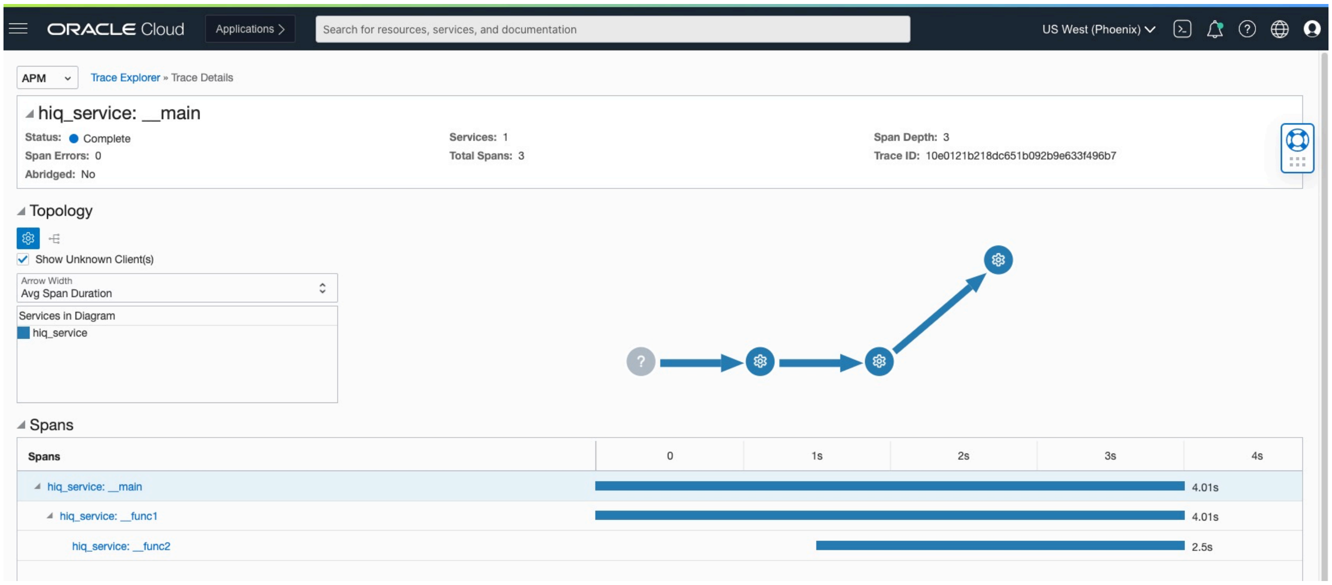
```
7
8
9  def run_main():
10     with HiQOpenTelemetryContext(exporter_type=OtmExporterType.ZIPKIN_JSON):
11         _ = hiq.HiQLatency(f"{here}/hiq.conf")
12         hiq.mod("main").main()
13
14
15 if __name__ == "__main__":
16     run_main()
```

Note: OCI APM doesn't support Protobuf metrics data for now. Only Json format data via HTTP is supported. So OtmExporterType.ZIPKIN_JSON is required in line 10 above.

Run the driver code and go to the OCI APM web console, we can see:




Click `hiq_service: __main`, we can see the trace details:




### 6.1.4 Reference

- OCI Application Performance Monitoring

## 6.2 OCI Functions

First you need to add `hiq` in the `requirements.txt`:

```
fdk>=0.1.39
hiq
```

We can easily send metrics data to APM inside an OCI function like below:

```python
import io
import json
import logging
import os

import hiq
from hiq.distributed import HiQOpenTelemetryContext, OtmExporterType
from fdk import response

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with HiQOpenTelemetryContext(exporter_type=OtmExporterType.ZIPKIN_JSON):
        _ = hiq.HiQLatency(f"{here}/hiq.conf")
        hiq.mod("main").main()

```

(continues on next page)

(continued from previous page)

```

def handler(ctx, data: io.BytesIO = None):
    name = "World"
    try:
        run_main()
        body = json.loads(data.getvalue())
        name = body.get("name")
    except (Exception, ValueError) as ex:
        logging.getLogger().info("error parsing json payload: " + str(ex))

    logging.getLogger().info("Inside Python Hello World function")
    return response.Response(
        ctx,
        response_data=json.dumps({"message": "Hello {0}".format(name)}),
        headers={"Content-Type": "application/json"},
    )
```

OCI Function is normally memory constrained. So you can use `HiQMemory` to replace `HiQLatency` above to get the memory consumption details.

## 6.3 OCI Telemetry(T2)

The Oracle Telemetry (T2) system provides REST APIs to help with gathering metrics, creating alarms, and sending notifications to monitor services built on the OCI platform. HiQ integrates with T2 seamlessly.

**class hiq.vendor_oci_t2.OciT2Client**(url, ad_longform=None, metrics_queue=None, trusted_cert=None, project='hiq', timeout=5)

OciT2Client is a class for transmitting metrics to Oracle T2

Examples:

```
from hiq.vendor_oci_t2 import OciT2Client as Client
from time import monotonic

client = Client(url=...)
METRIC_NAME = "hiq.predict"
metric = Client.metric_calc_delta(f"{METRIC_NAME}.success", start_
↪time=monotonic())
metrics_queue.put_nowait(metric)
#...
retry_count=1
client.gauge_metric("operation_retry_count", retry_count)
client.submit_metrics_queue()
```

**__init__**(url, ad_longform=None, metrics_queue=None, trusted_cert=None, project='hiq', timeout=5)

Constructor

Parameters

- **url** –The metrics data server URL
- **ad_longform** –the long-form name of the availability domain of the style like eu-frankfurt-ad-1
- **metrics_queue** –a Python queue.Queue object used for queuing metrics
- **trusted_cert** –The filename of the root certificate for authenticating the server, if needed

**timing_metric_data**(metric_data)

Send a list of metrics server

:param metric_data list of metrics

**wrap_metric_data**(metric_data) → dict

package up metrics in the data structure metrics server wants

Parameters **metric_data** –info[ 'metrics' ] => list of metrics

Returns info (metric dict/json)

**submit_metrics_queue**()

drain metrics_queue and submit the metrics to server

**gauge_metric**(metric, value)

emit a gauge metric with the given name and value.

**timing_metric**(metric: str, start_time: int)

convenience method to calculate a time delta for a metric and also put it in the metrics_queue

Parameters

- **metric** –metric name
- **start_time** –start time in millisecond unit

**get**(query, params=None)

method for mock metrics server API during testing, not applicable to prod

## 6.4 OCI Streaming

The OCI(Oracle Cloud Infrastructure) Streaming service provides a fully managed, scalable, and durable solution for ingesting and consuming high-volume data streams in real-time. Streaming is compatible with most Kafka APIs, allowing you to use applications written for Kafka to send messages to and receive messages from the Streaming service without having to rewrite your code. HiQ integrates with OCI streaming seamlessly.

To use OCI streaming you need to install oci python package first:

```
pip install oci
```

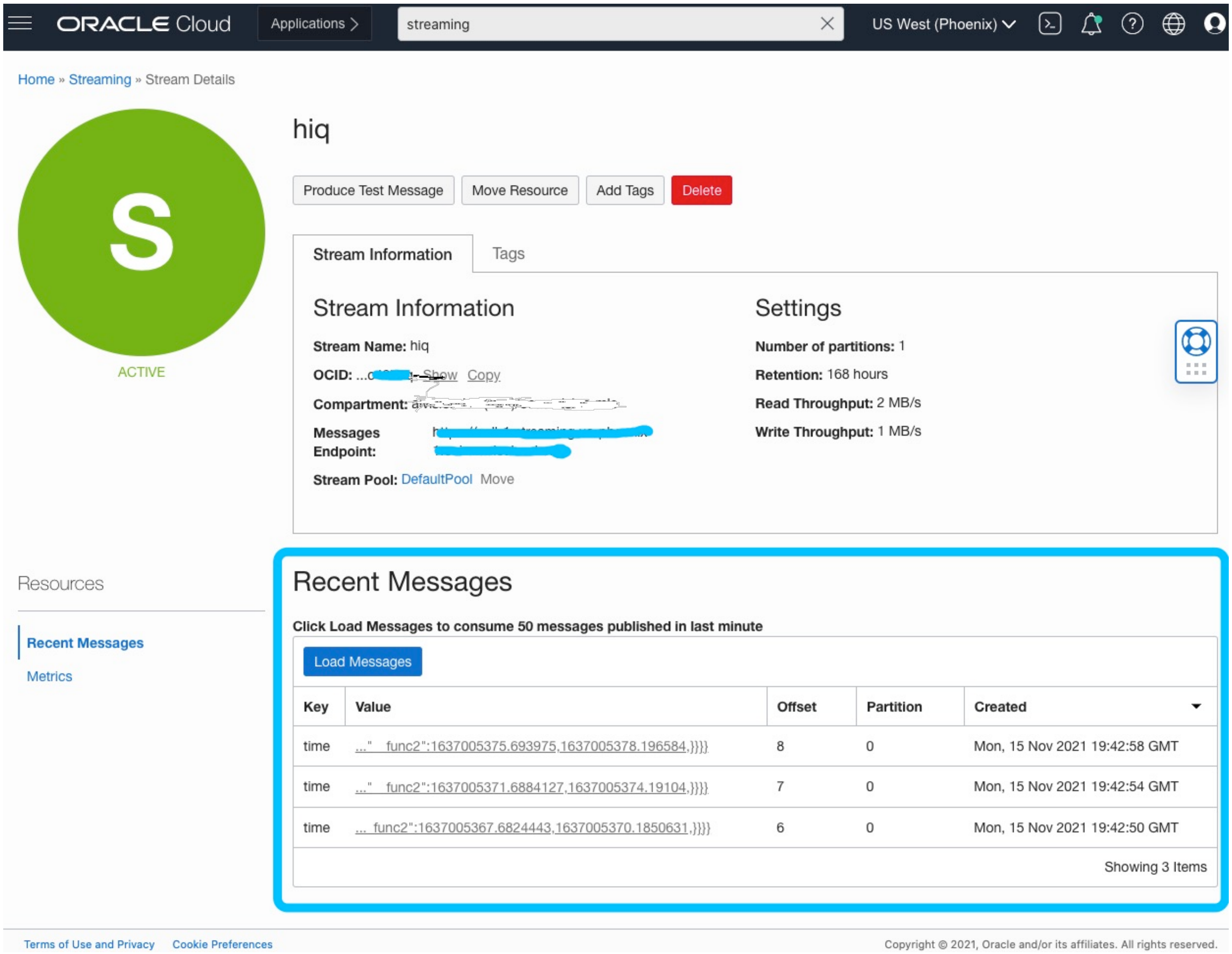
Then set up OCI streaming service and create a stream called `hiq` for instance. Please refer to OCI Streaming Document for how to set them up.

The target code is the same as before, and the following is the sample driver code:

```python
1  import os
2  import hiq
3  from hiq.hiq_utils import HiQIdGenerator
4
5  here = os.path.dirname(os.path.realpath(__file__))
6
7
8  def run_main():
9      with hiq.HiQStatusContext():
10         driver = hiq.HiQLatency(f"{here}/hiq.conf", max_hiq_size=0)
11         for _ in range(4):
12             driver.get_tau_id = HiQIdGenerator()
13             hiq.mod("main").main()
14             driver.show()
15
16
17 if __name__ == "__main__":
18     import time
19
20     os.environ["JACK"] = "1"
21     os.environ["HIQ_OCI_STREAMING"] = "1"
22     os.environ[
23         "OCI_STM_END"
24     ] = "https://cell-1.streaming.us-phoenix-1.oci.oraclecloud.com"
25     os.environ[
26         "OCI_STM_OCID"
27     ] = "ocid1.stream.oc1.phx.
   ↪amaaaaaa74akfsaawjmfsaeepurksns4oplsi5tobleyhfuxfqz24vc42k7q"
28
29     run_main()
30     time.sleep(2)
```

Due to the high latency of Kafka message sending, we process the metrics in the unit of HiQ tree in another process `Jack`. What you need to do is to set the environment variables `JACK` and `HIQ_OCI_STREAMING` to `1` like line 20 and 21, and also the streaming endpoint(`OCI_STM_END`) and streaming OCID(`OCI_STM_OCID`) with the information from your OCI web console.

Run the driver code and then go to OCI web console, you can see the HiQ trees have been recorded.




## 6.5 Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud, now a CNCF (Cloud Native Computing Foundation) project used by many companies and organizations. Prometheus collects and stores its metrics as time series data, i.e. metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels. If the targe code/service is a long running service, Prometheus is a good option for monitoring solution. HiQ provide an out-of-the-box solution for Prometheus.

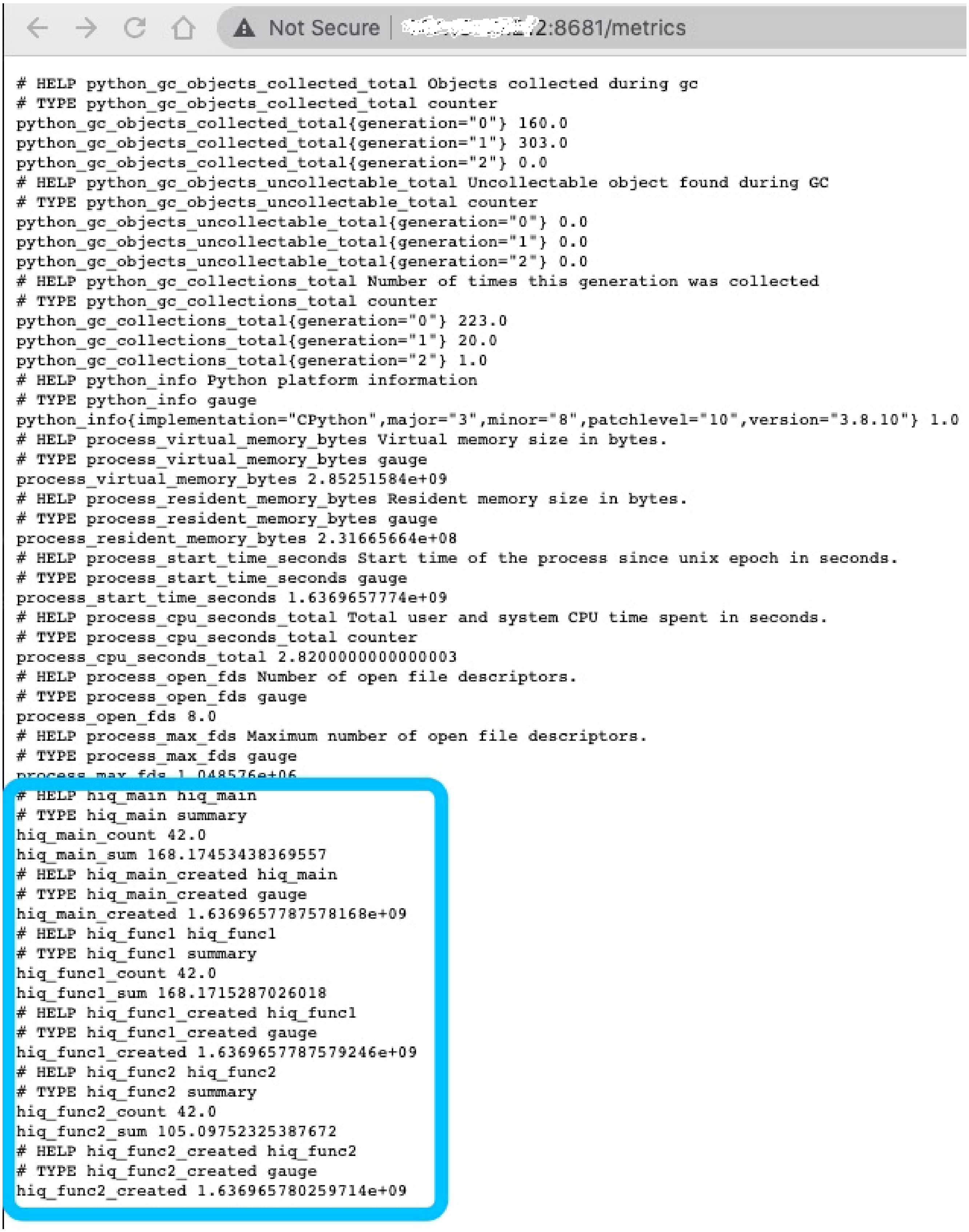
Like the other integration methods, you need to set environment variable `TRACE_TYPE`. To enable prometheus monitoring, you need to set it to `prometheus`.

Up to your performance SLA, you can call `start_http_server` from the main thread or, for better performance, you may want to use pushgateway but that involves more setup and operation overhead.

The following example shows how to expose Prometheus metrics with HiQ.

```python
import hiq
import os
import time
import random
from prometheus_client import start_http_server

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with hiq.HiQStatusContext():
        start_http_server(8681)
        count = 0
        while count < 10:
            with hiq.HiQLatency(f"{here}/hiq.conf") as driver:
                hiq.mod("main").main()
                driver.show()
            time.sleep(random.random())
            count += 1


if __name__ == "__main__":
    os.environ["TRACE_TYPE"] = "prometheus"
    run_main()
```

Run the driver code and visit `http://localhost:8681/metrics`, and we can see the metrics has been exposed. Please be noted that the metrics name has an `hiq_` as the prefix so that the metrics name is unique.




```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 160.0
python_gc_objects_collected_total{generation="1"} 303.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable object found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 223.0
python_gc_collections_total{generation="1"} 20.0
python_gc_collections_total{generation="2"} 1.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="8",patchlevel="10",version="3.8.10"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 2.85251584e+09
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.31665664e+08
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.6369657774e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 2.8200000000000003
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 8.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP hiq_main hiq_main
# TYPE hiq_main summary
hiq_main_count 42.0
hiq_main_sum 168.17453438369557
# HELP hiq_main_created hiq_main
# TYPE hiq_main_created gauge
hiq_main_created 1.6369657787578168e+09
# HELP hiq_func1 hiq_func1
# TYPE hiq_func1 summary
hiq_func1_count 42.0
hiq_func1_sum 168.1715287026018
# HELP hiq_func1_created hiq_func1
# TYPE hiq_func1_created gauge
hiq_func1_created 1.6369657787579246e+09
# HELP hiq_func2 hiq_func2
# TYPE hiq_func2 summary
hiq_func2_count 42.0
hiq_func2_sum 105.09752325387672
# HELP hiq_func2_created hiq_func2
# TYPE hiq_func2_created gauge
hiq_func2_created 1.636965780259714e+09
```

We can see the summary of `main`, `func1`, `func2` exposed. If the prometheus server is running in the same host, you can add the config in prometheus.yml to scrape the metrics for user to query.

```
- job_name: "hiq"
  static_configs:
    - targets: ["localhost:8681"]
```

# CHAPTER 7

# FAQ

## 7.1 HiQ vs cProfile

cProfile is a `built-in` python module that can perform profiling. It is the most commonly used profiler currently. It is non-intuitive and has wide support by third party modules.

We still use the same target code, and the driver code could be like this:

```
import cProfile
import hiq

with cProfile.Profile() as pr:
    hiq.mod("main").main()
    pr.dump_stats("result.pstat")
```

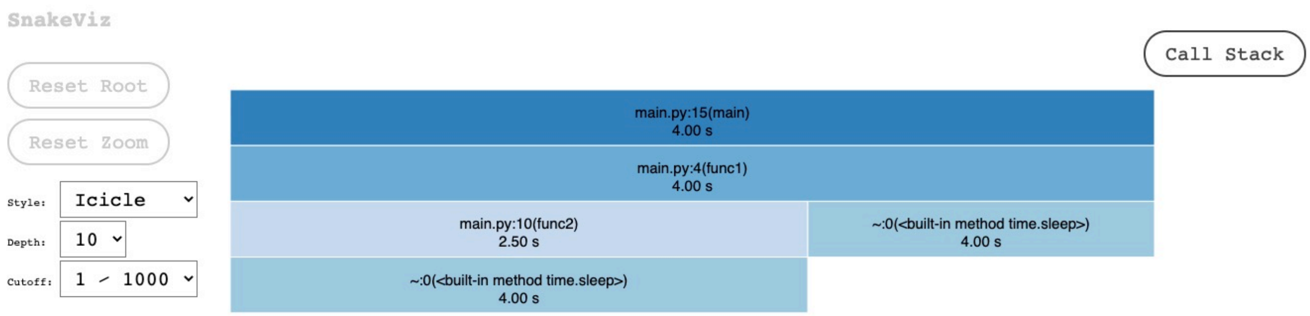
Running this will generate a stats file called `result.pstat`. We can use tools like `snakeviz` to analyze the result. SnakeViz is a browser based graphical viewer for the output of Python’s cProfile module and an alternative to using the standard library pstats module. SnakeViz is available on PyPI. Install with pip:

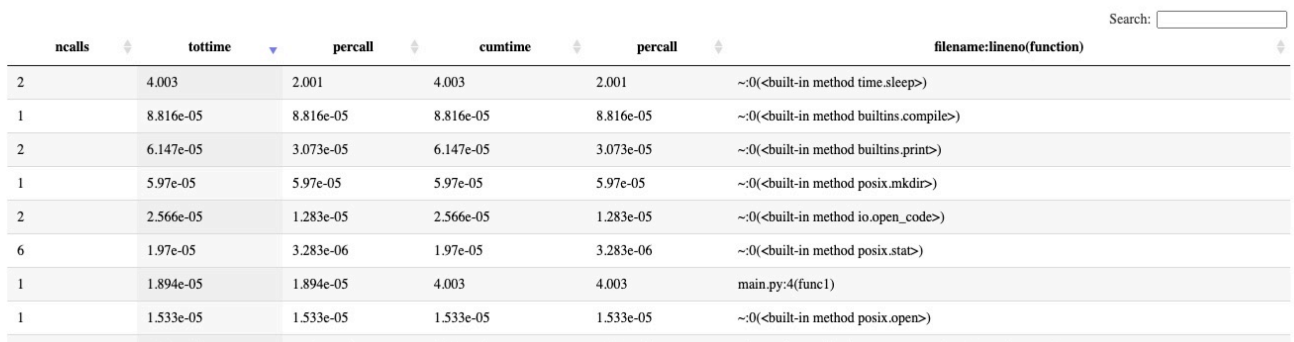
```
pip install snakeviz
```

Then simply run the command:

```
snakeviz result.pstat
```

A web browser will start and you can view the result like:



Search:

| ncalls | tottime | percall | cumtime | percall | filename:lineno(function) |
|---|---|---|---|---|---|
| 2 | 4.003 | 2.001 | 4.003 | 2.001 | ~:0(<built-in method time.sleep>) |
| 1 | 8.816e-05 | 8.816e-05 | 8.816e-05 | 8.816e-05 | ~:0(<built-in method builtins.compile>) |
| 2 | 6.147e-05 | 3.073e-05 | 6.147e-05 | 3.073e-05 | ~:0(<built-in method builtins.print>) |
| 1 | 5.97e-05 | 5.97e-05 | 5.97e-05 | 5.97e-05 | ~:0(<built-in method posix.mkdir>) |
| 2 | 2.566e-05 | 1.283e-05 | 2.566e-05 | 1.283e-05 | ~:0(<built-in method io.open_code>) |
| 6 | 1.97e-05 | 3.283e-06 | 1.97e-05 | 3.283e-06 | ~:0(<built-in method posix.stat>) |
| 1 | 1.894e-05 | 1.894e-05 | 4.003 | 4.003 | main.py:4(func1) |
| 1 | 1.533e-05 | 1.533e-05 | 1.533e-05 | 1.533e-05 | ~:0(<built-in method posix.open>) |

cProfile is based on c module `lsprof` (_lsprof.c) so it is very high performant in term of program execution. I even use cProfile to profile HiQ sometimes with small target code for development purpose.

However, it has many drawbacks:

- High Overhead: cProfile measures every single function call, so for program which has many function calls, it has high overhead and distorted results.
- Overwhelming Irrelevant Information: cProfile outputs too much information which is irrelevant to the real problem.
- Useful for Offline Development Only: Quite often your program will only be slow when run under real-world conditions, with real-world inputs. Maybe only particular queries from users slow down your web application, and you don't know which queries. Maybe your batch program is only slow with real data. But cProfile as we saw slows do your program quite a bit, and so you likely don't want to run it in your production environment. So while the slowness is only reproducible in production, cProfile only helps you in your development environment.
- Function Only and No Argument Information: cProfile can tell you "slowfunc() is slow", where it averages out all the inputs to that function. And that's fine if the function is always slow. But sometimes you have some algorithmic code that is only slow for specific inputs. cProfile will not be able to tell you which inputs caused the slowness, which can make it more difficult to diagnose the problem.

- Difficult to Customize: cProfile is designed to be a handy tool. You can write plugin with different cost functions, but that is not enough in many cases. It is not easy to customize.

HiQ, on the other hand, has low overhead and make it always transparent to users. It give users the option of which function to trace. With the zero span node filtered, the HiQ tree is even more concise and you can find the bottleneck at the first glance. It is fully customizable, fully dynamic. It is designed for production environment, so you can use HiQ in both production and development environment.

## 7.2 HiQ vs ZipKin vs Jaeger

HiQ can be used for both monolithic application and distributed tracing. HiQ can integrate with Zipkin and Jaeger and empower them with declarative, non-intrusive, dynamic and transparent distributed tracing.

## 7.3 HiQ vs GaalVM Insight

`GraalVM Insight` is able to trace information for all GraalVM languages (JavaScript, Python, Ruby, R) in a non-intrusive way with minimum overhead. However, it requires GraalVM installed, and it also suffers from compatability issue with third-party libraries like `numpy`.

# CHAPTER 8

# REFERENCE

- OpenTelemetry

# CHAPTER 9

# HIQ API

## 9.1 HiQ Classes

**class** hiq.base.**HiQBase**(hiq_table_or_path: typing.Union[str, typing.List[typing.Iterable[str]]] = [], metric_funcs: typing.List[typing.Callable] = [<built-in function time>], hiq_id_func: typing.Callable = <function get_tau_id>, func_args_handler: typing.Callable = <function func_args_handler>, target_path=None, max_hiq_size=30, verbose=False, fast_fail=True, tpl=None, extra_hiq_table: typing.List[str] = [], attach_timestamp=False, extra_metrics: typing.Set[hiq.constants.ExtraMetrics] = {}, lmk_path=None, lmk_handler=None, lmk_logger=None, *args, **kwargs)

**__init__**(hiq_table_or_path: typing.Union[str, typing.List[typing.Iterable[str]]] = [], metric_funcs: typing.List[typing.Callable] = [<built-in function time>], hiq_id_func: typing.Callable = <function get_tau_id>, func_args_handler: typing.Callable = <function func_args_handler>, target_path=None, max_hiq_size=30, verbose=False, fast_fail=True, tpl=None, extra_hiq_table: typing.List[str] = [], attach_timestamp=False, extra_metrics: typing.Set[hiq.constants.ExtraMetrics] = {}, lmk_path=None, lmk_handler=None, lmk_logger=None, *args, **kwargs)

constructor of ABC HiQBase

Parameters

- **hiq_table_or_path** (*Union[str, List[Iterable[str]], optional*) –this is an HiQ Conf, please refer to HiQ Core Concepts section in the HiQ documentation.Defaults to [].

- **metric_funcs** (*List[Callable], optional*) –simple metric function with empty argument. Defaults to [time.time].
- **hiq_id_func** (*Callable, optional*) –a callable to generate unique id for tau, the hiq map. Defaults to hiq.hiq_utils.get_tau_id.
- **func_args_handler** (*Callable, optional*) –a callable to convert function args/kwargs into a string. Defaults to hiq.hiq_utils.func_args_handler.
- **target_path** (*str, optional*) –the directory of the target code. Defaults to None.
- **max_hiq_size** (*int, optional*) –the max size of hiq map. if the number is exceeded, tree will be sent to LMK. Defaults to 30.
- **verbose** (*bool, optional*) –when verbose is true, more information will be recorded, like the full stack trace of exception will be recorded in HiQ tree node. Defaults to False.
- **fast_fail** (*bool, optional*) –when it is true, raise exception to the upper level, don't swallow exceptions. Defaults to True.
- **tpl** (*str, optional*) –hiq tpl path. Defaults to None.
- **extra_hiq_table** (*List[str], optional*) –a list of string to decide to include metrics other than latency. Defaults to [].
- **attach_timestamp** (*bool, optional*) –for non-time/non-latency metric tree, should we attach start, end timestamps in the extra dictionary? Defaults to False.
- **extra_metrics** (*set, optional*) –metrics to track in extra field of HiQ node. The value could be EXTRA_METRIC_ARGS, EXTRA_METRIC_FILE or EXTRA_METRIC_FUNC and it is normally used in development environment. Defaults to set().

**enable_hiq**(reset_trace=False)

Enable HiQ tracing

Parameters

- **s** (*HiQBase*) –self object of HiQBase
- **reset_trace** (*bool, optional*) –will the s.tau data structure will be reset to empty? Defaults to False.

Returns the current object itself

Return type HiQBase

**disable_hiq**(reset_trace=False)

Disable HiQ tracing

Parameters

- **s** (*HiQBase*) –self object of HiQBase
- **reset_trace** (*bool, optional*) –will the s.tau data structure will be reset to empty? Defaults to False.

**get_overhead**(format_=OverHeadFormat.ABS) → float

get tracing latency overhead in absolute format or percentage format

Parameters **format_** (*str, optional*) – "abs" - output absolute value in milliseconds; otherwise, output in float point format: overhead/total_latency. Defaults to "abs".

Returns absolute value in micro-second, or a number between 0 and 1 which means the percentage of overhead over total latency.

Return type float

**get_overhead_us()** → float

get tracing latency overhead in unit of micro-second

How to calculate latency overhead? The latency overhead is attached to the latency tree. the latency overhead calculation is based on monotonic time and has unit of micro-second. When a HiQ system is instantiated, the initial overhead is 0. Every time a target function is called, we accumulate the overhead. When we finish a level-2 node in the HiQ tree, we update the overhead in the tree to get the final overhead of that trace.

Returns absolute value in micro-second

Return type float

**get_overhead_pct()** → float

get tracing latency overhead in percentage format

Returns a number between 0 and 1 which means the percentage of overhead over total latency.

Return type float

**abstract custom()**

The abstract method for customizing tracing logic

**abstract custom_disable()**

The abstract method for disabling customized tracing logic

**set_extra_metrics**(extra_metrics: Iterable[hiq.constants.ExtraMetrics])

set extra metric information so that these information will enter the span node

**class hiq.base.HiQLatency**(hiq_table_or_path: typing.Union[str, list] = [], metric_funcs: typing.List[typing.Callable] = [<built-in function time>], hiq_id_func: typing.Callable = <function get_tau_id>, func_args_handler: typing.Callable = <function func_args_handler>, target_path=None, max_hiq_size=30, verbose=False, fast_fail=True, tpl=None, extra_hiq_table=[], attach_timestamp=False, extra_metrics={}, lmk_path=None, lmk_handler=None, lmk_logger=None)

A convenient class for latency tracking

Parameters

- **hiq_table_or_path** (*Union[str, List[Iterable[str]]], optional*) –this is an HiQ Conf, please refer to HiQ Core Concepts section in the HiQ documentation.Defaults to [].
- **metric_funcs** (*List[Callable], optional*) –simple metric function with empty argument. Defaults to [time.time].
- **hiq_id_func** (*Callable, optional*) –a callable to generate unique id for tau, the hiq map. Defaults to hiq.hiq_utils.get_tau_id.

- **func_args_handler** (*Callable, optional*) –a callable to convert function args/kwargs into a string. Defaults to hiq.hiq_utils.func_args_handler.
- **target_path** (*str, optional*) –the directory of the target code. Defaults to None.
- **max_hiq_size** (*int, optional*) –the max size of hiq map. if the number is exceeded, tree will be sent to LMK. Defaults to 30.
- **verbose** (*bool, optional*) –when verbose is true, more information will be recorded, like the full stack trace of exception will be recorded in HiQ tree node. Defaults to False.
- **fast_fail** (*bool, optional*) –when it is true, raise exception to the upper level, don't swallow exceptions. Defaults to True.
- **tpl** (*str, optional*) –hiq tpl path. Defaults to None.
- **extra_hiq_table** (*List[str], optional*) –a list of string to decide to include metrics other than latency. Defaults to [].
- **attach_timestamp** (*bool, optional*) –for non-time/non-latency metric tree, should we attach start, end timestamps in the extra dictionary? Defaults to False.
- **extra_metrics** (*set, optional*) –metrics to track in extra field of HiQ node. The value could be EXTRA_METRIC_ARGS, EXTRA_METRIC_FILE or EXTRA_METRIC_FUNC and it is normally used in development environment. Defaults to set().

Raises **ValueError** –Requires the input is valid

Example usage:

```
>>> from hiq.base import HiQLatency
>>> trace = HiQLatency()
```

**class** hiq.base.**HiQMemory**(hiq_table_or_path: typing.Union[str, list] = [], metric_funcs: typing.List[typing.Callable] = [<built-in function time>, <function get_memory_mb>], hiq_id_func: typing.Callable = <function get_tau_id>, func_args_handler: typing.Callable = <function func_args_handler>, target_path=None, max_hiq_size=30, verbose=False, fast_fail=True, tpl=None, extra_hiq_table=[], attach_timestamp=False, extra_metrics={}, lmk_path=None, lmk_handler=None, lmk_logger=None)

A convenient class for RSS(resident set size) of memory tracking. Unit of memory is MB.

RSS is the portion of memory occupied by a process that is held in main memory (RAM). The rest of the occupied memory exists in the swap space or file system, either because some parts of the occupied memory were paged out, or because some parts of the executable were never loaded.

Parameters

- **hiq_table_or_path** (*Union[str, List[Iterable[str]]], optional*) –this is an HiQ Conf, please refer to HiQ Core Concepts section

in the HiQ documentation.Defaults to [].
- **metric_funcs** (*List[Callable], optional*) –simple metric function with empty argument. Defaults to [time.time, get_memory_mb].
- **hiq_id_func** (*Callable, optional*) –a callable to generate unique id for tau, the hiq map. Defaults to hiq.hiq_utils.get_tau_id.
- **func_args_handler** (*Callable, optional*) –a callable to convert function args/kwargs into a string. Defaults to hiq.hiq_utils.func_args_handler.
- **target_path** (*str, optional*) –the directory of the target code. Defaults to None.
- **max_hiq_size** (*int, optional*) –the max size of hiq map. if the number is exceeded, tree will be sent to LMK. Defaults to 30.
- **verbose** (*bool, optional*) –when verbose is true, more information will be recorded, like the full stack trace of exception will be recorded in HiQ tree node. Defaults to False.
- **fast_fail** (*bool, optional*) –when it is true, raise exception to the upper level, don't swallow exceptions. Defaults to True.
- **tpl** (*str, optional*) –hiq tpl path. Defaults to None.
- **extra_hiq_table** (*List[str], optional*) –a list of string to decide to include metrics other than latency. Defaults to [].
- **attach_timestamp** (*bool, optional*) –for non-time/non-latency metric tree, should we attach start, end timestamps in the extra dictionary? Defaults to False.
- **extra_metrics** (*set, optional*) –metrics to track in extra field of HiQ node. The value could be EXTRA_METRIC_ARGS, EXTRA_METRIC_FILE or EXTRA_METRIC_FUNC and it is normally used in development environment. Defaults to set().

Raises **ValueError** –Requires the input is valid

Example usage:

```
>>> from hiq.base import HiQMemory
>>> trace = HiQMemory()
```

hiq.base.**HiQSimple**

alias of *hiq.base.HiQLatency*

## 9.2 Integration Classes

**class** hiq.vendor_oci_apm.**OciApmHttpTransport**(encoded_span, apm_url=None, debug=False)

a zipkin transport_handler class to emit traces to Oracle APM

To make it work, you need to set two environment variables: APM_BASE_URL, APM_PUB_KEY

Example:

```
import time

from py_zipkin import Encoding
from py_zipkin.zipkin import zipkin_span
from hiq.vendor_oci_apm import OciApmHttpTransport


def fun():
    with zipkin_span(
        service_name="hiq_test_apm",
        span_name="fun_test",
        transport_handler=OciApmHttpTransport,
        encoding=Encoding.V2_JSON,
        binary_annotations={"mode": "sync"},
        sample_rate=100,
    ):
        time.sleep(5)
        print("hello")


if __name__ == "__main__":
    fun()
```

**class** hiq.vendor_oci_apm.**HiQOciApmContext**(service_name='hiq_service_name', span_name='hiq_span_name', binary_annotations={'mode': 'sync'}, sample_rate=100)

Logs a root zipkin span with HiQ support

Example:

```
import time

from hiq.vendor_oci_apm import HiQOciApmContext


def fun():
    with HiQOciApmContext(
        service_name="hiq_test_apm", span_name="fun_test",
    ):
        time.sleep(5)
        print("hello")


if __name__ == "__main__":
    fun()
```

**__init__**(service_name='hiq_service_name', span_name='hiq_span_name', binary_annotations={'mode': 'sync'}, sample_rate=100)

Constructor of HiQOciApmContext class

Parameters

- **service_name** (*str, optional*) –The name of the called service. Defaults to "hiq_service_name".
- **span_name** (*str, optional*) –root span name. Defaults to "hiq_span_name".
- **binary_annotations** (*dict, optional*) –dict of str -> str span attrs. Defaults to { "mode" : "sync" }.
- **sample_rate** (*int, optional*) –Rate at which to sample; 0.0 - 100.0. If passed-in zipkin_attrs have is_sampled=False and the sample_rate param is > 0, a new span will be generated at this rate. This means that if you propagate sampling decisions to downstream services, but still have sample_rate > 0 in those services, the actual rate of generated spans for those services will be > sampling_rate. Defaults to 100.

**class** hiq.server_flask_with_oci_apm.**FlaskWithOciApm**(app=None, sample_rate=100, timeout=1, transport_handler_=<class 'hiq.vendor_oci_apm.OciApmHttpTran

A Flask App's helper class with Oci-Apm supported by default

To make it work, you need to set two environment variables: APM_BASE_URL, APM_PUB_KEY

Example:

```
import os
import time

from flask import Flask
from flask_request_id_header.middleware import RequestID
from hiq.server_flask_with_oci_apm import FlaskWithOciApm


def create_app():
    app = Flask(__name__)
    app.config["REQUEST_ID_UNIQUE_VALUE_PREFIX"] = "hiq-"
    RequestID(app)
    return app


app = create_app()
amp = FlaskWithOciApm()
amp.init_app(app)


@app.route("/", methods=["GET"])
def index():
    return "OK"


if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    host = "0.0.0.0"
    port = int(os.getenv("PORT", "8080"))
    debug = False
    app.run(host=host, port=port, debug=debug)
```

## 9.3 Distributed Tracing

**hiq.distributed.setup_jaeger**(service_name='hiq_service', agent_host_name='localhost', agent_port=6831, tracer_name='otm_hiq', collector_endpoint='localhost:14250', exporter_type: hiq.distributed.OtmExporterType = OtmExporterType.JAEGER_THRIFT)

Set up jaeger client

Parameters

- **service_name** (*str, optional*) –the name of the service. Defaults to "hiq_service" .
- **agent_host_name** (*str, optional*) –jaeger agent host name. Defaults to "localhost" .
- **agent_port** (*int, optional*) –jaeger agent port. Defaults to 6831.
- **tracer_name** (*str, optional*) –tracer name. Defaults to "otm_hiq" .

**hiq.distributed.HiQOpenTelemetryContext**(exporter_type: hiq.distributed.OtmExporterType = OtmExporterType.JAEGER_THRIFT, *args, **kwargs)

OpenTelemetry Context Manager with HiQ

Example:

```
import os

import hiq
from hiq.distributed import HiQOpenTelemetryContext, OtmExporterType

here = os.path.dirname(os.path.realpath(__file__))


def run_main():
    with HiQOpenTelemetryContext(
        exporter_type=OtmExporterType.JAEGER_PROTOBUF,
        collector_endpoint="localhost:14250"
    ):
        driver = hiq.HiQLatency(f"{here}/hiq.conf")
```

(continues on next page)

(continued from previous page)

```
        hiq.mod("main").main()
        driver.show()


if __name__ == "__main__":
    run_main()
```

## 9.4 Metrics Client

**class** hiq.http_metric_client.**HttpMetricsClient**(url, ad_longform=None, metrics_queue=None, trusted_cert=None, project='hiq', timeout=5)

HttpMetricsClient is a generic class for transmitting metrics to any metrics server by HTTP

This includes the config needed to create and submit metrics, along with the requests.Session we use to send the data.

Examples:

```
from hiq.http_metric_client import HttpMetricsClient as Client
from time import monotonic

client = Client(url=...)
METRIC_NAME = "hiq.predict"
metric = Client.metric_calc_delta(f"{METRIC_NAME}.success", start_
↪time=monotonic())
metrics_queue.put_nowait(metric)
#...
retry_count=1
client.gauge_metric("operation_retry_count", retry_count)
client.submit_metrics_queue()
```

**__init__**(url, ad_longform=None, metrics_queue=None, trusted_cert=None, project='hiq', timeout=5)

Constructor

Parameters

- **url** –The metrics data server URL
- **ad_longform** –the long-form name of the availability domain of the style like eu-frankfurt-ad-1
- **metrics_queue** –a Python queue.Queue object used for queuing metrics
- **trusted_cert** –The filename of the root certificate for authenticating the server, if needed

**timing_metric_data**(metric_data)

Send a list of metrics server

:param metric_data list of metrics

**wrap_metric_data**(metric_data) → dict

package up metrics in the data structure metrics server wants

Parameters **metric_data** –info[ 'metrics' ] => list of metrics

Returns info (metric dict/json)

**submit_metrics_queue**()

drain metrics_queue and submit the metrics to server

**gauge_metric**(metric, value)

emit a gauge metric with the given name and value.

**timing_metric**(metric: str, start_time: int)

convenience method to calculate a time delta for a metric and also put it in the metrics_queue

Parameters

- **metric** –metric name
- **start_time** –start time in millisecond unit

**get**(query, params=None)

method for mock metrics server API during testing, not applicable to prod

## 9.5 Utility Functions

hiq.**get_global_hiq_status**(name='hiq', default=True) → bool

Get the global HiQ status. True means HiQ is enabled, and False disabled.

When HIQ_STATUS_CACHED is set as True, the global hiq status will be cached for 5 seconds. Otherwise, it will read shared memory.

```
>>> import hiq
>>> hiq.get_global_hiq_status()
False
>>> hiq.set_global_hiq_status(1)
 set global hiq to 1
>>> hiq.get_global_hiq_status()
True
```

hiq.**set_global_hiq_status**(on=True, name='hiq', debug=False)

Set the global HiQ status. True is to enable HiQ, and False disable.

```
>>> import hiq
>>> hiq.get_global_hiq_status()
True
>>> hiq.set_global_hiq_status(0)
 set global hiq to 0
>>> hiq.get_global_hiq_status()
False
```

**hiq.mod**(module_name: str = '') → object

load python module with a string-type module_name without exception(None will be returned in case of exception)

**class hiq.HiQStatusContext**(target_status_on=True, debug=False)

An HiQ context manager

Inside HiQStatusContext, HiQ status is always enabled unless the status is changed in other processes(HiQ status is not guarded).

```
>>> from hiq import HiQStatusContext
>>> with HiQStatusContext():
        # HiQ will be enabled inside the `with` block, and reverted␣
↪to original value out of the block
>>> with HiQStatusContext(target_status_on=False):
        # HiQ will be disabled inside the `with` block, and reverted␣
↪to original value out of the block
```

**__init__**(target_status_on=True, debug=False)

Constructor of HiQStatusContext

Parameters

- **target_status_on** (*bool, optional*) –set the target HiQ status you want to set in the context manager. Defaults to True.
- **debug** (*bool, optional*) –print more information when debug is True. Defaults to False.

> Warning: hiq.HiQStatusContext is not multi-thread and multi-processing safe.

**class hiq.SingletonMeta**

**class hiq.SingletonBase**(*args, **kwargs)

**class hiq.SilencePrint**

**hiq.read_file**(file_path: str, binary_mode: bool = False, by_line: bool = True, filter_func: Optional[Callable] = None, as_json=False, bytes_as_string=True, raise_=False, strip=False) → Union[List[str], bytes]

A handy function to read file

Parameters

- **file_path** (*str*) –file path
- **raise** (*bool*) –populate exception when it happens, for instance, when it' s False, in case the file doesn' t exist, return None
- **strip** (*bool*) –strip the string output or not?

Don' t use this for super large file.

Examples:

```
>>> data = read_file(f"{here}/o.json")
>>> pp(data)
['{',
'    "eventKey": "repo:refs_changed",',
'    "date": "2021-09-21T04:52:51+0000",',
'    "actor": {',
'        "name": "fuhengwu",',
'        "emailAddress": "fuheng.wu@oracle.com",',
'        "id": 937189,',
'        "displayName": "fuheng wu",',
'        "active": true,',
'        "slug": "fuhengwu",',
'        "type": "NORMAL",',
'        "links": {',
'            "self": [{ "href": '
'"https://www.github.com/users/fuhengwu" }]',
'        }',
'    }',
'}']
>>> data = read_file(f"{here}/o.json", binary_mode=True)
>>> pp(data)
(b'{\n    "eventKey": "repo:refs_changed",\n    "date": "2021-09-
↪21T04:52:51+'
b'0000",\n    "actor": {\n        "name": "fuhengwu",\n
↪"emailAddress":'
b' "fuheng.wu@oracle.com",\n        "id": 937189,\n
↪"displayName": "F'
b'uheng Wu",\n        "active": true,\n        "slug": "fuhengwu",\n␣
↪       "t'
b'ype": "NORMAL",\n        "links": {\n            "self": [{ "href
↪": "https'
b'://www.github.com/users/fuhengwu" }]\n        }\n    }\n}')
>>> data = read_file(f"{here}/o.json", as_json=True)
>>> pp(data)
{'actor': {'active': True,
        'displayName': 'fuheng wu',
        'emailAddress': 'fuheng.wu@oracle.com',
        'id': 937189,
        'links': {'self': [{'href': 'https://www.github.com/users/
↪fuhengwu'}]},
        'name': 'fuhengwu',
        'slug': 'fuhengwu',
        'type': 'NORMAL'},
'date': '2021-09-21T04:52:51+0000',
'eventKey': 'repo:refs_changed'}
```

`hiq.write_file(file_path, data, as_owner=None, as_group=None, append=False, mod_='644')`

write data into file. if file_path is just a file name, the file will be created in current directory

`hiq.execute_cmd(command: str, split=True, verbose=True, check=False, shell=False, timeout=600, stderr_log=None, debug=False, keep_delim=False) → Union[str, List[str]]`

If verbose is true, print out input. If check is true, and the process exits with a non-zero exit code, a CalledProcessError exception will be raised. Attributes of that exception hold the arguments, the exit code, and stdout and stderr if they were captured. stderr_log could be: stderr_log=open( "/tmp/gamma.error.log" , "a+" ) https://docs.python.org/3/library/subprocess.html#subprocess.run

`hiq.download_from_http(uri, local_file_path, display=False, enable_proxy=True, auth=None) → str`

auth=(user, password)

`hiq.ensure_folder(path_str: str)`

ensure there is a folder for the path_str, the input is supposed to be a FILE path

`hiq.get_env_bool(x, default=None) → bool`

Get bool type environment variable

These values will be treated as True:

non-empty string, non-zero numeric

```
>>> import os
>>> from hiq.utils import get_env_bool
>>> get_env_bool('hello')
False
>>> os.environ["hello"]=""
>>> get_env_bool('hello')
False
>>> os.environ["hello"]="1"
>>> get_env_bool('hello')
True
>>> os.environ["hello"]="0"
>>> get_env_bool('hello')
False
>>> os.environ["hello"]="true"
>>> get_env_bool('hello')
True
```

`hiq.get_env_int(x, default=0) → int`

`hiq.get_env_float(x, default=0) → float`

`hiq.lmk_data_handler(data: dict = {}, pid=974524) → str`

convert raw data of Log Monkey King to the format we want to log

Parameters

- **data** (*dict, optional*) –a data dictionary. If not empty, the keys should be: id_, name, value, extra, is_start. Defaults to {}.
- **pid** (*[type], optional*) –LMK process id. Defaults to os.getpid().

Returns a log entry as a data string

Return type str

`hiq.get_home()`

`hiq.get_proxies()` → dict

`hiq.random_str(length_of_string=12)`

`hiq.memoize(func)`

`hiq.memoize_first(func)`

`hiq.is_hiqed(fun: Callable, fun_name: str)` → bool

check if this function has been registered in HiQ system

The function could be one of:

full_qualified_name=" <method 'read' of '_io.TextIOWrapper' objects>" , fun_name=' read'

full_qualified_name=' <function main at 0x7f7f2eaf6040>' , fun_name=' main'

full_qualified_name=' <built-in function read>' , fun_name=' read'

Parameters

- **fun** (*Callable*) –original function
- **fun_name** (*str*) –function name

Returns is this function has been registered in HiQ

Return type bool

`hiq.get_memory_mb()` → float

Get RSS in MB unit

Returns RSS of current process in MB

Return type float

`hiq.get_memory_kb()` → float

`hiq.get_memory_b()` → float

`hiq.ts_pair_to_dt(t1: float, t2: float)` → str

`hiq.ts_to_dt(timestamp: float)` → str

`hiq.utc_to_pst(utc: str, format_='%Y-%m-%dT%H:%M:%S+0000')` → str

utc2pst( "2021-06-19T20:32:18+0000" )

**hiq.get_graph_from_string**(s: str, time_format='_timestamp_', whole_node_name='__predict', with_ordered_node=True) → str

deserialize a string back into a HiQ tree and return the graph representation and ordered duration to the caller

```
>>> m = "t1^get_memory_mb,,0,54,5,0,1e-07,0#%n1*[None,inf,inf,0$0#[__
↪predict,9942.6796875,10064.375,0$0#[__pdf,9942.6796875,10013.
↪578125,0$0#][__txt,10013.578125,10175.28515625,0$0#[__det,10013.
↪578125,10174.68359375,0$0#[__ort,10228.8984375,10357.7421875,0$0
↪#]][__det,10174.68359375,10175.28515625,0$0#[__ort,10174.68359375,
↪10175.28515625,0$0#]]][_ort_sess,10175.28515625,10255.44921875,0$0
↪#]]]"
>>> print(get_graph_from_string(m))
[      9942.680 -      10064.375]  [100.00%] □_root_get_memory_
↪mb(121.6953)
[      9942.680 -      10064.375]  [100.00%]    l___predict(121.6953)
[      9942.680 -      10013.578]  [ 58.26%]       |___pdf(70.8984)
[     10013.578 -      10175.285]  [132.88%]       |___txt(161.7070)
[     10013.578 -      10174.684]  [132.38%]       |  |___det(161.
↪1055)
[     10228.898 -      10357.742]  [105.87%]       |  |  l___ort(128.
↪8438)
[     10174.684 -      10175.285]  [  0.49%]       |  l___det(0.6016)
[     10174.684 -      10175.285]  [  0.49%]       |     l___ort(0.
↪6016)
[     10175.285 -      10255.449]  [ 65.87%]       l__ort_sess(80.
↪1641)
ordered lowest level calls:__ort:129.4453, _ort_sess:80.1641, __
↪pdf:70.8984
```

**hiq.get_duration_from_hiq_string**(s: str, key: str) → float

```
>>> m = "t1^get_memory_mb,,0,54,5,0,1e-07,0#%n1*[None,inf,inf,0$0#[__
↪predict,9942.6796875,10064.375,0$0#[__pdf,9942.6796875,10013.
↪578125,0$0#][__txt,10013.578125,10175.28515625,0$0#[__det,10013.
↪578125,10174.68359375,0$0#[__ort,10228.8984375,10357.7421875,0$0
↪#]][__det,10174.68359375,10175.28515625,0$0#[__ort,10174.68359375,
↪10175.28515625,0$0#]]][_ort_sess,10175.28515625,10255.44921875,0$0
↪#]]]"
>>> print(get_duration_from_hiq_string(m, "__txt"))
161.70703125
>>> print(get_duration_from_hiq_string(m, "__ort"))
129.4453125
```

---

HiQ is a `declarative`, `non-intrusive`, `dynamic` and `transparent` tracking system for both monolithic application and distributed system. It brings the runtime information tracking and optimization to a new level without compromising with speed and system performance, or hiding any tracking overhead information. HiQ applies for both I/O bound and CPU bound applications.

To explain the four features, declarative means you can declare the things you want to track in a text file, which could be a json, yaml or even csv,and no need to change program code. Non-intrusive means HiQ doesn't requires to modify original python code. Dynamic means HiQ supports tracing metrics featuring at run time, which can be used for adaptive tracing. Transparent means HiQ provides the tracing overhead and doesn't hide it no matter it is huge or tiny.

In addition to latency tracking, HiQ provides memory, disk I/O and Network I/O tracking out of the box. The output can be saved in form of normal line by line log file, or HiQ tree, or span graph.

# CHAPTER 10

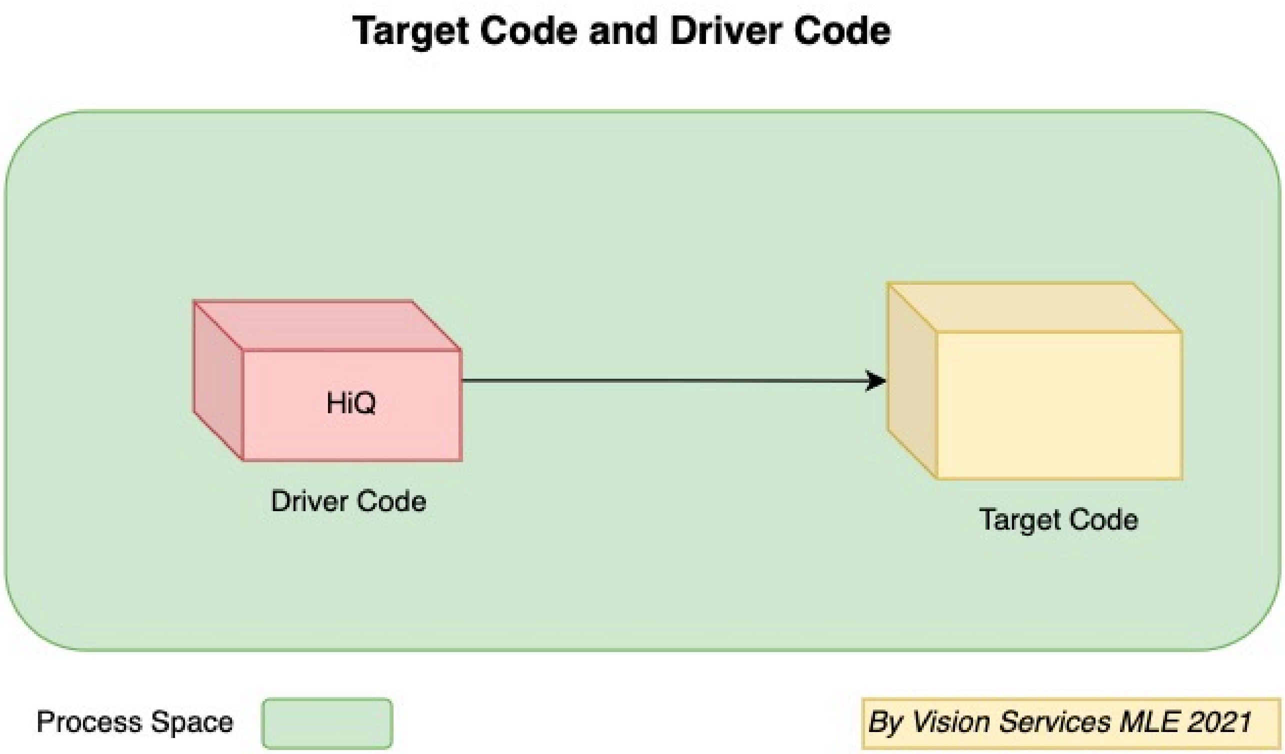
# INSTALLATION

```
pip install hiq-python
```

# CHAPTER 11

# GET STARTED

To use HiQ, you need to have `target code` and `driver code`.




Let start with a simplest example by running HiQ against a monolithic application. The target code is `main.py`:

```
import time
```

(continues on next page)

(continued from previous page)

```
def func1():
    time.sleep(1.5)
    print("func1")
    func2()

def func2():
    time.sleep(2.5)
    print("func2")

def main():
    func1()

if __name__ == "__main__":
    main()
```

In this target code, there is a simple chain of function calls: `main()` -> `func1` -> `func2`. We can actually run the target code:

```
cd examples
python main.py
```

And the output should be:

```
func1
func2
```

Now let's run the driver code, and if everything is fine, you should be able to see the output like this:

```
 python main_driver.py
func1
func2
[2021-11-01 21:54:18.222424 - 21:54:22.226879]  [100.00%] _root_time(4.0045)
                                                           [OH:163us]
[2021-11-01 21:54:18.222424 - 21:54:22.226879]  [100.00%]    l___main(4.0045)
[2021-11-01 21:54:18.222472 - 21:54:22.226868]  [100.00%]       l___func1(4.0044)
[2021-11-01 21:54:19.724213 - 21:54:22.226818]  [ 62.50%]          l___func2(2.5026)
```

- Explanation of driver code

```
import hiq


def run_main():
    driver = hiq.HiQLatency(
        hiq_table_or_path=[
            ["main", "", "main", "main"],
            ["main", "", "func1", "func1"],
            ["main", "", "func2", "func2"],
        ]
    )
    hiq.mod("main").main()
```

(continues on next page)

(continued from previous page)

```
    driver.show()


if __name__ == "__main__":
    run_main()
```

Line 1: import python module `hiq`.Line 5-11: create an object of class `hiq.HiQLatency` and declare we want to trace function `main()`, `func1()`, `func2()` in `main.py`.Line 12: call function `main()` in `main.py`.Line 13: print HiQ trees.

# CHAPTER 12

# DOCUMENTATION

HTML: HiQ Online Documents | PDF: Please check HiQ User Guide.

# CHAPTER 13

# EXAMPLES

Please check examples for usage examples.

# CHAPTER 14

# CONTRIBUTING

HiQ welcomes contributions from the community. Before submitting a pull request, please review our contribution guide.

# CHAPTER 15

# SECURITY

Please consult the security guide for our responsible security vulnerability disclosure process.

# CHAPTER 16

# LICENSE



## 16.1 Indices and tables

- genindex
- modindex
- search

# INDEX

## S



## T



## U



## W