Telemetry Service Private Edition Guide

Provision Telemetry Service

12/3/2025

Contents

- Administrator

Feature coming soon! Learn how to provision Telemetry Service.

Related documentation:

-
-
-

RSS:

- For private edition

Each Telemetry Client application has its own way to activate the connection to the Telemetry Service. The Telemetry Service can be configured to enable some advanced functionalities like custom trace contracts, monitored events, monitored metrics, and bucketized metrics.

Currently, the Telemetry service uses a remote storage for its configuration override like AWS S3 or Azure Blob Storage. To avoid the deployment of persistent volume in Private Edition, Environment variables are used for configuration.

```
TELEMETRY_CONFIG_TENANTS="{\"2868802f-1763-4ecd-94f6-203400001200\":\"pulse\"}"
TELEMETRY_CONFIG_CONTRACTS="[{ ... }]"
```

For provisioning, the following updates can be made to the **values.yaml** file:

```
tlm:
    context:
        envs:
            TELEMETRY_CONFIG_SERVICE: "env" # This will tell Telemetry service to use
environment variables provisioning
            TELEMETRY_CONFIG_TENANTS: "{\"2868802f-1763-4ecd-94f6-203400001200\":\"pulse\"}"
            TELEMETRY_CONFIG_CONTRACTS: '[{"appName": "wwe_ui", "properties": [ "connId",
        "agentSessionId", "browserSessionId", "interactionId", "POC_override" ],
"monitoredMetrics":
        [ { "name": "http_sync_req_StartContactCenterSession", "value": 2, "type":
        "gt" }, { "name": "http_sync_req_AttachUserData", "value": 5222000, "type":
        "lt" } ], "monitoredEvents": [ "error_.*", "disaster_recovery_.*",
"performance_worker_major",
        "performance_worker_severe" ]}, {"appName": "softphone", "properties": [ "ThisDN",
        "call_id", "call_uuid", "region", "level", "msgId", "sepsessionid", "value__NUM"
        ], "apdex": { "metrics": { "sip_call_mos": { "satisfiedTH": 3.6, "toleratedTH":
        2, "isExtended": true } } }}]'
            TELEMETRY_CONFIG_CORS: '{ "33cd4384-e0e7-4860-90e7-589712c33301":
{"urls":["http://localhost:8080",
        "http://localhost:3000"], "domains":[]}, "ed79bc34-768e-4d74-a924-cf10107c1807":
        {"urls":["http://localhost:8080", "http://localhost:7000"], "domains":[]}}'
```

Tenant provisioning

The Telemetry service has the information of the Contact Center by ID and not by name. Telemetry service provides the possibility of mapping contact center ID to a name by injecting it as an environment variable. This allows displaying the name of the contact center instead of the contact enter ID in the time series labels.

To add a name to the contact center ID, add the information in the **TELEMETRY_CONFIG_TENANTS** environment variable.

```
TELEMETRY_CONFIG_TENANTS: '{"my-contact-center-id":"my contact center"}'
```

As the JSON is parsed directly from the environment variable, use the [https://github.com/fastify/secure-json-parse JSON-SECURE-PARSE] library and [https://github.com/ajv-validator/ajv AJV] library for validating the JSON schema.

Provisioning Telemetry Service in AKS

Update CORS settings

Use the following command to update the CORS settings:

```
$ curl --location --request POST '/environment/v3/cors' \--header 'Content-Type: application/
json' \

--header 'Authorization: Bearer 201ad145-3b79-4d25-b88e-6c3279e00c63' \ --- Bearer Token

--data-raw '{

"data": {

"origin": "", ------------ TLM url

"contactCenterId": "" -------------- CCID

}

}'
```

Update the below setting in Configserver:

Navigate to the following location in configserver and update the below settings, **configserver** -> **cloudcluster application** -> **interaction-workspace section**.

```
system.telemetry.service =

system.telemetry.enabled = true

system.telemetry.enable-metrics = true
```

```
system.telemetry.enable-traces = true
```

Configuring Telemetry Contracts

Each application that wants to send data to Telemetry service needs a contract to be declared and provisioned in the service. Depending on the features, the contract values can be configures. An example of a contract with all features activated:

```
[{
  "appName": "nameOfTheApplication",
  "properties": ["property1", "property2"],
  "apdex": {
    "default": {
      "satisfiedTH": 1201,
      "toleratedTH": 4801
    },
    "metrics": {
      "http_async_req_Accept": {
        "satisfiedTH": 1200,
        "toleratedTH": 4800
      },
      "call_Quality_ReversedApdex": {
        "satisfiedTH": 4,
        "toleratedTH": 3,
        "isExtended": true
      }
    }
  },
  "buckets": {
    "foo": [400, 500, 900]
  },
  "monitoredMetrics": [
    { "name": "", "value": , "type": "" }
  ],
  "monitoredEvents": [
    "error_.*",
    "disaster_recovery_.*",
    "business_attribute_option_config_issue"
  ]
}]
```

Property	Description
appName	This is the application name to be used to identify the application. for example, wwe_ui / softphone / hca / nexus_chat
properties	An array of properties to serialize when traces are pushed to Telemetry service, all other properties will be ignored.
apdex	Apdex calculation is done by applying Satisfied and Tolerated Threshold to a metric. This metric can be any unit as long as the threshold that have to be applied are on the same range. Once a metric needs to have an apdex calculation the looking for threshold is applied and it goes like this:

Property	Description
	1. There is a lookup in the config file if the `metricname` is in the property **apdex.metrics** of the configuration object. 2. If the first step is not successful, there is a lookup on **apdex.default** values. 3. If none of the previous step is successful, the apdex calculation falls back to default threshold which are : SatisfiedTH: 1200 / toleratedTH: 4800. **Note**: Those default metrics are used to calculate the render speed of UI components in milliseconds.
isExtended	isExtended property is a Boolean value to activate or not the display of the raw numbers which have been used to calculate the apdex. Once it's set to true, 3 more counters will be added: • satisfied count • tolerated count • frustrated count **Note**: Apdex definition is first meant to define the lowest value as the better score. For specific concerns, Telemetry APDEX supports the reversed behavior if you switch values of satisfied and tolerated threshold.
buckets	Buckets are made with the interval declared between the thresholds; from -Infinity to +Infinity. And then those values will be exposed in buckets named with the index of the interval starting at 1 to X. For example: • Bucket definition: [400 , 500] The API receives those metrics : 200,340,350,600. This will end up with: • {metricType="Bucket", BucketId="1"} 3 • {metricType="Bucket", BucketId="2"} 0 • {metricType="Bucket", BucketId="3"} 1
monitoredMetrics	Monitored Events are declared in Telemetry contracts with plain text event or Regular Expression using the property, **monitoredEvents**. There is no default activation, definition is done per Telemetry client. Each metric received from Telemetry clients through the API endpoint **telemetry/v1/record** is evaluated by the monitored metric module. If a Monitored Metric has been configured for this metric, the received value

Property	Description

	is evaluated against the threshold configured for this monitored metric.
	• "**appName**":: the keyword representing the Telemetry client application ('wwe_ui', 'softphone', 'hca', 'nexus_chat'...).
	• "**name**":: the name of the metric as posted by the Telemetry client application at runtime.
	• "**value**":: the numeric value representing the threshold that can trigger the recording of the monitored metric. It must be consistent in unit with the value that the Telemetry client application is posting at runtime.
	• "**type**":"": the type of threshold
	  • 'gt' (default): the values greater than the threshold are recorded as 'monitored metric' records.
	  • 'lt': the values lower than the threshold are recorded as 'monitored metric' records.
	The data is collected and compiled every 10 minutes by default and can be changed with the environment variable, **TELEMETRY_EVENT_MONITOR_TIME**. After each compilation the data is sent to ElasticSearch like other traces in the index **tlm-traces-***. Those can be identified with the property **trace_type** set to **eventMonitor**