Oracle® Banking Microservices Architecture

Observability User Guide

Release 14.6.1.0.0
F61129-01
August 2022

Oracle Banking Microservices Architecture Observability User Guide, Release 14.6.1.0.0

F61129-01

Contents

Preface

Purpose

This guide helps to use the tools that enable users to observe the Oracle Banking Microservices Architecture suite of products better.

The sections provide tools that can enable a user to:

- Observe the spans associated with various API calls and the response of each API.
- Troubleshoot Kafka better.
- Aggregate logs and interpret out of log searches.

This guide also describes recommended tools to enhance monitoring and observability aspects of the Oracle Banking Microservices Architecture products.

Audience

This guide is intended for the implementation teams.

Acronyms and Abbreviations

The list of the acronyms and abbreviations that you are likely to find in the guide are as follows:

Table 1 Acronyms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
ELK	Elasticsearch Logstash Kibana
UI	User Interface
URL	Uniform Resource Locator

List of Topics

This guide is organized as follows:

Topics	Description
Observability Improvements using Zipkin Traces	This topic provides the information about observability improvements using Zipkin traces.
Observability Improvements Logs using ELK Stack	This topic provides the information about observability improvements logs using ELK stack.
Health Checks	This topic explains the possible approaches to monitor health of Oracle Banking Microservices Architecture services.

Topics	Description
Troubleshooting Kafka Issues	This topic provides the information about troubleshooting Kafka issues.
Troubleshooting Flyway Issues	This topic provides the information about troubleshooting flyway issues.

Prerequisites

The prerequisites are as follows:

- Basic understanding of Eventing platform.
- Basic understanding application log analysis using tools.
- Basic understanding DB changes.

General Prevention

Do not make any changes to Flyway scripts manually.

Best Practices

The best practices are as follows:

- It is ideal to have ELK stack installed on a separate VM outside the product VMs to ensure flow of logs in case of app crash.
- Log levels can be adjusted to INFO and above to enable relevant logs to flow in.
- Verify all Kafka settings as per User Troubleshooting Guide before the health check.

Related Documents

The related documents are as follows:

- *Oracle Banking Getting Started User Guide*
- *Troubleshooting Guide*

1

Observability Improvements using Zipkin Traces

This topic describes the troubleshooting procedures using the Zipkin Traces.

- Setting Zipkin Server
  This topic provides the information to install and setup Zipkin server.
- Troubleshooting Zipkin
  This topic provides the systematic instructions to troubleshooting using Zipkin Traces.
- Zipkin Issues
  This topic describes the various issues faced in Zipkin.

1.1 Setting Zipkin Server

This topic provides the information to install and setup Zipkin server.

Refer to **ANNEXURE - 2** for the installation and setup the Zipkin server.

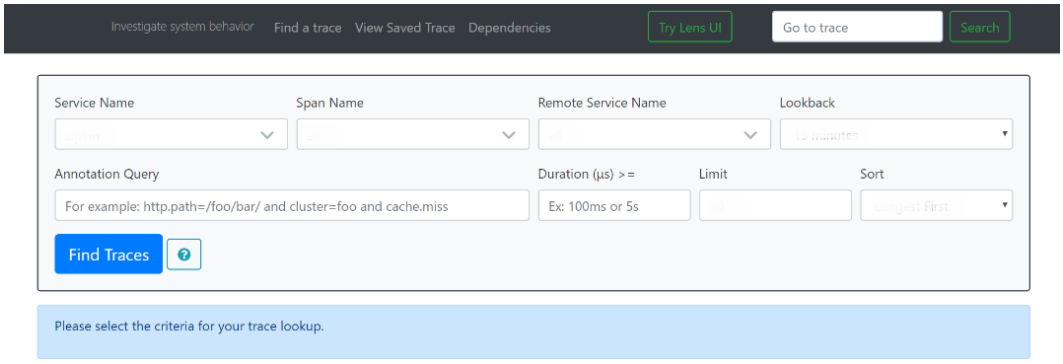
1.2 Troubleshooting Zipkin

This topic provides the systematic instructions to troubleshooting using Zipkin Traces.

1. Launch the Zipkin URL.

   The **Layout of Zipkin** screen displays.

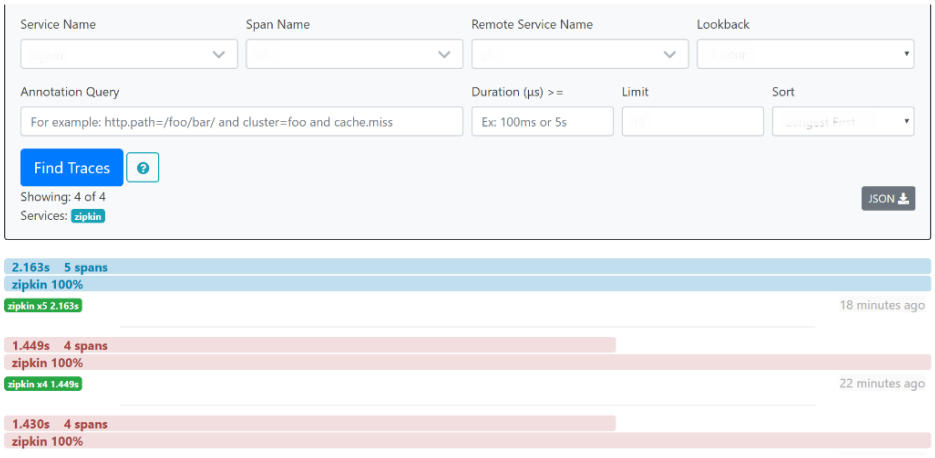
Figure 1-1 Layout of Zipkin



2. Use **Search** to find the traces of required API calls and services.

   The **List of Traces** screen displays.

Figure 1-2 List of Traces

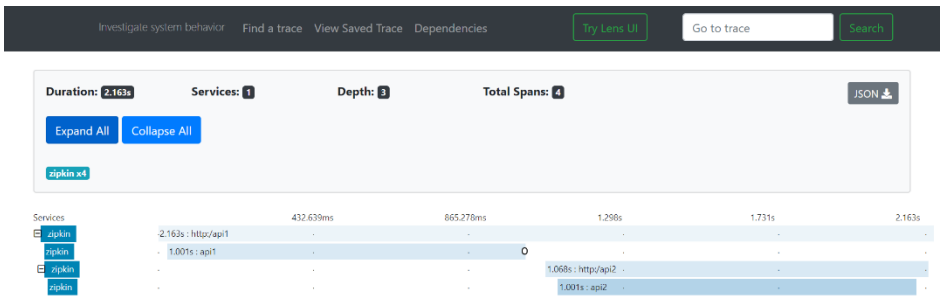


> **Note:**
>
> The search options given in the user interface are self-explanatory, and there is another UI option (**Try Lens UI**). It is given a different user interface with the same functionality. The list of the traces can be seen as shown in the above screen. Error API calls are made to showcase how to track errors. The blue listing shows the successful API hits, and the red listing indicates the errors. Each block indicates a single trace in the listing.

3. Open an individual trace to view the details of the trace.

   The **Individual Trace** screen displays.

Figure 1-3 Individual Trace

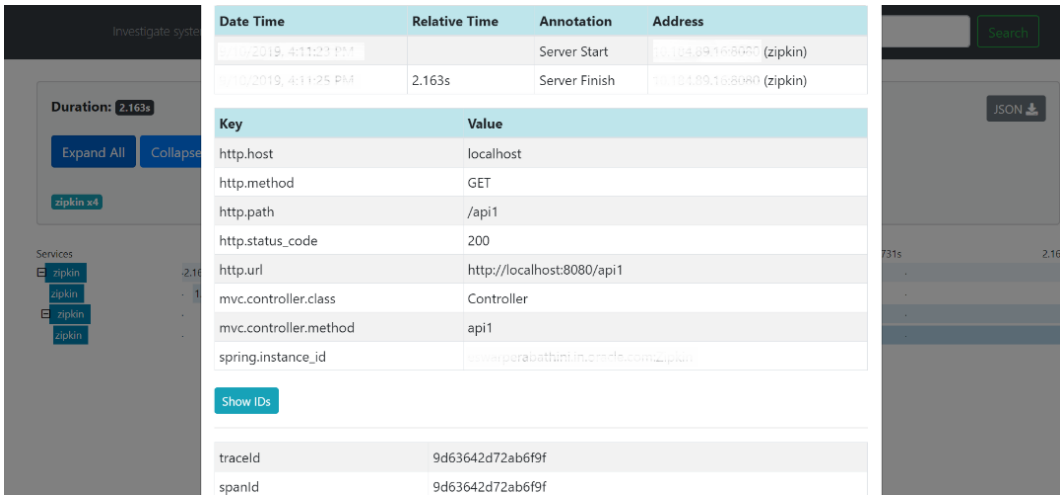


> **Note:**
>
> **Individual Trace** describes the time taken for each block. As the two custom spans are created inside two service calls, user can find a total of four blocks. The time taken for an individual block is shown in the above screen.

4. Click an individual block to display the details.

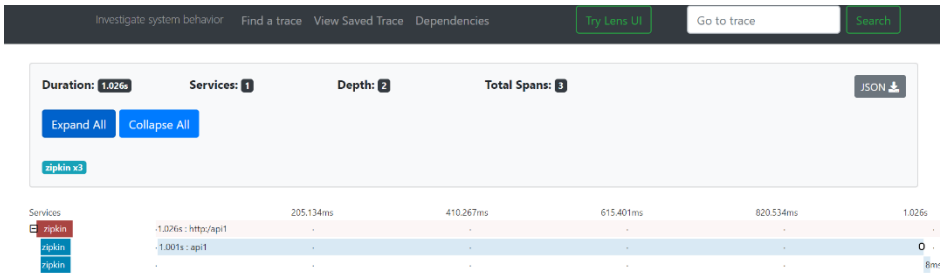
   The **Details of Individual Block** screen displays.

Figure 1-4 Details of Individual Block



> **Note:**
>
> The user can also view the span block details and logging events in the Zipkin UI as small circular blocks. An example of an error log is shown in the below screen.

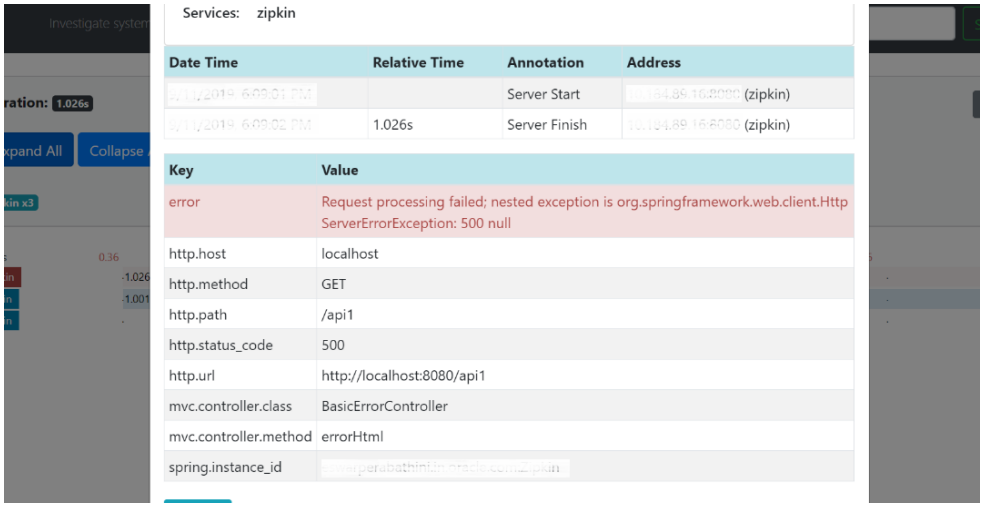
Figure 1-5 Sample Error Log



5. Click the error portion to get clear details and place of the error.

The **Details of Error** screen displays.

Figure 1-6 Details of Error

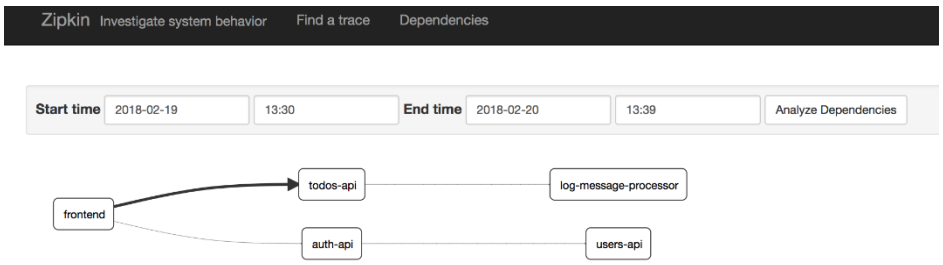


> **Note:**
>
> If the Lens UI is used in Zipkin, the above figures are not applicable but are relatable to the Lens UI as well. Traces of the application can be found using TraceId. The TraceId can be found in the debug logs of the deployment when spring-cloud-sleuth is included in the dependencies (included in spring-cloud-starter-Zipkin dependency).

6. Click **Dependencies** to get the dependency graph information between micro-services.

   The **Sample Dependency Graph** displays.

Figure 1-7 Sample Dependency Graph



1.3 Zipkin Issues

This topic describes the various issues faced in Zipkin.

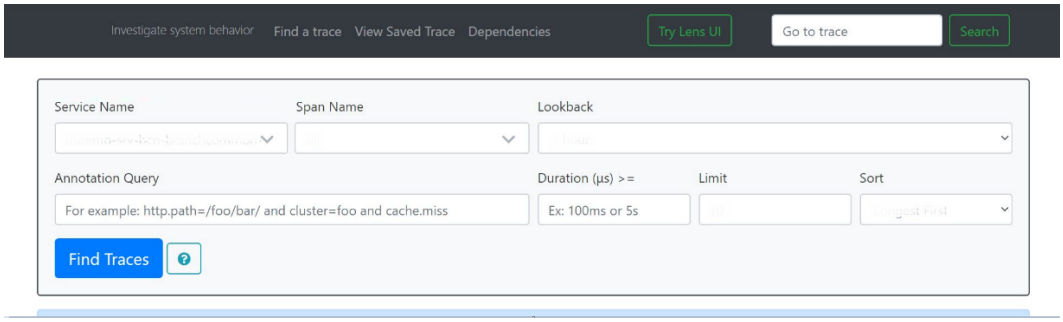
- Application Service is not Registered
  This topic describes the systematic instructions to register the application service.
- 404 Error
  This topic provides the information to troubleshoot the 404 Error in the application.
- Unable to Change Zipkin Default Port Number
  This topic provides the information to change the Zipkin Default Port Number in the application.

1.3.1 Application Service is not Registered

This topic describes the systematic instructions to register the application service.

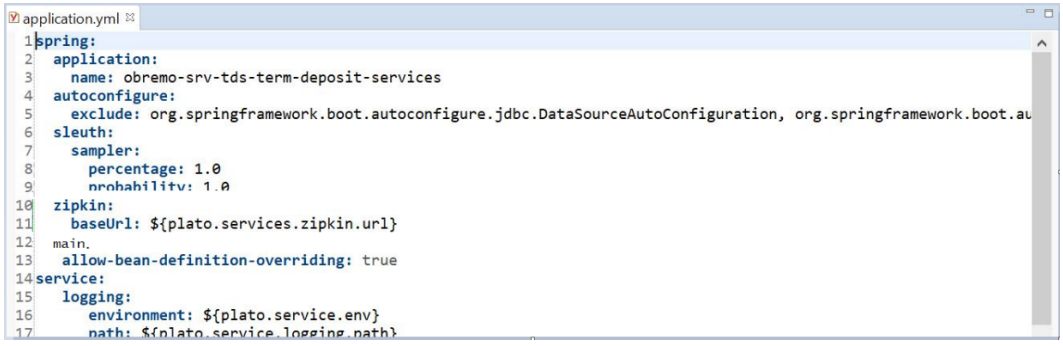
1. Check the applications that are sending the trace report to the Zipkin server from **Service Name** drop-down list.

Figure 1-8 Find Traces



2. If the required application is not listed in Zipkins, check the `application.yml` file for Zipkin base URL configuration.

Figure 1-9 Application.yml File



> **Note:**
>
> The shipped `application.yml` should have the Zipkin entry. Every service should have a `spring-cloud-sleuth-zipkin` dependency added in the build gradle file for the service to generate and send trace Id and span Id.

3. Specify the necessary values are as follows:

   Compile group: org.springframework.cloud
   Name: spring-cloud-sleuth-zipkin

   Version: 2.1.2.RELEASE

Figure 1-10 Branch Common Services

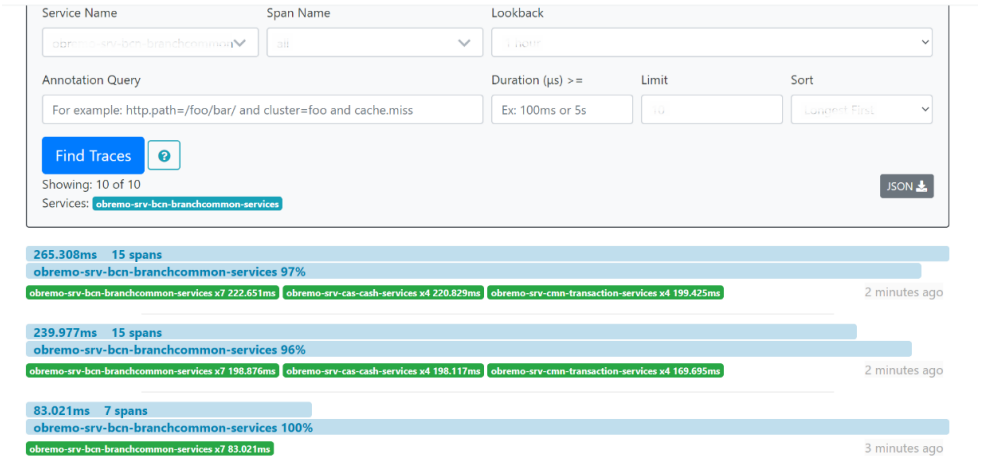
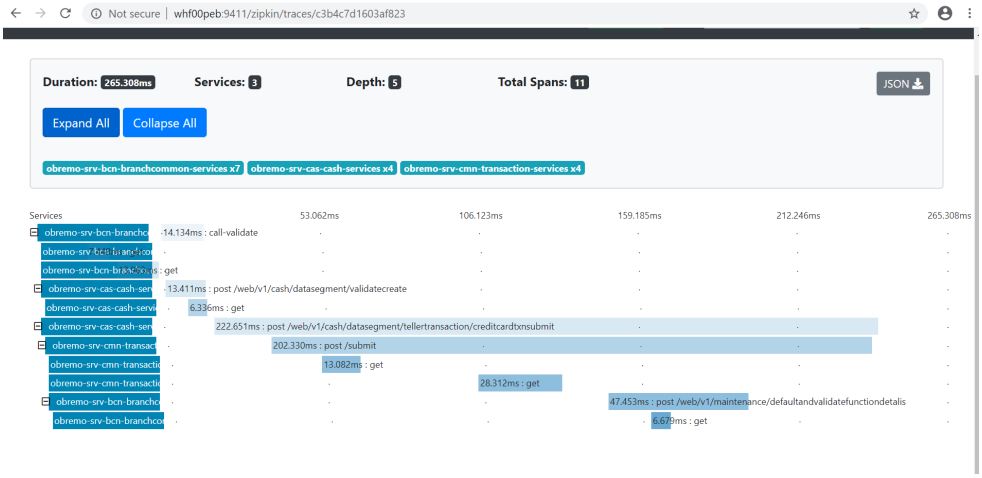


Figure 1-11 Branch Common Services Trace



1.3.2 404 Error

This topic provides the information to troubleshoot the 404 Error in the application.

If there is a 404 error, check if the **zipkin-server.jar** is running in the system where the application is deployed. To check this, execute the following command:

```
netstat -ltnup | grep ':9411'
```

The output should be like as below:

```
tcp6 0 0 :::9411 :::* LISTEN 10892/java
```

Here 10892 is the PID.

1.3.3 Unable to Change Zipkin Default Port Number

This topic provides the information to change the Zipkin Default Port Number in the application.

Zipkin default port number is not editable.

Hence, make sure that the port 9411 is available to start `Zipkin-server.jar file`.

2

Observability Improvements Logs using ELK Stack

This topic describes the troubleshooting procedures using the ELK Stack.

- Setting up ELK Stack
  This topic describes the systematic instruction to download, run and access the ELK Stack.
- Kibana Logs
  This topic describes the information to setup, search and export the logs in Kibana.

2.1 Setting up ELK Stack

This topic describes the systematic instruction to download, run and access the ELK Stack.

Download ELK Stack

1. Download the Elastic search from https://www.elastic.co/downloads/elasticsearch.
2. Download the Kibana from https://www.elastic.co/downloads/kibana.
3. Download the Logstash from https://www.elastic.co/downloads/logstash.

Figure 2-1 ELK Setup

```
# Kibana is served by a back end server. This setting specifies the port to use.
#server.port: 5601

# Specifies the address to which the Kibana server will bind. IP addresses and host names are both valid values.
# The default is 'localhost', which usually means remote machines will not be able to connect.
# To allow connections from remote users, set this parameter to a non-loopback address.
server.host: "whf00peb"

# Enables you to specify a path to mount Kibana at if you are running behind a proxy.
# Use the `server.rewriteBasePath` setting to tell Kibana if it should remove the basePath
# from requests it receives, and to prevent a deprecation warning at startup.
# This setting cannot end in a slash.
#server.basePath: ""

# Specifies whether Kibana should rewrite requests that are prefixed with
# `server.basePath` or require that they are rewritten by your reverse proxy.
# This setting was effectively always `false` before Kibana 6.3 and will
# default to `true` starting in Kibana 7.0.
#server.rewriteBasePath: false

# The maximum payload size in bytes for incoming server requests.
#server.maxPayloadBytes: 1048576

# The Kibana server's name.  This is used for display purposes.
#server.name: "your-hostname"

# The URL of the Elasticsearch instance to use for all your queries.
elasticsearch.url: "http://localhost:9200"

# When this setting's value is true Kibana uses the hostname specified in the server.host
```

> **Note:**
>
> Default port for Elastic search is 9200 and the default port for Kibana is 5601.

- Run ELK Stack
  This topic describes the systematic instruction to run the ELK Stack.
- Access Kibana
  This topic describes the information to access the kibana.

2.1.1 Run ELK Stack

This topic describes the systematic instruction to run the ELK Stack.

Perform the following steps:

1. Run the elasticsearch.sh file present in the folder path `/scratch/software/ELK/elasticsearch-6.5.1/bin`.
2. Configure the Kibana to point the running instance of elastic search in the kibana.yml file.
3. Follow the below steps to configure the Logstash.
   a. **Input:** This configuration is used to provide the log file location for the Logstash to read from.
   b. **Filter:** This configuration is used to control or format the read operation (Line by line or Bulk read).
   c. **Output:** This configuration is used to provide the running elastic search instance to send the data for persisting.

Figure 2-2 Logstash Configuration

```
input {
  file {
    type => "java"
    path => "/scratch/Software/Weblogic_Installation/user_projects/domains//base_domain/logs/obremo-srv-cmn-transaction-services.log"
    codec => multiline {
      pattern => "Transation Ended!"
      negate => "true"
      what => "next"
    }
  }
}

filter {
  #If log line contains tab character followed by 'at' then we will tag that entry as stacktrace
  if [message] =~ "\tat" {
    grok {
      match => ["message", "^(\tat)"]
      add_tag => ["stacktrace"]
    }
  }

}

output {

  stdout {
    codec => rubydebug
  }

  # Sending properly parsed log events to elasticsearch
  elasticsearch {
    hosts => ["localhost:9200"]
  }
}
```

2.1.2 Access Kibana

This topic describes the information to access the kibana.

Figure 2-3 Access Kibana

2.2 Kibana Logs

This topic describes the information to setup, search and export the logs in Kibana.

Setup Dynamic Log Levels in Oracle Banking Microservices Architecture Services without Restart

The plato-logging-service is dependent on two tables, which needs to be present in the PLATO schema (JNDI name: jdbc/PLATO). The two tables are as follows:

- **PLATO_DEBUG_USERS**: This table contains the information about whether the dynamic logging is enabled to a user for a service. The table will have records, where DEBUG_ENABLED values for a user and a service have values **Y** or **N**, and depending on that plato-logger will enable dynamic logging.

Figure 2-4 PLATO_DEBUG_USERS

	ID	DEBUG_ENABLED	SERVICE_CODE	USER_ID
1	2	Y	plato-logger-ref	
2	3	Y	platoref	

- **PLATO_LOGGER_PARAM_CONFIG**: This table contains the key-value entries of different parameters that can be changed at runtime for the dynamic logging.

Figure 2-5 PLATO_LOGGER_PARAM_CONFIG

	ID	MODIFY_FIELD	PARAM_NAME	PARAM_VAL
1	3	N	LOG_PATH	C:\
2	2	N	LOG_LEVEL	INFO
3	1	N	LOG_MSG_WITH_TIME	Y

The values that can be passed are as follows:

- **LOG_PATH**: This specifies a dynamic logging path for the logging files to be stored. Changing this in runtime, changes the location of the log files at runtime. If this value is not passed then by default, the LOG_PATH value is taken from the -D parameter of `plato.service.logging.path`.
- **LOG_LEVEL**: The level of the logging can be specified on runtime as **INFO** or **ERROR** etc. The default value of this can be set in the `logback.xml`.
- **LOG_MSG_WITH_TIME**: Making this **Y** appends the current date into the log file name. Setting the value of this as **N** cannot append the current date into the filename.

Search for Logs in Kibana

Search logs in Kibana using https://www.elastic.co/guide/en/kibana/current/search.html.

Export Logs for Tickets

Perform the following steps to export logs:

1. Click **Share** from the top menu bar.
2. Select the **CSV Reports** option.
3. Click **Generate CSV** button.

3
Health Checks

This section describes the possible approaches to monitor health of Oracle Banking Microservices Architecture services.

- Discovery Health Check
  This topic describes about the health status of all the registered services and their instances.
- Actuator Health Indicator Endpoint
  This topic describes about the Health Status of the Endpoint

3.1 Discovery Health Check

This topic describes about the health status of all the registered services and their instances.

Figure 3-1 Discovery Health Check

CONFIG-SERVICE	n/a (1)	(1)	UP (1) - whf00fpr.in.oracle.com:config-service:7002
FEED-DOMAIN1-SERVICES	n/a (4)	(4)	UP (4) - localhost:feed-domain1-services:7003 , whf00fpr.in.oracle.com:feed-domain1-services:7003 , host.docker.internal:feed-domain1-services:7003 , TGEETEY-T490.in.oracle.com:feed-domain1-services:7203
PLATO-API-GATEWAY	n/a (2)	(2)	UP (2) - SAHT-T490.in.oracle.com:plato-api-gateway:7002 , whf00fpr.in.oracle.com:plato-api-gateway:7002
PLATO-BATCH-SERVER	n/a (1)	(1)	UP (1) - whf00fpr.in.oracle.com:plato-batch-server:8088
PLATO-DISCOVERY-SERVICE	n/a (1)	(1)	UP (1) - whf00fpr.in.oracle.com:plato-discovery-service:7002
PLATO-FEED-SERVICES	n/a (4)	(4)	UP (4) - host.docker.internal:plato-feed-services:7003 , whf00fpr.in.oracle.com:plato-feed-services:7003 , TGEETEY-T490.in.oracle.com:plato-feed-services:7203 , localhost:plato-feed-services:7003
PLATO-O	n/a (1)	(1)	UP (1) - whf00fpr
PLATO-O4	n/a (1)	(1)	UP (1) - DEEPMATH-T490
PLATO-RULE-SERVICE	n/a (1)	(1)	UP (1) - whf00fpr.in.oracle.com:plato-rule-service:7003
PLATO-UI-CONFIG-SERVICES	n/a (1)	(1)	UP (1) - whf00fpr.in.oracle.com:plato-ui-config-services:7002

3.2 Actuator Health Indicator Endpoint

This topic describes about the Health Status of the Endpoint

- Generic Service
- Kafka Consumers and Producers

3.2.1 Generic Service

To check the health status of any service hit the below endpoint:

```
http://<Host>:<Port>/context_path/actuator/health
```

Example:`http://localhost:8089/refapp/actuator/health`

With headers similar to:

userId: **XYZ**

appId: **PLATOREFAPP**

entityId: **DEFAULTENTITY**

branchCode: **000**

Sample Response:

```
{
    "status": "UP"
}
```

To get more detailed health status add following property:

`management.endpoint.health.show-details=always`

Sample Response:

```
{
    "status": "UP",
    "components": {
        "binders": {
            "status": "UP",
            "components": {
                "kafka": {
                    "status": "UP"
                }
            }
        },
        "clientConfigServer": {
            "status": "UP",
            "details": {
                "propertySources": [
                    "refapp-jdbc"
                ]
            }
        },
        "db": {
            "status": "UP",
            "components": {
                "PLATO_LOGGER_DS": {
                    "status": "UP",
                    "details": {
                        "database": "Oracle",
                        "validationQuery": "isValid()"
                    }
                },
                "dataSource": {
                    "status": "UP",
                    "details": {
                        "database": "Oracle",
```

```
                    "validationQuery": "isValid()"
                }
            }
        }
    },
    "discoveryComposite": {
        "status": "UP",
        "components": {
            "discoveryClient": {
                "status": "UP",
                "details": {
                    "services": [
                        "plato-feed-services",
                        "plato-api-gateway",
                        "plato-rule-service",
                        "refapp"
                    ]
                }
            },
            "eureka": {
                "description": "Remote status from Eureka server",
                "status": "UP",
                "details": {
                    "applications": {
                        "PLATO-API-GATEWAY": 1,
                        "PLATO-RULE-SERVICE": 1,
                        "REFAPP": 1,
                        "PLATO-FEED-SERVICES": 4,
                    }
                }
            }
        }
    },
    "diskSpace": {
        "status": "UP",
        "details": {
            "total": 248031522816,
            "free": 81710915584,
            "threshold": 10485760,
            "exists": true
        }
    },
    "hystrix": {
        "status": "UP"
    },
    "ping": {
        "status": "UP"
    },
    "refreshScope": {
        "status": "UP"
    }
  }
}
```

3.2.2 Kafka Consumers and Producers

To check the health status of kafka consumers and producers hit the following endpoint: `http://<Host>:<Port>/context_path/actuator/health`

To stop discovery service from routing requests to kafka consumers or producers when connection to kafka is not successful, following flag needs to be set: **`eureka.client.healthcheck.enabled=true`**

4
Troubleshooting Kafka Issues

This topic describes the troubleshooting procedures for the Kafka issues.

- Kafka Health
  This topic describes the troubleshooting procedures for the Kafka Health.
- Prometheus and Grafana
  This topic describes about the Troubleshooting Kafka issues using Prometheus and Grafana.

4.1 Kafka Health

This topic describes the troubleshooting procedures for the Kafka Health.

- Verify Kafka Health
- Verify Zookeeper Health

4.1.1 Verify Kafka Health

Run the below command and verify: **`$ netstat -tlnp | grep :9092`**

> **Note:**
>
> 9092 is default port of kafka

4.1.2 Verify Zookeeper Health

Kafka instance will not start if Zookeeper is not yet started.

1. Run the below command and verify.
   ```
   $ netstat -tlnp | grep :2181 (2181 is default port of zookeeper)

   tcp6 0 0 :::2181 :::* LISTEN 19936/java
   ```
2. To debug, check if the permissions of Kafka log folder are correct.
   The log folder path can be found by looking at the value of the property `log.dirs` in the `server.properties` file of Kafka installation.

4.2 Prometheus and Grafana

This topic describes about the Troubleshooting Kafka issues using Prometheus and Grafana.

- Prometheus Setup
- JMX-Exporter Setup
- Grafana Setup

- Prometheus Metrics

4.2.1 Prometheus Setup

Prometheus is an open-source project, which helps monitoring of the applications metrics. It is widely used for the monitoring of Kafka and its metrics. The installer for Prometheus can be downloaded Prometheus from https://prometheus.io/download/.

4.2.2 JMX-Exporter Setup

A JMX-Exporter application is used to integrate with the Kafka broker as a Java agent to expose the values of JMX MBeans as an API. The JMX-Exporter is used by the Prometheus to fetch the values of the JMX metrics.

Perform the following steps:

1. Download the latest jmx_prometheus_javaagent jar file from the maven repository in the Kafka directory along with the bin, config directories.

> **Note:**
>
> This can be used to monitor consumer_lagthttps://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/0.15.0/jmx_prometheus_javaagent-0.15.0.jar

2. Set the *KAFKA_OPTS* variable to the desired value to execute the jar as a java agent `export KAFKA_OPTS="$KAFKA_OPTS -javaagent:$PWD/jmx_prometheus_javaagent-0.15.0.jar=7071:$PWD/kafka-0-8-2.yml"`

> **Note:**
>
> You can choose the port according to our preference.

3. Restart Kafka Broker.

4.2.3 Grafana Setup

Perform the following steps:

1. Download Grafana from https://grafana.com/grafana/download in the stand-alone application mode, and extract its contents.
2. Go to the bin folder in the extracted contents, and start the Grafana server.

> **Note:**
>
> Grafana should start on the default port 3000 (HOST: 3000). The default user and password for Grafana are admin/admin.

Perform the following steps to integrate Grafana with the Prometheus instance installed:

3. Click on the Grafana logo to open the sidebar.
4. Click **Data Sources** in the sidebar.
5. Choose **Add New**.
6. Select **Prometheus** as the data source.
7. Click **Add** to test the connection and to save the new data source.

4.2.4 Prometheus Metrics

The Prometheus Metrics are as follows:

- process_cpu_seconds_total.
- http_request_duration_seconds.
- node_memory_usage_bytes.
- http_requests_total.
- process_cpu_seconds_total.

5
Troubleshooting Flyway Issues

This topic describes the troubleshooting procedures for the flyway issues.

- Failed Migrations
  This topic describes about the Troubleshooting procedures for failed migrations.

5.1 Failed Migrations

This topic describes about the Troubleshooting procedures for failed migrations.

- Success Column Verification
- Migration Checksum Mismatch for a Version
- Placeholder errors

5.1.1 Success Column Verification

Perform the following steps for the success column verification:

1. Check the f`lyway_schema_history` table to identify the migration record with success column as ‘0’.
2. Delete the record with status as ‘0.
3. Restart deployment.

5.1.2 Migration Checksum Mismatch for a Version

Perform the following steps for the success column verification:

1. Make sure that the flyway script is not manually updated before deployment.
2. If yes, then replace with original and restart deployment.

5.1.3 Placeholder errors

Pass the placeholder values using `setUserOverrides.sh` in Weblogic. Alternatively, these issues can be debugged from Weblogic console during deployment. In addition, the application specific logs can be verified for further inputs.

Index