Observability User Guide

Oracle Banking Trade Finance Process Management

Release 14.7.1.0.0

Part Number F82594-01

May 2023

ORACLE®

FINANCIAL SERVICES

Observability User Guide

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax: +91 22 6718 3001

https://www.oracle.com/industries/financial-services/index.html

ORACLE®

Contents

ORACLE®

1. Preface

1.1 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:

1. Observe the spans associated with various API calls and the response of each API.

2. Troubleshoot Kafka better.

3. Aggregate logs and interpret out of log searches.

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

1.2 Audience

This guide is intended for the implementation teams.

1.3 Acronyms, Abbreviations, and Definitions

The following acronyms/abbreviations are used in this guide:

Table 1: Acronyms and Abbreviations

Acronyms	Definition
API	Application Programming Interface
ELK	Elasticsearch Logstash Kibana
UI	User Interface
URL	Uniform Resource Locator

1.4 Document Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

1.5 List of Topics

This guide is organized into the following topics:

ORACLE®

Table 2: List of Topics

Topics	Description
Observability Improvements using Zipkin Traces	This topic explains the possible ways and benefits of using tools like Zipkin to enhance troubleshooting possibilities.
Observability Improvements Logs using ELK Stack	This topic explains the possible log aggregation and search features that can be availed using ELK stack.
Health Checks	This topic explains the possible approaches to monitor health of Oracle Banking Microservices Architecture services.
Troubleshooting Kafka Issues	This topic explains the steps to troubleshoot basic issues in Kafka.
Troubleshooting Flyway Issues	This topic explains the steps to troubleshoot Flyway issues during deployment.

1.6 <u>Prerequisites</u>

The prerequisites are as follows:

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

1.7 <u>General Prevention</u>

Do not make any changes to Flyway scripts manually.

1.8 <u>Best Practices</u>

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.

1.9 <u>Related Documentation</u>

The related documentation are as follows:

- Getting Started User Guide
- Troubleshooting Guide

2. Observability Improvements using Zipkin Traces

This section describes the troubleshooting procedures using the Zipkin Traces.

2.1 Setting Zipkin Server

Zipkin works as an independent application, and it can be downloaded as a runnable jar from the official website of Zipkin https://zipkin.io/. The latest version of Zipkin needs a Java version above 8.

The direct download link of jar is as follows:

https://search.maven.org/remote_content?g=io.zipkin&a=zipkin-server&v=LATEST&c=exec

- The downloaded jar can be executed using the java –jar JAR_NAME command.
- The configuration of Zipkin can be done environment variables. The port of the Zipkin can be set using QUERY_PORT environment variable.
- The application starts on the port number assigned for QUERY_PORT environment variable or its default value of 9411. The web UI of Zipkin can be accessed at http://localhost:PORT.
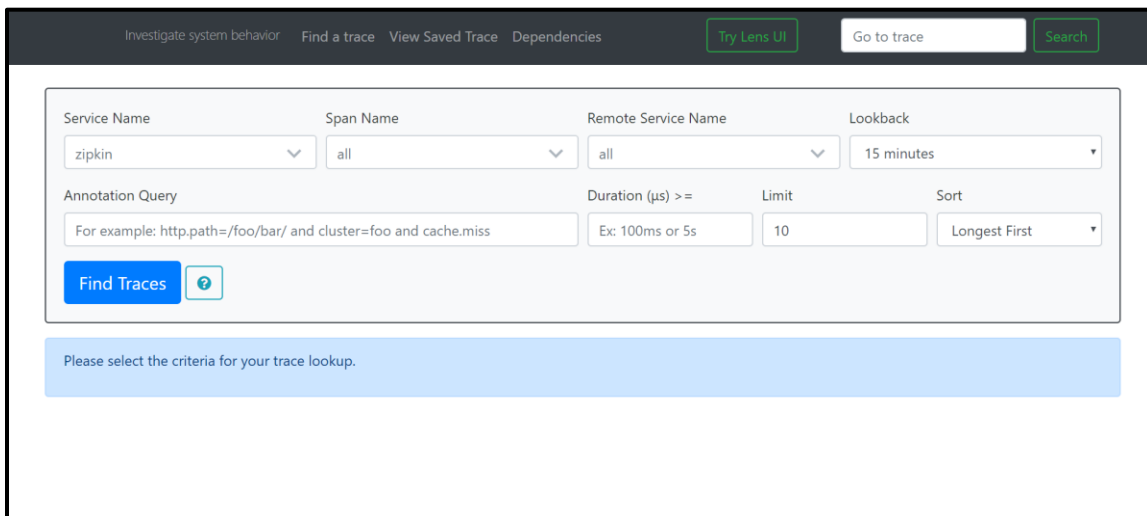
2.2 Login to Zipkin

Perform the following steps for the troubleshooting using Zipkin Traces:

1. Launch the Zipkin URL.

NOTE: The basic layout of Zipkin is shown in *Figure 1*.

Figure 1: Layout of Zipkin

2. Use the search option to find the traces of required API calls and services.

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 *Figure 2.* Error API calls are made to showcase how to track errors. The blue listings show the successful API hits, and the red listings indicate errors. Each block indicates a single trace in the listings.

Figure 2: List of Traces
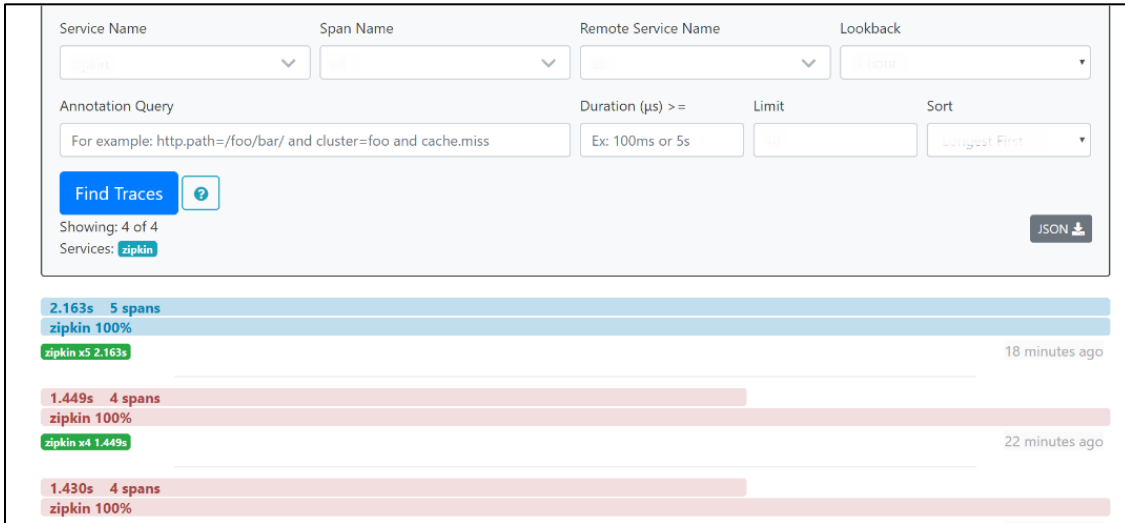
3. Open the individual trace to the details of the trace.

NOTE: *Figure 3* shows an individual trace when it is opened. It also describes the time taken for each block. As the two custom spans are created inside two service calls, you can find a total of four blocks. The time taken for an individual block can be seen in *Figure 3*.

Figure 3: Individual Trace
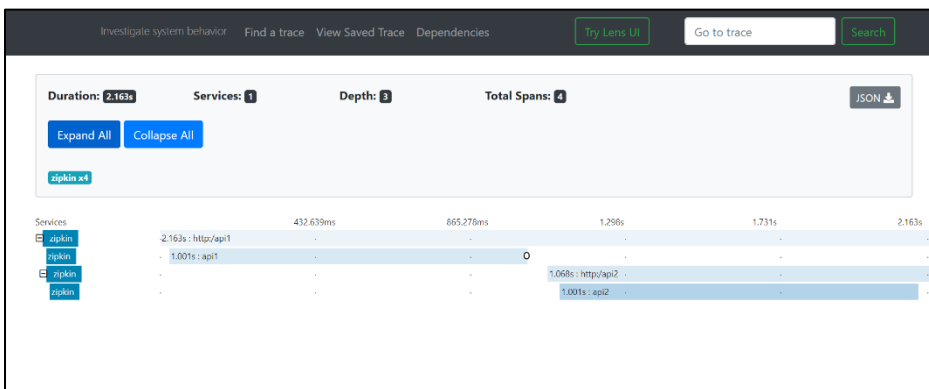
4. Click an individual block to display the details.

Figure 4: Details of Individual Block

NOTE: The details of the specific span block are shown in *Figure 4* and the logging events can also be seen in the Zipkin UI as small circular blocks. An example of an error log is shown in *Figure 5*.

Figure 5: Sample Error Log
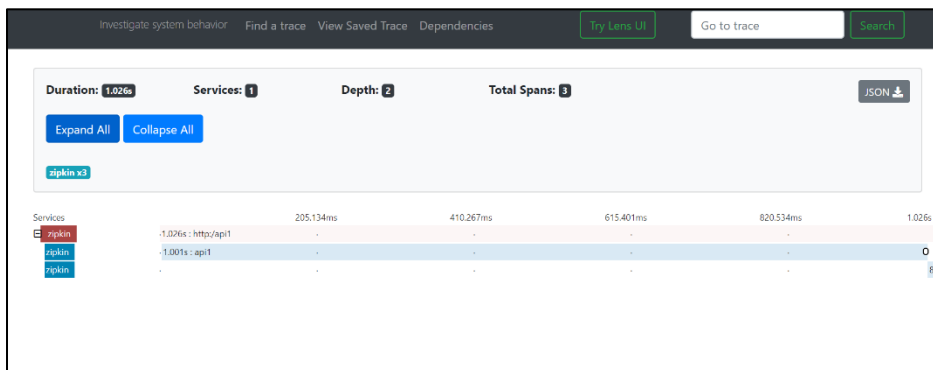
5. Click on the error portion to get a clear detail about the error, and where the error has arisen. An example is shown in *Figure 6*.

Figure 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** tab to get the dependency graph info between micro-services. An example dependency graph is shown in *Figure 7*.

Figure 7: Sample Dependency Graph
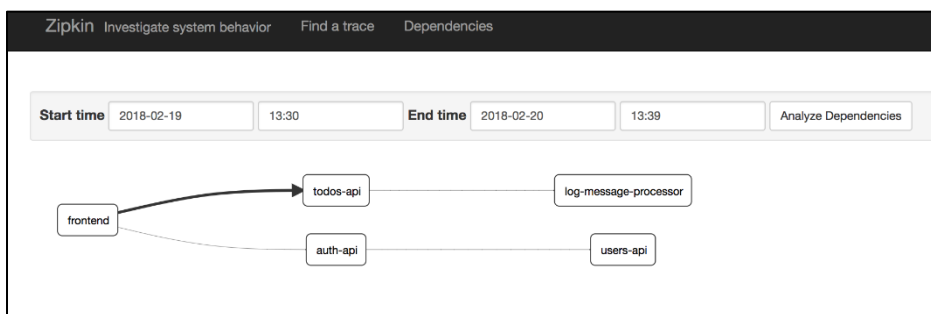
2.3 Zipkin Issues

2.3.1 Application Service is not Registered

Perform the following steps to find the cause of this error:

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

ORACLE®

Figure 8: Find Traces

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

Figure 9: Application.yml File

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

3. The necessary values are as follows:

Compile group: *'org.springframework.cloud'*

name: *'spring-cloud-sleuth-zipkin'*

version: *'2.1.2.RELEASE'*

ORACLE®

Figure 10: Branch Common Services
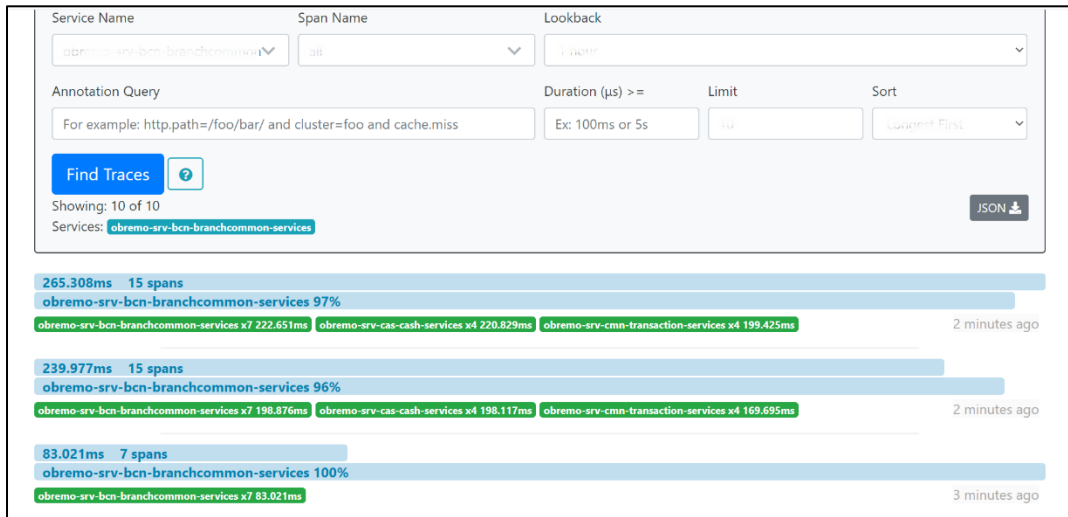
Figure 11: Branch Common Services Trace
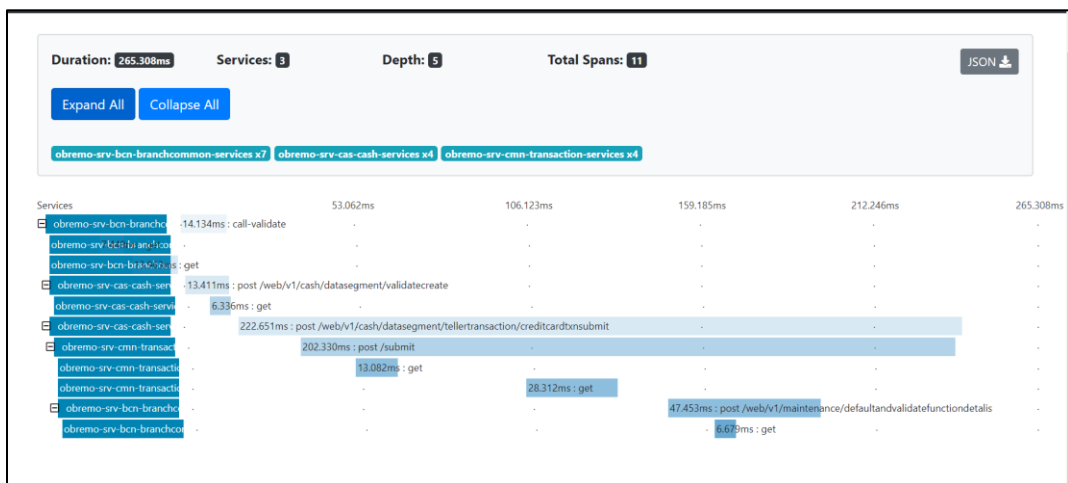
2.3.1.1 404 Error

If there is 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'

Output should be like:

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

Here 10892 is the PID.

2.3.1.2 Unable to Change Zipkin Default Port Number

Zipkin default port number is not editable. Hence, make sure that the port 9411 is available to start *Zipkin-server.jar* file.

ORACLE®

3. Observability Improvements Logs using ELK Stack

This section describes the troubleshooting procedures using the ELK Stack.

3.1 Introduction

ELK Stack was a collection of the following open-source products:

- **Elasticsearch**
- **Logstash**
- **Kibana**

Elasticsearch is an open source, full-text search, and analysis engine, based on the Apache Lucene search engine. Logstash is a log aggregator that collects data from various input sources, executes different transformations and enhancements and then ships the data to various supported output destinations. Kibana is a visualization layer that works on top of Elasticsearch, providing users with the ability to analyze and visualize the data.

Together, these different components are most commonly used for monitoring, troubleshooting, and securing IT environments. Logstash takes care of data collection and processing, Elasticsearch indexes and stores the data, and Kibana provides a user interface for querying the data and visualizing it.

3.2 Architecture

The below architecture provides a comprehensive solution for handling all the required facets:
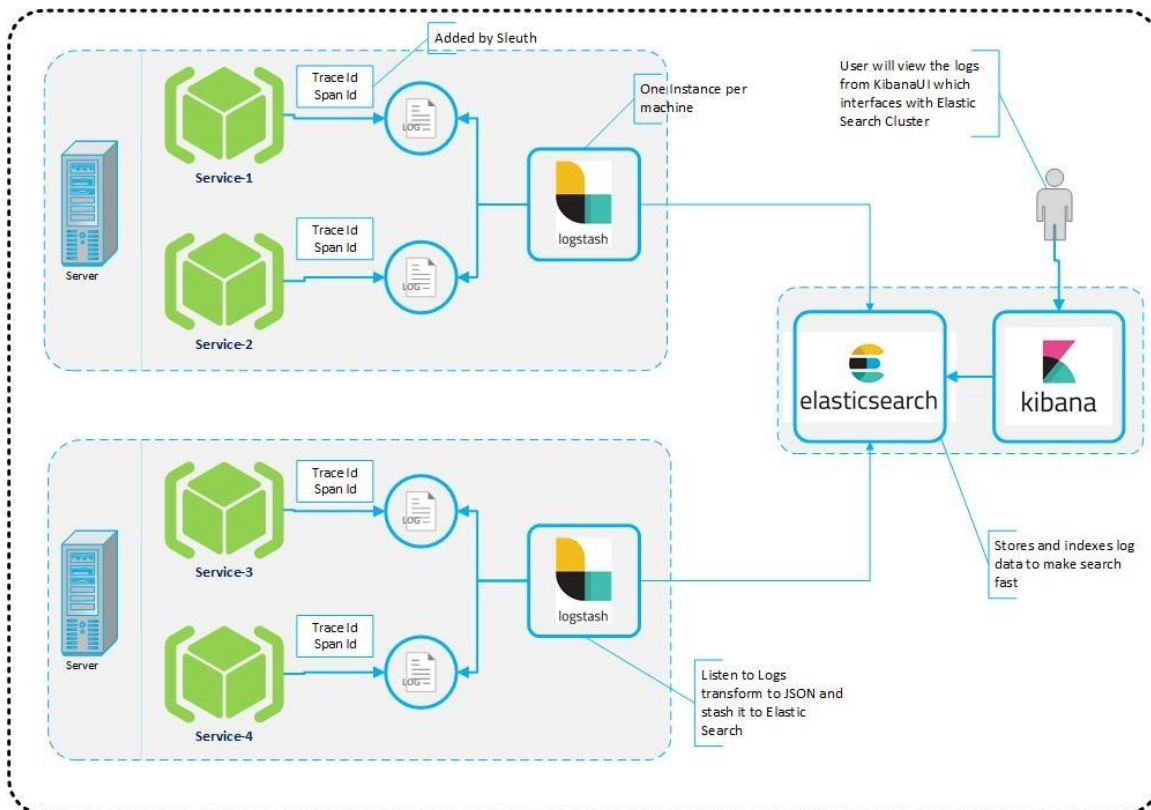
Spring cloud Sleuth also provides additional functionality to keep a trace of the application calls by providing us with a way to create intermediate logging events. Thus, the Spring Cloud Sleuth dependency must be added to applications.

3.3 Setting up ELK

Perform the following steps:

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 12: 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: 5601.

3.3.1 Steps to run ELK

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 Kibana to point the running instance of elastic search in kibana.yml file.

3. Configuration of Logstash consists of the following steps:

a) **Input**- This configuration is required to provide the log file location for the Logstash to read from.

b) **Filter**- Filters in Logstash is basically used to control or format the read operation (Line by line or Bulk read)

c) **Output**- In this section, provide the running elastic search instance to send the data for persisting.

Figure 13: 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"]
  }
}
```

3.3.1.1 Start ElasticSearch

1. Go to Elasticsearch root folder, and use nohup to start the Elasticsearch process as follows:

```
> nohup ./bin/elasticsearch
```

3.3.1.2 Setup Logstash and Start

1. Create a new **logstash.conf** file that provides the required file parsing and integration to Elasticsearch.

logstatsh.conf:

```
#Point to the application logs
input {
  file {
    type => "java"
    path => "/scratch/app/work_area/app_logs/*.log"
    codec => multiline {
      pattern => "^%{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{TIME}.*"
      negate => "true"
      what => "previous"
    }
```

```
  }
}
#Provide the parsing logic to transform logs into JSON
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"]
    }
  }

  #Grokking Spring Boot's default log format
  grok {
    match => [ "message",
               "(?<timestamp>%{YEAR}-%{MONTHNUM}-%{MONTHDAY}
%{TIME})  %{LOGLEVEL:level} %{NUMBER:pid} --- \[(?<thread>[A-Za-z0-
9-]+)\] [A-Za-z0-9.]*\.(?<class>[A-Za-z0-
9#_]+)\s*:\s+(?<logmessage>.*)",
               "message",
               "(?<timestamp>%{YEAR}-%{MONTHNUM}-%{MONTHDAY}
%{TIME})  %{LOGLEVEL:level} %{NUMBER:pid} --- .+?
:\s+(?<logmessage>.*)"
               ]
  }
   # pattern matching logback pattern
   grok {
         match =>
{ "message" => "%{TIMESTAMP_ISO8601:timestamp}\s+%{LOGLEVEL:severity
}\s+\[%{DATA:service},%{DATA:trace},%{DATA:span},%{DATA:exportable}\
]\s+\[%{DATA:environment}\]\s+\[%{DATA:tenant}\]\s+\[%{DATA:user}\]\
s+\[%{DATA:branch}\]\s+%{DATA:pid}\s+---
\s+\[%{DATA:thread}\]\s+%{DATA:class}\s+:\s+%{GREEDYDATA:rest}" }
   }
  #Parsing out timestamps which are in timestamp field thanks to
previous grok section
  date {
    match => [ "timestamp" , "yyyy-MM-dd HH:mm:ss.SSS" ]
  }
}
#Ingest logs to Elasticsearch
output {
```

```
  elasticsearch { hosts => ["localhost:9200"] }
  stdout { codec => rubydebug }
}
```

2. Start Logstash process

```
>nohup ./bin/logstash -f logstash.conf
```

3.3.1.3 Setup Kibana and start

1. Navigate to the **kibana.yml** available under <kibana_setup_folder>/config and modify the file to include the below:

```
#Uncomment the below line and update the IP address to your host
machine IP.
server.host: "xx.xxx.xxx.xx"
#Provide the elasticsearch url. If this is running on the same
machine then you can use the below config as is
elasticsearch.url: "http://localhost:9200"
```

2. Start Kibana process using the below command:

```
>nohup ./bin/kibana
```

3.3.2 Accessing Kibana

The Kibana can be accessed as shown below:

Figure 14: Accessing Kibana
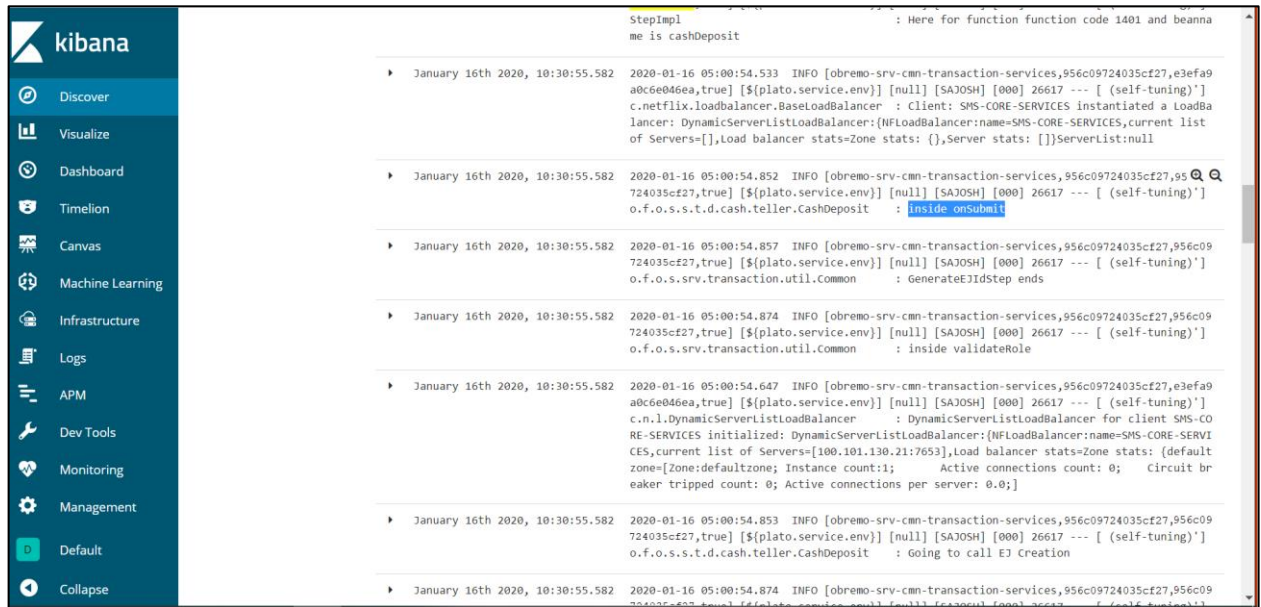
3.3.3 Steps to setup dynamic log levels in Oracle Banking Microservices Architecture services without restart

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 will be 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 15: 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 16: PLATO_LOGGER_PARAM_CONFIG

	ID	MODIFY_FIELD	PARAM_NAME	PARAM_VAL
1	3 N		LOG_PATH	C:\\Neoweblogic\\user_projects\\domains\\base_domain\\logs
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 will specify a dynamic logging path for the logging files to be stored. Changing this in runtime will change the location of the log files at runtime. If this value is not passed then by default, the LOG_PATH value will be 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** will append the current date into the log file name. Setting the value of this as **N** will not append the current date into the filename.

3.3.4 <u>Searching for Logs in Kibana</u>

The URL for searching logs in Kibana is
https://www.elastic.co/guide/en/kibana/current/search.html.

3.3.5 <u>How to Export Logs for Tickets</u>

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.

ORACLE®

4. Health Checks

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

4.1 Discovery Health Check

Discovery service shows the health status of all the registered services and their instances.

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

4.2 Actuator Health Indicator Endpoint:

4.2.1 Generic service

To check the health status of any service hit the following endpoint:
http://HOST:PORT/context_path/actuator/health (eg: 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

ORACLE®

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"
        }
    }
}
```

4.2.2 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

5. Troubleshooting Kafka Issues

This section describes the troubleshooting procedures for the Kafka issues.

5.1 Kafka Health

5.1.1 Verifying Kafka Health

Run the below command and verify:

*$ **netstat -tlnp | grep :9092** (9092 is default port of kafka)*

5.1.2 Verify Zookeeper Health

Kafka instance will not start if Zookeeper is not yet started. Run the below command and verify:

$ netstat -tlnp | grep :2181   (2181 is default port of zookeeper)

tcp6   0   0 :::2181   :::   LISTEN   19936/java*

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.

5.2 Prometheus and Grafana

5.2.1 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/.

5.2.2 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_lag.

    https://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.

5.2.3 <u>Grafana Setup</u>

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:

1. Click on the Grafana logo to open the sidebar.

2. Click **Data Sources** in the sidebar.

3. Choose **Add New**.

4. Select **Prometheus** as the data source.

5. Click **Add** to test the connection and to save the new data source.

5.2.4 <u>Prometheus Metrics</u>

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

ORACLE®

6. Troubleshooting Flyway issues

This section describes the troubleshooting procedures for the flyway issues.

6.1 <u>Failed Migrations</u>

6.1.1 <u>Success Column Verification</u>

Perform the following steps for the success column verification:

1. Check the *flyway_schema_history* table to identify the migration record with *success* column as '0'.

2. Delete the record with status as '0'.

3. Restart deployment.

6.1.2 <u>Migration Checksum Mismatch for a Version</u>

Perform the following steps:

1. Make sure that the flyway script is not manually updated before deployment.

2. If yes, then replace with original and restart deployment.

6.1.3 <u>Placeholder errors</u>

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.

ORACLE®