ACIT3855 - 2025.10 - Final exam

Thursday 17 April, 13:00 to 16:00

New service: anomaly detection

You have been tasked with adding a new feature to your system: anomaly detection. Your platform needs to be able to track any events that have potentially out of range numerical values.

Preliminary work

Create a new document. You will use this document throughout the assignment - including screenshots and short comments on your code / results. It is suggested that you empty the queue and database of your microservices platform before you start working on the assignment.

It is recommended that you read through all the instructions before you start working on the assignment.

Every time the instructions request you to take a screenshot, you must add it to your submission document, even if the instructions do not specify it explicitly.

1. Identify issues

You need to identify two anomalies for your system: one for each event type. This could be a **numerical** value in the event that is too high (i.e., above a configurable threshold) or too low (i.e., below a configurable threshold). You will then create a new service to detect these anomalies. This service will be called *Anomaly Detector* (`anomaly_detector` as its service name).

In the document, write a few sentences explaining which anomalies you are going to detect, and the possible thresholds. For example:

```
My anomaly service will detect all blood pressure events that are too low (for example, systolic pressure under 90). It will also detect heart beat events where the heart beat is too high (for example, above 140).
```

2. Service definition

Requirements

The `anomaly_detector` service must meet the following requirements:

- The service finds anomalies when the `PUT` endpoint is accessed (similar to the `consistency_check` service in assingment 2).
- If an event has an anomaly, a record is added to a JSON file managed by the Anomaly Detector service. The JSON file contains a list of records. Each record is an anomaly, and must contain:
    - Event's unique ID (NOT the trace ID, but the ID received from the client)
    - Event's Trace ID
    - Type of event
    - Description of the anomaly (must include the value detected and the threshold value exceeded)

Endpoint: update anomalies

The `PUT /update` endpoint:

- reads from the Kafka queue (from the beginning, every time)
- finds events with anomalies
- updates the JSON datastore from scratch with anomaly data (= the JSON file is overwritten every time the endpoint is accessed)

The endpoint returns an HTTP response that contains the number of anomalies detected in the queue. See the OpenAPI YAML specification for more details.

Endpoint: return anomalies

The `GET /anomalies` endpoint defined in the OpenAPI spec returns a list of anomalies. The endpoint allows to filter by event type (i.e. show anomalies for "EVENT1" only, or "EVENT2" only) using a parameter in the query string (URL).

- If the event type provided is invalid, return 400.
- If the event type is not provided, show all anomalies.
    - The event type is optional in the OpenAPI spec.
    - Make sure you use a default value for the argument (use the following signature for your endpoint function: `def get_anomalies(event_type=None)`).
    - If the event type is not provided in the URL, the function will use the default value provided in the signature.
- If there are no anomalies to return, return an empty response with code `204`.
- If there are issues with the anomaly datastore (missing or invalid data), return `404`.

See the OpenAPI specification for details.

Service configuration

The following must be configurable using **a configuration file**:

- Kafka connection - hostname and port
- filepath to the JSON file datastore

The thresholds for each of the two anomalies must be configured **using environment variables**.

For example:

- `HEART_BEAT_MAX=140`
- `BLOOD_PRESSURE_MIN=90`

> The service takes its settings from two different sources: a configuration file, and environment variables.

Logging

At minimum, the following shall be logged:

- on startup of the service: log the threshold values for anomalies (`INFO`)
- When accessing the "update" endpoint (`DEBUG`)
- When an anomaly is detected and added to the JSON file, including the value detected and threshold exceeded (`DEBUG`)

- When the "update" endpoint has completed the anomaly detection. Make sure you include how long the service took to retrieve the anomalies (`INFO`).
- When a `GET /anomalies` request is received and the response returned (`DEBUG`)

Integration

The service shall be fully integrated into your Docker Compose deployment including:

- Centralized Logging
- Centralized Configurations
- Volume for the data store
- Correct service dependencies
- Behind the NGINX reverse proxy

4. Screenshots

For all the following, **only use jMeter when required**. Otherwise, use an HTTP client like Bruno or Postman where we can see the request body (JSON) and response. Make sure the URL of the request is visible too (you should be using your VM).

MAKE SURE YOUR SCREENSHOTS SHOW THE CURRENT DATE/TIME ON YOUR COMPUTER (CLOCK).

Initial situation

Take a screenshot of your dashboard. It must show:

- analyzer stats
- processing stats

Take a screenshot of the `/anomalies` endpoint. Make sure there are no anomalies returned.

Send events

Using an HTTP client such as Bruno or Postman, send **ONE** event of **EACH TYPE** to your platform. Make sure the event data does not have anomalies. Take screenshots of each request and response.

Take a screenshot of your dashboard. You should have 2 events (1 of each type). Take a screenshot of the `/anomalies` endpoint. There should be no anomalies returned.

Send anomalies

Send **ONE** event of your **FIRST TYPE** (= 1 request). Make sure the event data **DOES** have an anomaly. Take a screenshot of the request and response. Take a screenshot of your dashboard. Take a screenshot of the

`/anomalies` endpoint. You should have **ONE** anomaly for your first type of event.

Send more anomalies

Send **TWO** events of your **SECOND TYPE** (= 2 requests). Make sure the data **DOES** have an anomaly for both requests. Take a screenshot of each request and responses. Take a screenshot of your dashboard. Take a screenshot of the `/anomalies` endpoint. You should now have 3 anomalies: 1 for your first type, 2 for your second type.

Send more events

Using jMeter, send a large number of events to your platform (at least 20). Make sure jMeter makes requests that are both valid and bearing anomalies. Wait for all events to be processed. Then take a screenshot of your dashboard, and take a screenshot of the `/anomalies` endpoint.

Bonus marks: dashboard updates and automated deployment

Dashboard

Update the Dashboard UI to show a list of anomalies for each event type in a different "area" of the page (i.e. make two requests, each for one event type). Each list must show at least the event's ID, Trace ID and anomaly description. You can display the raw JSON data from the endpoint above, but it would be nice to extract and present the information in a legible way. For example (note yours will show data relevant to your system):

```
Blood Pressure Anomalies: None
Heart Rate Anomalies:
- Detected: 150; too high (threshold 140)
 * Event ID: 055cee1e-2c4a-4624-a754-9d6e33ec3b53
 * Trace ID: 43289048957345435
```

Take screenshots of your dashboard, with and without anomalies detected and displayed.

Automated deployment

Make changes to your automated deployment strategy to make sure the new service can be easily deployed. Briefly explain the changes you made in the document. Make sure you attach the files to your submission.

Submission

Generate a PDF file from your submission document.

Generate a ZIP file containing:

- files for your Anomaly Detection service
  - folder with the code
  - data and configuration files (JSON datastore, configuration files)
- your submission document (PDF)
- other supporting files, including:
  - Docker Compose file
- For bonus marks: Ansible playbook(s) / deployment files

Submit the ZIP file to the Learning Hub.

Grading

Item	Marks
Anomaly description	**2**
Service: update endpoint	**4 total**
-- service consumes the Kafka queue	1
-- service detects anomalies	1
-- JSON records management	1
-- logging	1
Service: get endpoint	**3 total**
-- anomalies returned and status codes	1
-- filtering by event type	1
-- logging	1
Anomaly detection: service integration	**5 total**
-- runs on your VM	1
-- Docker and Docker Compose	1
-- persistent volumes	1
-- configuration settings	1
-- NGINX reverse proxy (lab 11)	1
Screenshots	**4**

Item	Marks
TOTAL	**18 marks**