Summary of Proposed Solutions to Fix Prometheus Metrics Issue

This document outlines the proposed solutions to address the discrepancies between Splunk and Prometheus metrics and to resolve the UTF-8 encoding issue encountered in the Go-Link Metrics service. The solutions are presented in order of precedence, with the first solution to try listed first. Each solution includes a clear explanation and implementation steps.

# 1. Verify Splunk Query Consistency

Explanation:

* Ensure that the Splunk query used in the Go-Link Metrics service is identical to the one used manually. Any slight difference could result in varying counts. Make sure that the time window for querying Splunk in Go-Link Metrics is correctly set to match the 30-day window used in Prometheus.

Implementation:

1. 1. Review and compare the Splunk query in the Go-Link Metrics code against the manually executed query.
2. 2. Adjust the query if any discrepancies are found to ensure consistency.

# 2. Check Prometheus Query Configuration

Explanation:

* Validate the Prometheus query used (`increase(golink\_redirect\_hit\_awsconsole\_total[30d])`) to ensure it accurately reflects the data collected from Splunk. Consider whether this query is appropriate for the data type being handled. If the data retrieved from Splunk is not a counter, but an aggregated count, consider using a gauge metric or directly querying the stored value.

Implementation:

1. 1. Test the Prometheus query with different time windows to ensure accuracy.
2. 2. Adjust the query logic or metric type in Prometheus if the current setup is not appropriate for the data format.

# 3. Inspect and Adjust Go-Link Metrics Data Processing

Explanation:

* Review the data aggregation and processing logic within the Go-Link Metrics service to ensure it correctly handles and forwards the data retrieved from Splunk to Prometheus. Check for any potential data transformations or errors that could result in discrepancies between Splunk and Prometheus values.

Implementation:

1. 1. Debug and log data processing steps within Go-Link Metrics, especially during the transformation and forwarding of data to Prometheus.
2. 2. Adjust the processing logic if any inconsistencies are found.

# 4. UTF-8 Validation in ScalarMetric Class

Explanation:

* Validate UTF-8 encoding within the ScalarMetric class to ensure that all metric keys and short keys are properly encoded right from the point of creation. This prevents any invalid characters from propagating through the system.

Implementation:

1. 1. Add a `validateUtf8` method in the `ScalarMetric` class and use it to validate the `shortKey` and `metricKey` after sanitization.

Code Implementation:

private String validateUtf8(String value) {  
 if (value == null) {  
 return null;  
 }  
 byte[] bytes = value.getBytes(StandardCharsets.UTF\_8);  
 return new String(bytes, StandardCharsets.UTF\_8);  
}  
  
public void setMetricKey(String metricKey) {  
 this.metricKey = validateUtf8(metricKey.replaceAll("[\-.\*~@!$=]", SANITIZED\_CHARACTER));  
}  
  
public void setShortKey(String shortKey) {  
 this.shortKey = validateUtf8(shortKey);  
}

# 5. UTF-8 Validation in MetricsController Class

Explanation:

* Perform a final UTF-8 validation in the MetricsController before exposing data to Prometheus. This acts as a safeguard to ensure that all data being exposed is valid UTF-8.

Implementation:

1. 1. Add a `validateUtf8` method in the `MetricsController` class and apply it to all metric-related strings just before writing the response to Prometheus.

Code Implementation:

private String validateUtf8(String value) {  
 if (value == null) {  
 return null;  
 }  
 byte[] bytes = value.getBytes(StandardCharsets.UTF\_8);  
 return new String(bytes, StandardCharsets.UTF\_8);  
}  
  
@GetMapping(value = "/metrics")  
public void getMetrics(HttpServletResponse response) throws IOException {  
 response.setContentType(MediaType.TEXT\_PLAIN.toString());  
  
 StringBuilder metricsBuilder = new StringBuilder();  
  
 for (ScalarMetric sm : metricsRepository.getAll()) {  
 String metricKey = validateUtf8(sm.getMetricKey());  
 String shortKey = validateUtf8(sm.getShortKey());  
  
 String scalarEntry = String.format(  
 HELP\_TEXT + TYPE\_TEXT + METRIC\_TEXT,  
 metricKey, metricKey, metricKey, shortKey, sm.getCount()  
 );  
 metricsBuilder.append(scalarEntry);  
 }  
  
 writeResponse(response, metricsBuilder.toString());  
}

# 6. Sanitize Input Strings and Correct Regex in ScalarMetric Class

Explanation:

* Ensure that all metric keys are properly sanitized to remove or replace problematic characters. Adjust the regex used for sanitization to only replace specific characters instead of broadly replacing all non-whitespace characters.

Implementation:

1. 1. Update the sanitization regex in the ScalarMetric class to target specific characters.

Code Implementation:

static final String CHARACTERS\_TO\_SANITIZE = "-.\*~@!$=";  
static final String SANITIZED\_CHARACTER = "\_";  
  
public void setMetricKey(String metricKey) {  
 this.metricKey = validateUtf8(metricKey.replaceAll("[\-.\*~@!$=]", SANITIZED\_CHARACTER));  
}