Application Logging and Sending Alerts on Teams

Objective:

This document outlines the process of logging application events and sending alerts to Microsoft Teams. The logging process involves sending logs directly from the application to a specific endpoint, which is a virtual machine (VM) hosted on Google Cloud Platform (GCP). Within this VM, Fluentd is configured to collect application logs and forward them to Elastic Cloud. Subsequently, we set up alerts in Elastic Cloud to send error logs to a Microsoft Teams channel.

Process Overview:

1. **Python Logging Configuration:**
2. A Python logging module is utilized within the application to configure logging. Below is an example of Python code snippet:

```
# logger.py


import requests

import json

from datetime import datetime

import os


class Logger:

    def __init__(self, service,logs_url,log_index,transaction_id,user_id):

        self.service = service

        self.url = logs_url

        self.log_index = log_index

        self.user_id = user_id

        self.transaction_id = transaction_id


    def log(self, message,log_level='INFO', error_details=None, additional_data={}):

        obj = {

            'timestamp': datetime.now().isoformat(),

            'service': self.service,
            'instance': self.log_index,

            'userId': self.user_id,

            'transactionId': self.transaction_id,

            'logLevel': log_level,

            'message': message,

            'errorDetails': error_details,

            'additionalData': additional_data,

        }



        print(obj)

        try:

            response = requests.post(self.url, json=obj, headers={'Content-Type': 'application/json'})

            response.raise_for_status()

            print('Log successfully sent')

        except Exception as e:

            print(f'Error sending log: {e}')


if __name__ == '__main__':



    log_index = 'disearch-app'

    logger =
Logger(service='testing',logs_url='http://34.29.169.192:24224/myapp.logs',log_index=log_index,transaction_i
d='123',user_id='my_user')

    logger.log('logs from Hassan Bashir disearchmt-dev')
```

1. **Configuration of Logging in Application:**
   - Define a Logger class with methods to log messages and send them to the specified endpoint.
   - Configure the logger to include necessary details such as timestamp, service, log level, message, etc.
   - Instantiate the logger object with appropriate parameters like service name, endpoint URL, log index, transaction ID, and user ID.
2. **Fluentd Configuration:**

3. Fluentd is configured within a container running on the VM. Below is the Fluentd configuration file (fluent.conf):

```
# fluent.conf

<system>

  log_level info

</system>

# Input section for receiving logs via HTTP

<source>

  @type http

  port 24224

  bind 0.0.0.0

  format json

  body_size_limit 32m

  keepalive_timeout 10s

</source>



<filter **>

  @type record_transformer

  enable_ruby

 <record>

   timestamp ${record["timestamp"]}

   service ${record["service"]}

   logLevel ${record["logLevel"]}

   message ${record["message"]}

#   instance ${record["instance"]}

   instance ${record["instance"] || "default-instance"}

   userId ${record.dig("userId") || "N/A"}

   transactionId ${record.dig("transactionId") || "N/A"}
```

```
#   errorCode ${record.dig("errorDetails", "code") || "N/A"}

#   errorMessage ${record.dig("errorDetails", "message") || "N/A"}

#   errorCode ${record.dig("errorDetails").is_a?(Hash) && record.dig("errorDetails", "code").is_a?(String) ? record.dig("errorDetails", "code") : "N/A"}

#   errorMessage ${record.dig("errorDetails").is_a?(Hash) && record.dig("errorDetails", "message").is_a?(String) ? record.dig("errorDetails", "message") : "N/A"}

   errorCode ${record["errorDetails"].is_a?(Hash) && record["errorDetails"].key?("code") ? record["errorDetails"]["code"] : "N/A"}

   errorMessage ${record["errorDetails"].is_a?(Hash) && record["errorDetails"].key?("message") ? record["errorDetails"]["message"] : "N/A"}

   additionalData ${record["additionalData"].to_json || "{}"}

 </record>

  <record>

    instance ${record['instance'] || 'default-instance'}

  </record>

   remove_keys errorDetails

</filter>

# Buffer section for more reliable log forwarding

<match **>

  @type elasticsearch

  cloud_id "My-deployment:dXMtY2VudHJhbDEuZ2NwLmNsb3VkLmVzLmlvOjQ0MyQxYWJlNDQ4ZjBmN2I0NzUzYTFkZWMyYjhiMTVhOGUxNCRjYzk0N2NkYTRjNzM0ODFmYWRhZjNmOTE1OWU2Yzg4MQ=="

  cloud_auth "elastic:EWsTCrASie5n6OgQcbMEArtY"

  scheme http

  ssl_verify false

  logstash_format false

  enable_ruby true

  index_name "${instance}"
  include_tag_key true

  # Buffer settings for improved reliability

  <buffer tag, instance>

    @type file

    path /var/log/fluentd/buffer

    flush_interval 5s

    chunk_limit_size 100m

  </buffer>

  # TLS settings for secure connection to Elasticsearch Cloud

</match>

# Healthcheck endpoint (optional)

<source>

  @type monitor_agent

  bind 0.0.0.0

  port 24220

</source>
```

Fluentd is configured to listen on port 24224 for incoming logs in JSON format.

- Records are transformed as per requirements before forwarding to Elasticsearch.
- Elasticsearch is configured with appropriate cloud credentials for authentication.

1. **Sending Alerts to Microsoft Teams:**
   - Configure a webhook in Microsoft Teams to receive alerts.
   - Set up alerts in Elastic Cloud to trigger on error logs.
   - Configure the alert to use the previously created webhook URL for sending alerts to the Teams channel.
2. **Manual Verification:**
3. Execute the provided Python code to manually verify that logs are successfully sent to the specified endpoint.

Conclusion:

By following the outlined process, application logging is effectively managed, and alerts are sent to Microsoft Teams for timely response to error events. This ensures better monitoring and troubleshooting of applications hosted on Google Cloud Platform.

this is for my understanding:

process:

we have given LOG_INDEX as cloud run variable in frontend. in this index we gave the project id or every gcp project on which we need to get indices having logs. the index will go to the backend and from backend it will go to elastic cloud in index section. now we can create data view by using this index. And with that data view we can see logs in discover section.

And we can also set alert to send ERROR logs to teams chennel by selecting this dataveiw having indices in alerts section in elastic cloud. we also need to create the chennel on team and get the chennel connection using webhook and give that webhook url of team to elastic cloud in alert section.