**Automation of Daily Health Check for APIC Dashboard**

**Current Process**

The current process involves manually viewing the APIC dashboard in Splunk, filling out a table manually, and then adding this table to a larger email that is sent out to the team daily.

**Proposed Solution**

The proposed solution involves automating the process of generating the table from the APIC dashboard and sending it via email. This will be achieved through the following steps:

1. **Automate PDF Generation and Emailing**: Configure Splunk to automatically generate a PDF of the APIC dashboard and send it to a specified email address. Splunk has built-in functionality for scheduling reports and sending them via email.
2. **PDF Parsing and Table Generation**: Develop a Python program that will parse the received PDF, extract the necessary data, and generate a table. This program will use libraries such as PyPDF2 for PDF parsing and pandas for table creation.
3. **Automate Emailing of the Table**: The Python program will also automate the process of sending the generated table via email. This can be achieved using libraries such as smtplib for email sending.

**Detailed Steps**

**Automate PDF Generation and Emailing**

1. In Splunk, navigate to the APIC dashboard.
2. Click on “Export” and select “Schedule PDF Delivery”.
3. Configure the schedule according to your needs (e.g., daily) and specify the recipient email address.

**PDF Parsing and Table Generation**

1. Use the PyPDF2 library to read the PDF file from the email.
2. Extract the necessary data from the PDF.
3. Use the pandas library to create a table from the extracted data.

**Automate Emailing of the Table**

1. Use the smtplib library to send the generated table via email.
2. Configure the email settings (SMTP server, port, sender email, recipient email, etc.).
3. Attach the generated table to the email and send it.

**Conclusion**

By automating the process of generating and emailing the APIC dashboard report, we can save time and reduce the possibility of human error. The Python program will handle the task of parsing the PDF and generating the table, which can then be easily copied and pasted into the daily email. This solution will streamline the daily health check process and make it more efficient.

Sample Code Below:

import logging

import PyPDF2

import pandas as pd

import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

# Set up logging

logging.basicConfig(filename='app.log', filemode='w', format='%(name)s - %(levelname)s - %(message)s')

def parse\_pdf(file\_path):

    try:

        pdf\_file\_obj = open(file\_path, 'rb')

        pdf\_reader = PyPDF2.PdfFileReader(pdf\_file\_obj)

        page\_obj = pdf\_reader.getPage(0)

        text = page\_obj.extractText()

        pdf\_file\_obj.close()

        return text

    except Exception as e:

        logging.error("Exception occurred during PDF parsing", exc\_info=True)

def create\_dataframe(text):

    try:

        data = {'Column1': ['Value1', 'Value2', 'Value3'],

                'Column2': ['Value4', 'Value5', 'Value6']}

        df = pd.DataFrame(data)

        return df

    except Exception as e:

        logging.error("Exception occurred during DataFrame creation", exc\_info=True)

def send\_email(df, sender\_email, sender\_password, receiver\_email):

    try:

        msg = MIMEMultipart()

        msg['From'] = sender\_email

        msg['To'] = receiver\_email

        msg['Subject'] = 'APIC Dashboard Report'

        html = """\

        <html>

        <head></head>

        <body>

        {0}

        </body>

        </html>

        """.format(df.to\_html())

        msg.attach(MIMEText(html, 'html'))

        server = smtplib.SMTP('smtp.gmail.com', 587)

        server.starttls()

        server.login(sender\_email, sender\_password)

        server.send\_message(msg)

        server.quit()

    except Exception as e:

        logging.error("Exception occurred during email sending", exc\_info=True)

def main():

    try:

        text = parse\_pdf('path\_to\_your\_pdf.pdf')

        df = create\_dataframe(text)

        send\_email(df, 'your\_email@gmail.com', 'your\_password', 'receiver\_email@gmail.com')

    except Exception as e:

        logging.error("Exception occurred in main function", exc\_info=True)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Please incorporate all these solutions into this document :

1. **PDF Parsing (Current Method)**:
   * **Tools**: Python libraries such as PyPDF2 or PDFMiner.six can be used to extract text from the PDF.
   * **Process**: The script would need to open the PDF file, read the contents, and then parse the text to extract the necessary data. This might involve searching for specific keywords or patterns in the text.
2. **Web Scraping**:
   * **Tools**: Python libraries such as BeautifulSoup or Scrapy can be used to scrape web pages.
   * **Process**: The script would need to send a request to the URL of the APIC dashboard, receive the HTML of the page, and then parse the HTML to extract the necessary data. This might involve searching for specific HTML tags or attributes.
3. **Database Querying**:
   * **Tools**: Python libraries such as SQLAlchemy or psycopg2 can be used to interact with databases. The specific library to use would depend on the type of database (e.g., PostgreSQL, MySQL, SQLite).
   * **Process**: The script would need to establish a connection to the database, send a SQL query to retrieve the necessary data, and then process the returned data as needed.