

Automation of Daily Health Check for APIC Dashboard

# Introduction

The current process for the daily health check of the APIC dashboard is manual and involves several steps: manually viewing the APIC dashboard in Splunk, filling out a table manually, adding this table to a larger email, and sending the email to the team daily. This process is time-consuming and prone to human error. Automating this process would save time and improve accuracy.

# Proposed Automation Solutions

## 1. PDF Parsing (Current Method)

Tools: PyPDF2 or PDFMiner.six.

Process:

* Open the PDF file that contains the APIC dashboard report.
* Read the contents of the PDF file.
* Use PDF parsing tools to extract the necessary data from the text within the PDF.
* Identify specific keywords or patterns in the text to extract the relevant data.
* Generate a table using the extracted data.
* Send the generated table via email to the specified recipients.

### PDF Parsing Code Sample

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:  
 # Example: Splitting text into columns based on a delimiter (customize as needed)  
 rows = text.split('\n')  
 data = [row.split(' ') for row in rows if row.strip()]  
 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 = f"""  
 <html>  
 <head></head>  
 <body>  
 {df.to\_html()}  
 </body>  
 </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()

## 2. Web Scraping

Tools: BeautifulSoup or Scrapy.

Process:

* Send an HTTP request to the URL of the APIC dashboard.
* Receive the HTML content of the dashboard page.
* Use web scraping tools to parse the HTML content.
* Identify specific HTML tags or attributes to extract the relevant data.
* Generate a table using the extracted data.
* Send the generated table via email to the specified recipients.

### Web Scraping Code Sample

import requests  
from bs4 import BeautifulSoup  
import pandas as pd  
import smtplib  
from email.mime.multipart import MIMEMultipart  
from email.mime.text import MIMEText  
import logging  
  
# Set up logging  
logging.basicConfig(filename='app.log', filemode='w', format='%(name)s - %(levelname)s - %(message)s')  
  
def scrape\_dashboard(url):  
 try:  
 response = requests.get(url)  
 response.raise\_for\_status() # Check for HTTP errors  
 soup = BeautifulSoup(response.content, 'html.parser')  
 return soup  
 except Exception as e:  
 logging.error("Exception occurred during web scraping", exc\_info=True)  
  
def create\_dataframe(soup):  
 try:  
 # Example: Extract table data from HTML (customize as needed)  
 table = soup.find('table')  
 headers = [th.text.strip() for th in table.find\_all('th')]  
 rows = []  
 for tr in table.find\_all('tr'):  
 cells = [td.text.strip() for td in tr.find\_all('td')]  
 if cells:  
 rows.append(cells)  
 df = pd.DataFrame(rows, columns=headers)  
 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 = f"""  
 <html>  
 <head></head>  
 <body>  
 {df.to\_html()}  
 </body>  
 </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:  
 url = 'http://your\_apic\_dashboard\_url.com'  
 soup = scrape\_dashboard(url)  
 df = create\_dataframe(soup)  
 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()

## 3. Database Querying

Tools: SQLAlchemy or psycopg2.

Process:

* Establish a connection to the database where the APIC dashboard data is stored.
* Send a SQL query to retrieve the necessary data.
* Process the data returned by the SQL query.
* Generate a table using the processed data.
* Send the generated table via email to the specified recipients.

# Conclusion

Each of the proposed solutions offers a different approach to automating the daily health check for the APIC dashboard. The choice of solution will depend on the specific requirements and constraints, such as access to the data, the format of the data, and the resources available for implementation.

- PDF Parsing is suitable if existing PDF reports are reliable and consistent.

- Web Scraping is useful for accessing live data directly from the dashboard, but requires careful handling of HTML content and compliance with site policies.

- Database Querying offers the most direct and efficient access to structured data, provided that database access is available and properly managed.

By implementing any of these solutions, the daily health check process can be significantly streamlined, reducing manual effort and the risk of errors.