create a python script to generate two separate excel based reports

Here is an example Python script that uses the **pandas** and **openpyxl** libraries to generate two separate Excel-based reports:

import pandas as pd

from openpyxl import Workbook

# Create a pandas DataFrame with sample data

data = {'Name': ['John', 'Jane', 'Bob', 'Samantha'],

'Age': [25, 32, 45, 27],

'Country': ['USA', 'Canada', 'UK', 'Australia']}

df = pd.DataFrame(data)

# Create the first report

report1 = df[['Name', 'Age']]

# Create a new Excel workbook and add the report to the first sheet

wb = Workbook()

ws = wb.active

for row in dataframe\_to\_rows(report1, index=False, header=True):

ws.append(row)

# Save the report as 'report1.xlsx'

wb.save('report1.xlsx')

# Create the second report

report2 = df[['Name', 'Country']]

# Create a new Excel workbook and add the report to the first sheet

wb = Workbook()

ws = wb.active

for row in dataframe\_to\_rows(report2, index=False, header=True):

ws.append(row)

# Save the report as 'report2.xlsx'

wb.save('report2.xlsx')

The above script creates a pandas DataFrame with sample data and then uses the **pandas** library to create two separate reports with different columns. Then it uses the **openpyxl** library to create a new Excel workbook, add each report to a separate sheet and save it as two separate files 'report1.xlsx' and 'report2.xlsx' respectively. Please note that this script is just an example and would need to be adapted to your specific use case and data.

create a python script to generate two separate excel based reports, one for DS and one for DI

Here is an example Python script that uses the **pandas** and **openpyxl** libraries to generate two separate Excel-based reports, one for "DS" and one for "DI":

import pandas as pd

from openpyxl import Workbook

# Read the data from a CSV file

df = pd.read\_csv('data.csv')

# Create the first report for "DS"

ds\_report = df[df['Department'] == 'DS']

# Create a new Excel workbook and add the report to the first sheet

wb = Workbook()

ws = wb.active

for row in dataframe\_to\_rows(ds\_report, index=False, header=True):

ws.append(row)

# Save the report as 'DS\_report.xlsx'

wb.save('DS\_report.xlsx')

# Create the second report for "DI"

di\_report = df[df['Department'] == 'DI']

# Create a new Excel workbook and add the report to the first sheet

wb = Workbook()

ws = wb.active

for row in dataframe\_to\_rows(di\_report, index=False, header=True):

ws.append(row)

# Save the report as 'DI\_report.xlsx'

wb.save('DI\_report.xlsx')

This script reads a csv file 'data.csv' and creates a pandas DataFrame. Then it filters the DataFrame based on the column 'Department' and creates two separate reports, one for 'DS' and one for 'DI' using the **pandas** library. Then it uses the **openpyxl** library to create a new Excel workbook, add each report to a separate sheet and save it as two separate files 'DS\_report.xlsx' and 'DI\_report.xlsx' respectively. Please note that this script is just an example and would need to be adapted to your specific use case, data and file format.

develop a end to end method to automate enabling/disabling of alarm actions in vrops when host is disconnected, maintenance mode, not responding with yml

Here is an example of an end-to-end method to automate the enabling and disabling of alarm actions in vROps when a host is disconnected, in maintenance mode, or not responding using a YAML file:

1. Create a YAML file that contains the configuration for the alarm actions, including the conditions under which the actions should be enabled or disabled. For example:

alarm\_name: Host Disconnected

enabled\_conditions:

- event\_type: Host disconnected

- event\_type: Host in maintenance mode

disabled\_conditions:

- event\_type: Host responding

1. Create a Python script that uses the **pyvmidc** library to connect to vROps and the **pyyaml** library to read the YAML file. The script should first read the YAML file and extract the configuration for the alarm actions.

import yaml

with open("alarm\_config.yml", 'r') as stream:

config = yaml.safe\_load(stream)

1. Use the **pyvmidc** library to connect to vROps and retrieve the current status of the alarm.

from pyVmomi import vmodl

from pyVim.connect import SmartConnect, Disconnect

from pyVim.task import WaitForTask

si = SmartConnect(host=vrops\_host, user=vrops\_user, pwd=vrops\_password)

alarm\_manager = si.content.alarmManager

alarm = alarm\_manager.GetAlarm(Name=config['alarm\_name'])

1. Compare the current status of the alarm with the conditions specified in the YAML file and enable or disable the alarm actions as necessary.

enabled = True

for condition in config['enabled\_conditions']:

if condition not in alarm.info.enabled:

enabled = False

break

for condition in config['disabled\_conditions']:

if condition in alarm.info.enabled:

enabled = False

break

if enabled:

alarm.EnableActions()

else:

alarm.DisableActions()

1. Schedule the Python script to run at a specific time or interval using a task scheduler like Windows Task Scheduler or cron on Linux.

Please note that this script is just an example and would need to be adapted to your specific use case and vROps configuration. Also, you might need to install pyvmidc library before running the script.

develop a end to end method to automate enabling/disabling of alarm actions in vrops when host is disconnected, maintenance mode, not responding with Ansible playbook example

Here is an example of an end-to-end method to automate the enabling and disabling of alarm actions in vROps using Ansible Playbook:

1. Create a YAML file that contains the configuration for the alarm actions, including the conditions under which the actions should be enabled or disabled. For example

alarm\_name: Host Disconnected

enabled\_conditions:

- event\_type: Host disconnected

- event\_type: Host in maintenance mode

disabled\_conditions:

- event\_type: Host responding

1. Create an Ansible playbook that uses the **vrops\_alarm** module to connect to vROps and enable or disable the alarm actions.

- name: Enable/Disable Alarm Actions

hosts: localhost

vars\_files:

- alarm\_config.yml

tasks:

- name: Enable/disable alarm actions

vrops\_alarm:

vrops\_hostname: "{{ vrops\_host }}"

vrops\_username: "{{ vrops\_user }}"

vrops\_password: "{{ vrops\_password }}"

alarm\_name: "{{ alarm\_name }}"

enabled\_conditions: "{{ enabled\_conditions }}"

disabled\_conditions: "{{ disabled\_conditions }}"

state: present

1. Create a variable file called **vars.yml** with the vROps host, user, and password information.

vrops\_host: vrops.example.com

vrops\_user: admin

vrops\_password: Passw0rd

Run the Ansible playbook with the command **ansible-playbook -i "localhost," -c local playbook.yml --extra-vars "@vars.yml"**.

Schedule the Ansible Playbook to run at a specific time or interval using a task scheduler like Windows Task Scheduler or cron on Linux.

Please note that this playbook is just an example and would need to be adapted to your specific use case and vROps configuration. Also, you might need to install vrops\_alarm Ansible module and the vROps SDK for python before running the playbook.