script to download:

python_hyper-v_topology_grapher.py

PY: Hyper-V topology mapper

Introduction

The following code retrieves information about virtual switches (VMSwitches) and virtual machine network adapters (VMNetworkAdapters) in Hyper-V. It then builds a graph, where the nodes are the virtual switches and VMs, and the edges represent the connections between these elements. The whole thing is visualised using the Matplotlib library.

Programme content

The programme consists of several functions that perform various tasks:

1. get_vmswitches function

```
def get_vmswitches():
    """
    Use PowerShell to get Hyper-V virtual switches as JSON.
    """
    cmd = ["powershell", "-Command", "Get-VMSwitch | ConvertTo-Json"]
    proc = subprocess.run(cmd, capture_output=True, text=True)
    try:
        switches = json.loads(proc.stdout)
    except json.JSONDecodeError as e:
        print("Error decoding JSON for switches:", e)
        print("PowerShell output was:", proc.stdout)
        switches = []
    # Ensure the data is a list
    if isinstance(switches, dict):
        switches = [switches]
    return switches
```

This function uses PowerShell to retrieve information about the virtual switches on the Hyper-V system and then converts the result into JSON format. If the JSON decoding fails, it prints an error message and returns an empty list. If the data is not in list form, the function converts it to a list.

2. get_vmnetworkadapters function

```
def get_vmnetworkadapters():
```

```
    """
    Use PowerShell to get Hyper-V VM network adapters as JSON.
    This version pipes all VMs to Get-VMNetworkAdapter to ensure we get
data.
    """
    # Updated command to get network adapters from all VMs
    cmd = ["powershell", "-Command", "Get-VM | Get-VMNetworkAdapter |
ConvertTo-Json"]
    proc = subprocess.run(cmd, capture_output=True, text=True)
    if proc.stderr:
        print("PowerShell error output:", proc.stderr)
    try:
        adapters = json.loads(proc.stdout)
    except json.JSONDecodeError as e:
        print("Error decoding JSON for adapters:", e)
        print("PowerShell output was:", proc.stdout)
        adapters = []
    # Ensure the data is a list
    if isinstance(adapters, dict):
        adapters = [adapters]
    return adapters
```

This function works similarly to the previous one, but instead of virtual switches, it fetches data about the network adapters of the virtual machines in Hyper-V. In case of errors during JSON decoding, it prints an error message and returns an empty list.

3. build_graph function

```
def build_graph(switches, adapters):
    """
    Build a graph with virtual switches as one type of node and VMs as
another.
    An edge represents a connection between a VM and a virtual switch.
    """
    G = nx.Graph()

    # Add virtual switch nodes
    for sw in switches:
        sw_name = sw.get("Name")
        if sw_name:
            G.add_node(sw_name, type="switch")

    # Add VM nodes and edges based on network adapter connections
    for adapter in adapters:
        vm_name = adapter.get("VMName")
        switch_name = adapter.get("SwitchName")
        if vm_name:
            G.add_node(vm_name, type="vm")
            # Create an edge if the adapter is connected to a switch
            if switch_name:
```

```
                G.add_edge(vm_name, switch_name)
    return G
```

This function creates a graph using the NetworkX library. It adds nodes to the graph representing virtual switches (of type 'switch') and virtual machines (of type 'vm'). The edges of the graph are created between the VMs and the virtual switches if a network adapter is connected to the switch in question.

4 The draw_graph function

```
def draw_graph(G):
    """
    Draw the graph using matplotlib.
    Switches are drawn as squares (blue) and VMs as circles (green).
    """
    pos = nx.spring_layout(G, k=1.0, iterations=100, seed=42)  # For
consistent layout

    # Separate nodes by type for custom styling
    switch_nodes = [n for n, d in G.nodes(data=True) if d.get("type") ==
"switch"]
    vm_nodes = [n for n, d in G.nodes(data=True) if d.get("type") == "vm"]

    plt.figure(figsize=(10, 8))
    nx.draw_networkx_nodes(G, pos, nodelist=switch_nodes,
node_color='lightblue', node_shape='s',
                           node_size=1500, label="Virtual Switch")
    nx.draw_networkx_nodes(G, pos, nodelist=vm_nodes,
node_color='lightgreen', node_shape='o',
                           node_size=1500, label="Virtual Machine")
    nx.draw_networkx_edges(G, pos)
    nx.draw_networkx_labels(G, pos, font_size=10)

    plt.title("Hyper-V Topology: Virtual Switches & VMs")
    plt.axis('off')
    plt.legend(scatterpoints=1, labelspacing=1.5, handletextpad=1.0,
borderaxespad=1.0)
    plt.tight_layout()
    plt.show()
```

This function is responsible for visualising the graph using the Matplotlib library. Nodes representing virtual switches are drawn as blue squares and nodes representing virtual machines are drawn as green circles. The edges of the graph represent the connections between the switches and the virtual machines.

5 The main function

```
def main():
```

```
    print("Extracting Hyper-V virtual switches...")
    switches = get_vmswitches()
    print(f"Found {len(switches)} switch(es).")

    print("Extracting Hyper-V VM network adapters...")
    adapters = get_vmnetworkadapters()
    print(f"Found {len(adapters)} network adapter(s).")

    if not switches and not adapters:
        print("No data retrieved. Ensure Hyper-V is installed and you have
proper permissions.")
        return

    graph = build_graph(switches, adapters)
    draw_graph(graph)
```

The `main` function starts the process of collecting data about the virtual switches and network adapters of the virtual machines, and then builds and draws a graph representing this data. This function is the entry point of the programme and is started when the programme is executed directly.

6 Program call

```
if __name__ == "__main__":
    main()
```

This is the standard way of running a program in Python. If the script is run directly, the `main()` function is called, which starts the whole process.

Summary

The code is used to retrieve data about the virtual switches and network adapters of virtual machines in Hyper-V, build a graph representing these elements, and then visualise this graph using the Matplotlib library. It uses PowerShell to retrieve the data and NetworkX to manipulate the graph.