CodiLime Sp. z o.o.
CodiLime, Inc.
www.codilime.com
contact@codilime.com

Autocon2 WS:A1 - Utilize GenAI for Network Troubleshooting

Instructions for workshop participants

Get familiar with the setup

The setup used during the workshop consists of two main components:

- The Net-Chat Assistant running as a docker container
- Containerlab environment simulating network topology

Network topology overview

The diagram below presents the network topology to be used during the workshop. This is a simple ISP topology with PE, CE routers and PCs emulating customer endpoints. To keep things simple we only used BGP and OSPF here. No MPLS/LDP/ISIS/RSVP will be used during the hands-on. The entire environment will come installed and ready to use on VMs running in a public cloud. To avoid any licensing issues, we decided to use open-source NOS such as Alpine & FRR for PEs and VYOS for CEs.

Note: Although full administrative access is granted to each device participating in the topology, we strongly recommend that you do not change existing configurations unless explicitly asked to do so.

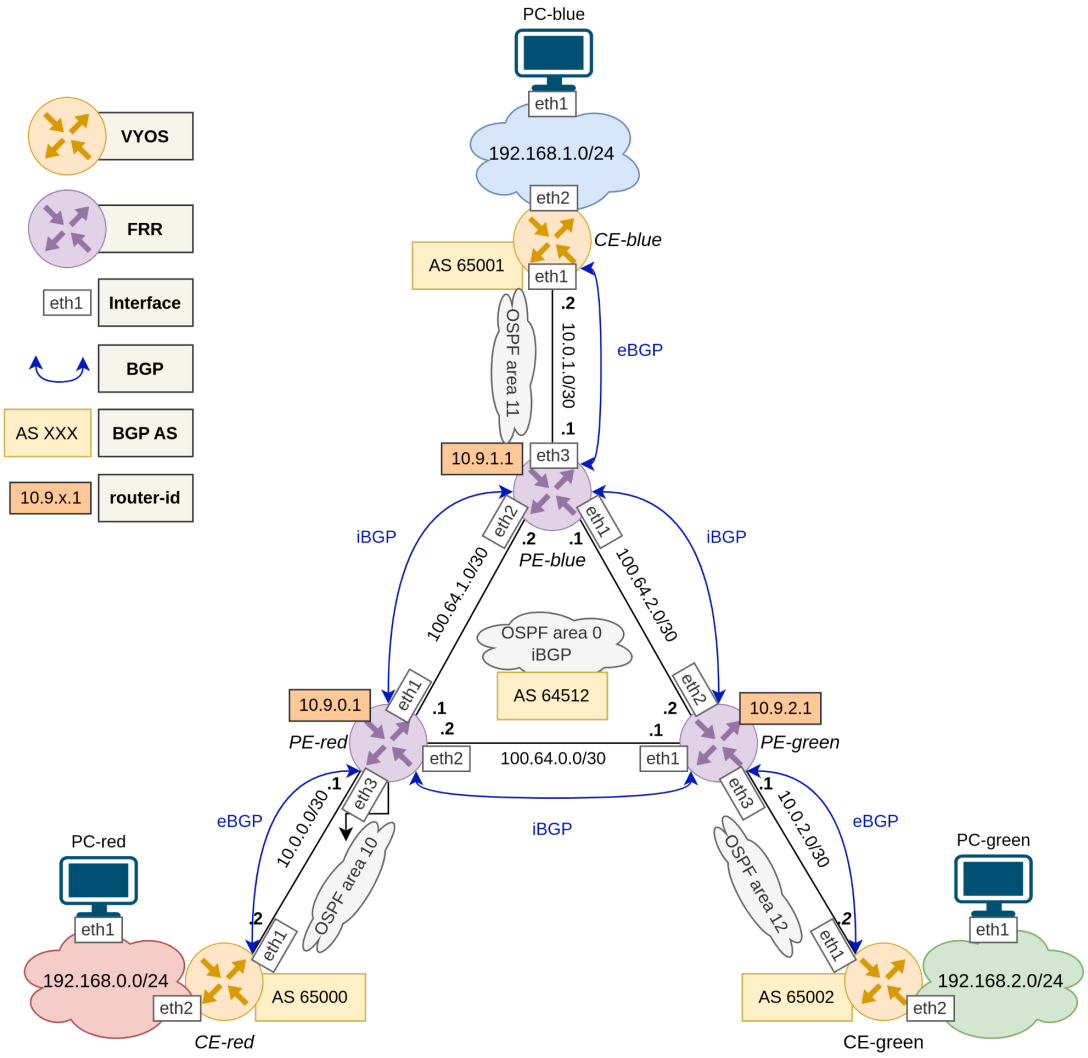


Figure 1: Network topology overview

Access to VMs

Each workshop participant will have access to a dedicated VM in the cloud. Hostname and login credentials will be given during the workshop. Access is provided via SSH protocol. However, since the Net-Chat Assistant is available via web

interface and listening on its loopback on `8501` TCP port, an SSH tunnel is needed to gain access.

SSH session with port forwarding using PuTTY

Here's how to set it up in PuTTY (vm_nb and password will be given to you at the beginning of the workshop) :

1. **Open PuTTY**: Launch the PuTTY application.
2. **Session Settings**:
    - In the left pane, go to **Session**.
    - Enter the **Host Name (or IP address)** as `vm-<vm_nb>.ac2.codilime.com`.
    - Set the **Port** to `22` (default for SSH).
    - Under **Connection type**, select **SSH**.
3. **Define the SSH Tunnel (Port Forwarding)**:
    - In the left pane, expand **Connection** > **SSH** and then select **Tunnels**.
    - In the **Source port** field, enter `8501` (the local port).
    - In the **Destination** field, enter `127.0.0.1:8501` (the address and port on the remote server).
    - Ensure **Local** is selected (for local port forwarding).
    - Click **Add**. The forwarded port should now appear in the **Forwarded ports** list as `L8501 127.0.0.1:8501`.
4. **Save the Session** (optional):
    - Return to the **Session** category at the top of the left pane.
    - Under **Saved Sessions**, type a name (e.g., `VM SSH Tunnel`) and click **Save** to easily reuse this configuration in the future.
5. **Connect**:
    - Click **Open** to start the SSH connection.

- A terminal window will open, prompting you to enter the password for `codi` user.

SSH session with port forwarding using Linux terminal (alternative)

Here's how to set it up in Linux (vm_nb and password will be given to you at the beginning of the workshop) :

1. Open a terminal and type the following command:

```
# ssh -L 8501:127.0.0.1:8501 codi@vm-<vm_nb>.ac2.codilime.com
```

2. Enter the password for codi user for your VM

Note: Please always keep at least one SSH session open to ensure that the web UI is accessible at all times.

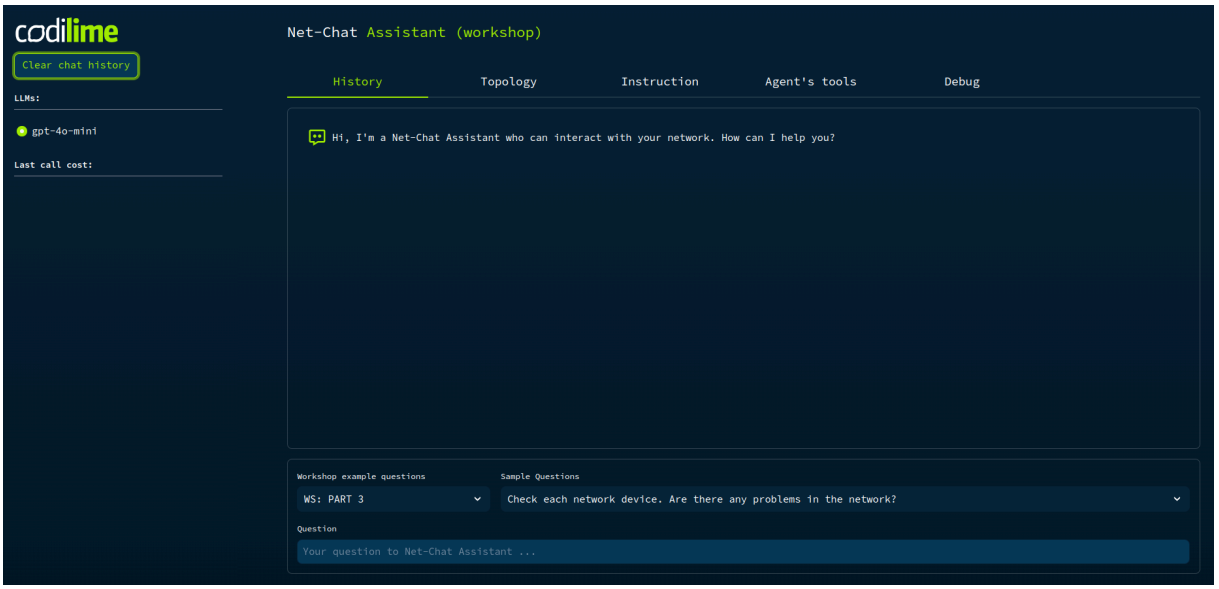
Web access

Having SSH tunneling up and running, you can access the web interface of the Net-Chat Assistant:

1. Open your browser (Chrome/Firefox) and paste the link:

   `http://localhost:8501`

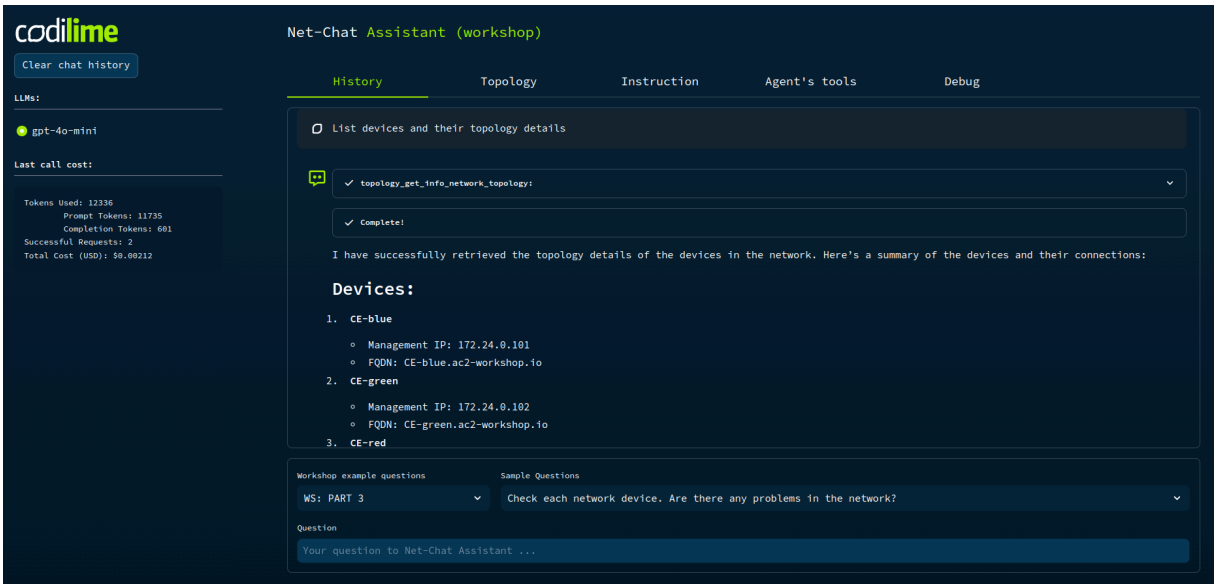
2. You should see the Net-Chat Assistant web application.



3. Ask a question to the Net-Chat Assistant (in order to check if everything is working as intended):

   ```
   List devices and their topology details
   ```

4. You should see a response from the Net-Chat Assistant



What the VM contains

The following chapter is mostly optional; however, you should familiarize yourself with the section about accessing Containerlab devices, as this knowledge will be needed later on.

[OPTIONAL] Directory contents

In order to get familiar with the VM content, you can also use the `ls` command to list the directories that will be used during the workshop:

- **clab/** - directory with a network model used by containerlab
- **clab_backup/** - backup directory with a network model used by containerlab
- **src/** - directory with the Net-Chat Assistant python code
- **ac2.env** - env variables with OPENAI_API_KEY and credentials to network devices
- **docker-compose.yml** - docker compose config for convenient restarts of the Net-Chat Assistant
- **scripts/failure_*.sh** - scripts for simulating failures in the network during the workshop
- **scripts/recovery_*.sh** - scripts for network recovery after each hands-on part
- **scripts/fix_code_*.sh** - scripts for proper code changes in the Net-Chat Assistant
- **scripts/reset_net_chat.sh** - script to reset the Net-Chat Assistant (redeploy it from scratch)
- **scripts/restart_net_chat.sh** - script to restart the Net-Chat Assistant (after code change)

- **scripts/reset_containerlab.sh** - script to reset the Containerlab network (redeploy it from scratch)

Note: Please do not execute any scripts or change the contents of any file at this time.

[OPTIONAL] Containers are running on VM

To check what containers are running on the VM you also execute:

```
# sudo docker ps --format "table {{.Names}}\t{{.Status}}"
```

This command should give you the following output:

Unset

```
# sudo docker ps --format "table {{.Names}}\t{{.Status}}"
NAMES       STATUS
# docker container running the Net-Chat Assistant
net-chat    Up 10 minutes
# docker containers running network devices (Containerlab)
PE-blue     Up 8 hours
PE-red      Up 8 hours
PE-green    Up 8 hours
CE-blue     Up 8 hours
CE-green    Up 8 hours
CE-red      Up 8 hours
# docker containers running endpoint devices (Containerlab)
PC-red      Up 8 hours
PC-green    Up 8 hours
PC-blue     Up 8 hours
```

Access to network devices via SSH

Using the credentials provided in the tables below, you can gain access to specific devices used in the network topology. Access is provided via SSH from the VM. You can use either their hostnames or IP addresses:

PE devices:

Hostname	IP	Login	Password
PE-red	172.24.0.10	root	root
PE-blue	172.24.0.11	root	root
PE-green	172.24.0.12	root	root

CE devices:

Hostname	IP	Login	Password
CE-red	172.24.0.100	vyos	vyos
CE-blue	172.24.0.101	vyos	vyos
CE-green	172.24.0.102	vyos	vyos

PC endpoints:

Hostname	IP	Login	Password
PC-red	172.24.0.200	root	root
PC-blue	172.24.0.201	root	root
PC-green	172.24.0.202	root	root

Examples:

1. **$ ssh -t root@PE-red vtysh**
   PE-red# show version
   PE-red# exit
2. **$ ssh vyos@CE-blue**
   vyos@CE-blue:~$ show version
   vyos@CE-blue:~$ exit
3. **$ ssh root@172.24.0.202**
   PC-green:~# cat /etc/os-release
   PC-green:~# exit

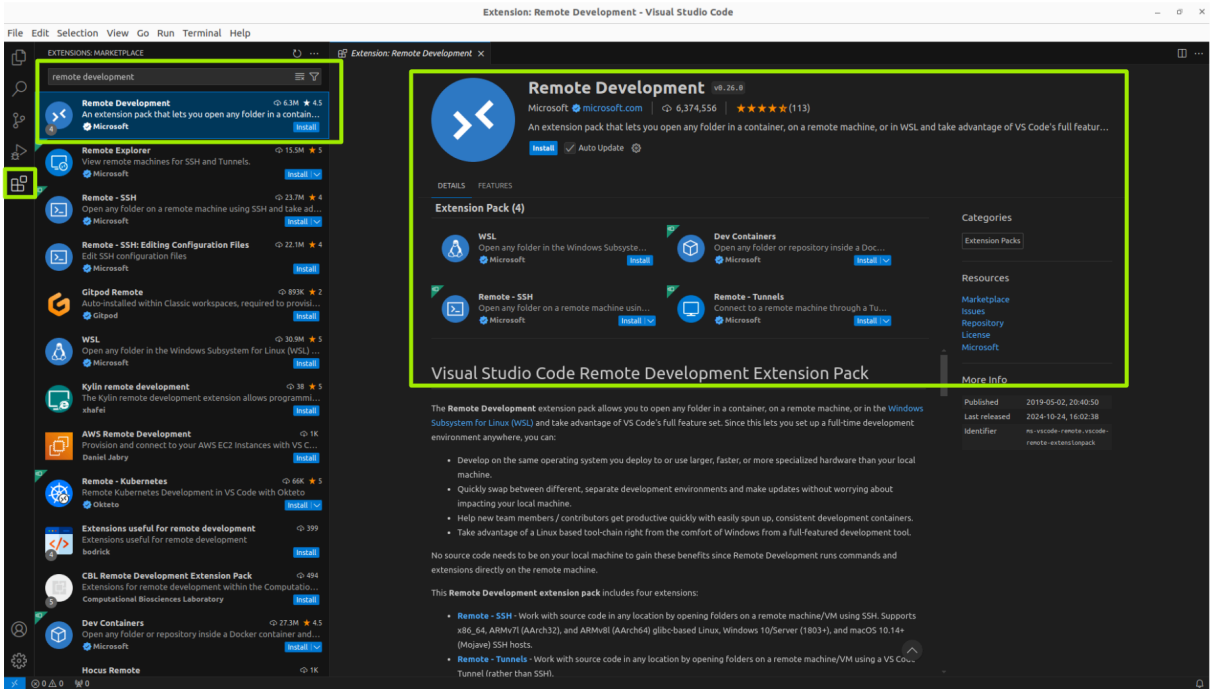
Note: Please do not execute any configuration commands or change device states at this time.

[OPTIONAL] VS Code configuration steps

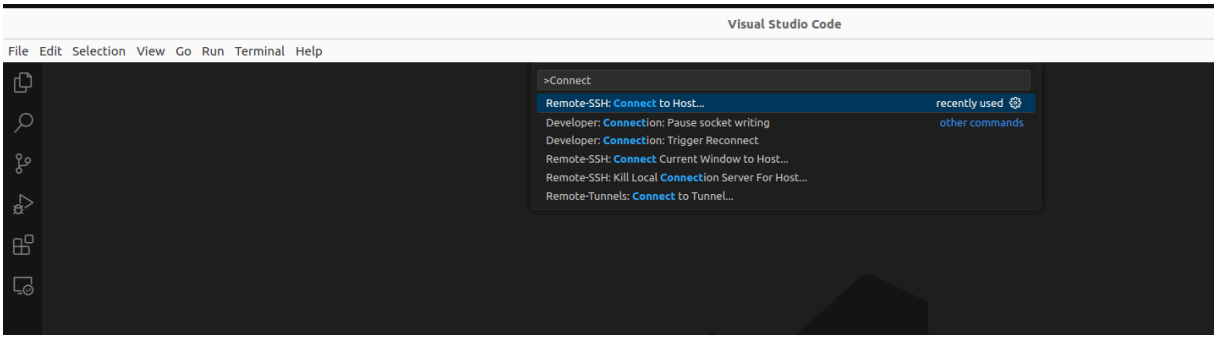
Note: If you decide not to use VS code, please skip this chapter.

To configure your VS Code to access Python code on a remote VM, please take the following steps :

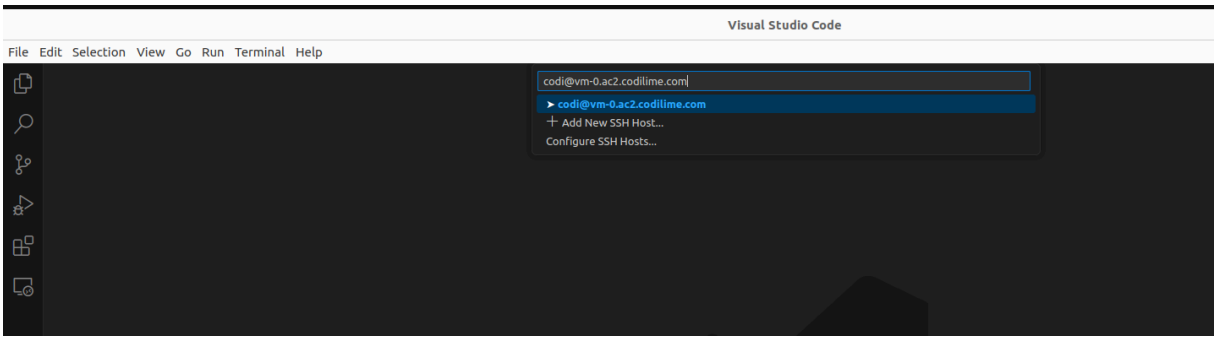
1. Open VS Code
   a. If you haven't done it already, install the "Remote Development" extension from Microsoft to VS Code



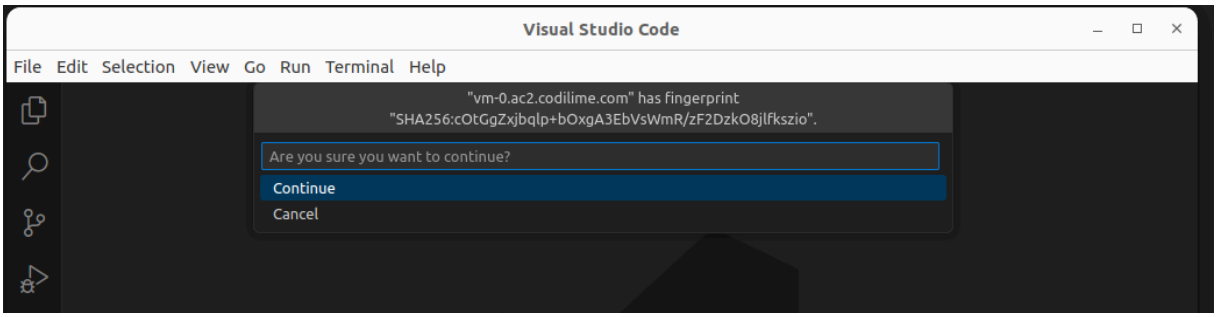
2. Click on menu View>Command Pallette… (Ctrl+Shift+P), find and select the Remote SSH: Connect to Host… action and click it



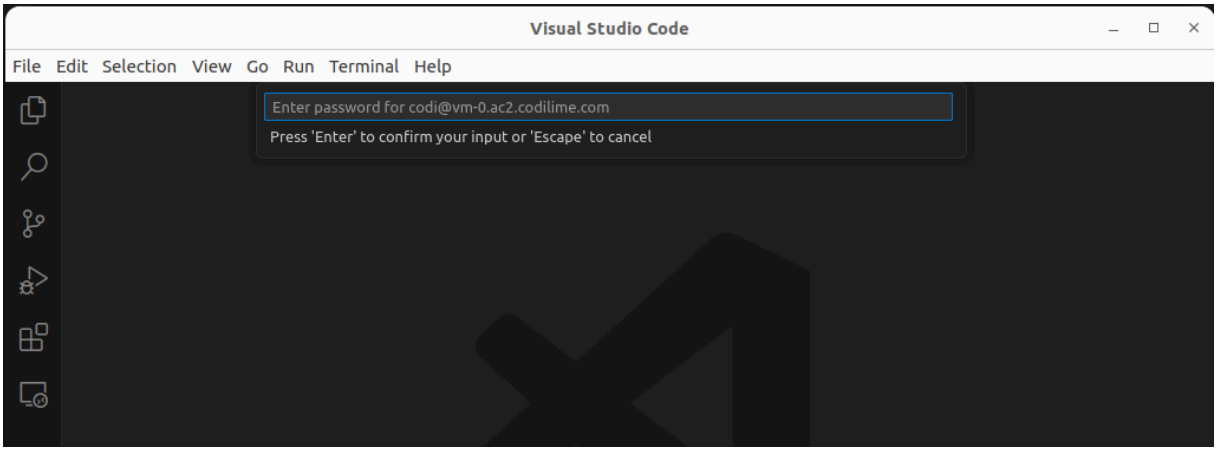
3. Enter connection details (user@host) to your VM: `codi@vm-<vm_nb>.ac2.codilime.com`. Be sure to provide the user name “codi” at the beginning. Press enter.



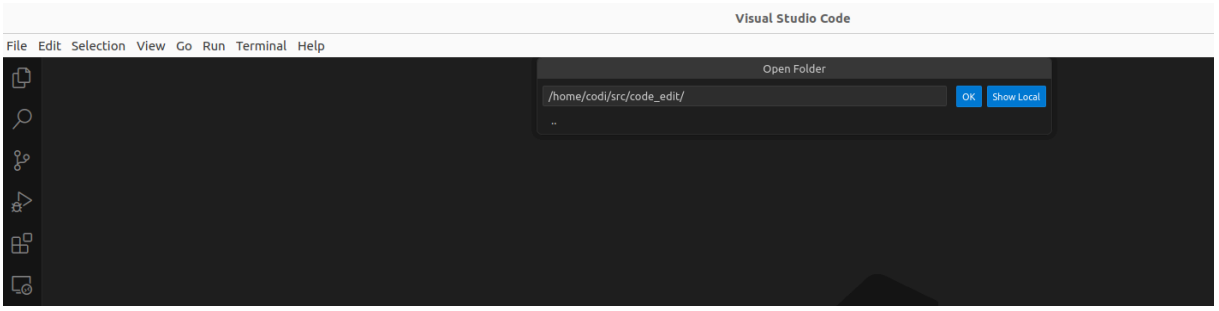
4. Click Continue if you see the following window:



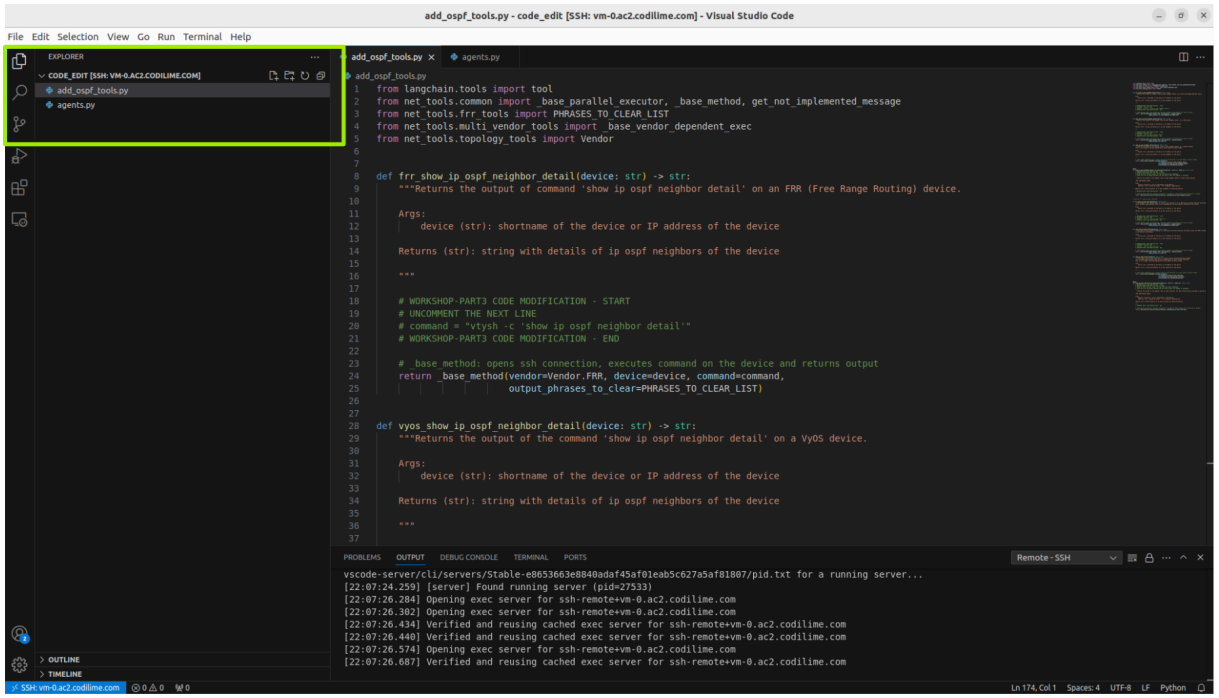
5. Enter your password



6. After successful connection click on menu File>Open Folder... (Ctrl+K), select remote folder /home/codi/src/code_edit/ and click OK.



7. You should see 2 python files to be edited in WS PART-3.



Note: Please do not edit any files yet.

Hands-on

Part 1 - Obtaining information about the network

Objectives

- Get familiar with the Net-Chat Assistant UI and underlying network topology
- Use AI to obtain network inventory information

Steps

Query the Net-Chat Assistant

- Switch to the Net-Chat Assistant in the browser and ask the following questions
- Observe the results and verify them against the network topology described in this guide
- If the output is not satisfactory, try repeating the same query or rephrasing it
- Example queries:

```
List devices and their details

List devices and their topological details

List links and their topological details

List networking devices with their platform and OS
version

List networking devices with their platform and OS
version in tabular format

How are CE devices connected?

Check connectivity from CE to PE devices, present the
results in tabular form
```

```
Check connectivity between PC endpoints, present the
results in tabular form

Check WWW connectivity from PC endpoints to
www.google.com, present the results in tabular form

Check WWW connectivity from PC endpoints to PC-blue,
present the results in tabular form

Get basic data on interfaces on PE devices, present the
results in tabular form

Get the ip addressing for loopback interfaces on PE-red

Is there a direct connection between PE-red and PE-green?

Is there a direct connection between PE-red and PE-green?
If so, present its details.

Summarize the connection details between PE-red and
PE-green and provide IP addresses on those interfaces
```

Part 2 - Failure scenario I

Objectives

- Introduce a link failure between PE-red and PE-green
- Debug a link failure between PE devices

Steps

Introduce a failure

Execute the following script in the VM shell:

```
$ sudo ~/scripts/failure_2.sh
```

```
INFO[0000] Executed command "ip link set eth2 down" on the
node "PE-red". stdout:
```

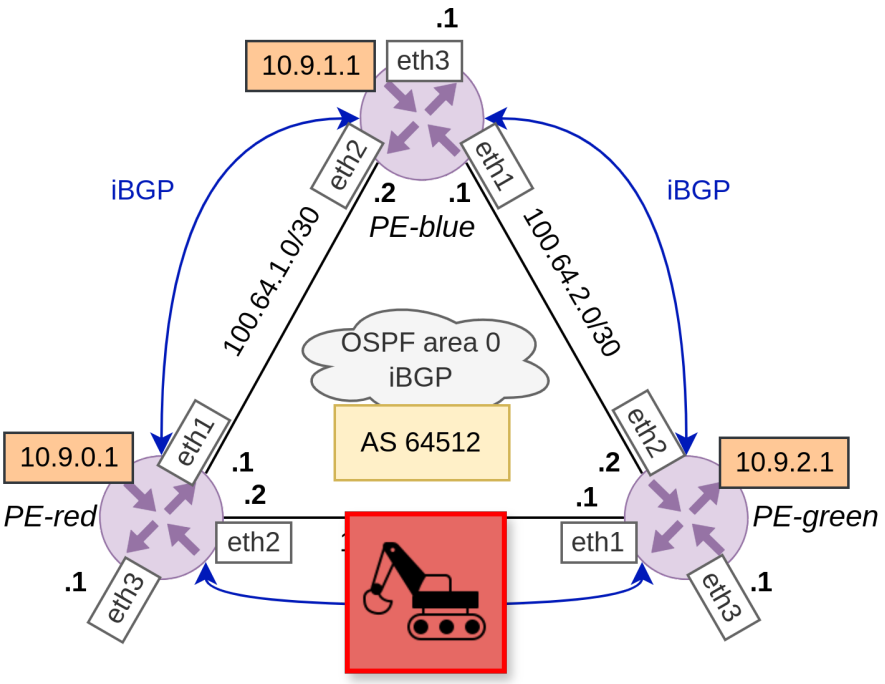


Figure 2: Simulated failure in Part 2

The failure_2.sh script simulates a network link failure by bringing down the eth2 interface on the PE-red node. This failure can disrupt connectivity for any route or traffic that relies on eth2, potentially causing network congestion or rerouting as alternative paths are utilized, impacting network performance and availability.

Check the impact of the failure on the network

To check if there is indeed a problem between devices, issue the following commands and analyze the output:

```
$ ssh -t root@PE-red vtysh
PE-red# show interface eth2
Interface eth2 is down
 Link ups:       0    last: (never)
 Link downs:     1    last: 2024/11/12 09:23:44.87
```

```
$ ssh -t root@PE-green vtysh
PE-green# show interface eth1
Interface eth1 is up, line protocol is down
 Link ups:       1    last: 2024/11/12 08:46:58.39
 Link downs:     1    last: 2024/11/12 09:23:44.87
```

Use the Net-Chat Assistant to debug issues

- Switch to the Net-Chat Assistant in the browser and ask the following questions
- Observe the results and verify them against the network topology affected by the failure

- If the output is not satisfactory, try repeating the same query or rephrasing it
- Example queries:

```
Are there any problems in the network? Check each network
device.

Which interfaces with IP addresses are down? Check each
network device.

Check all network devices. Which interfaces with IP
addresses are down?

Show me more details about the eth1 interface on the
PE-green device.

Show me more details about the eth1 interface on the
PE-green device. Outline what is suspicious.

What is suspicious about the PE-red device?
```

Restore the network to a normal state

Run the following script:

```
$ sudo ~/scripts/recovery_2.sh
```

```
INFO[0000] Executed command "ip link set eth2 up" on the node
"PE-red". stdout:
```

Part 3 - Adding OSPF support & failure scenario II

Objectives

- Add OSPF support to the Net-Chat Assistant
- Debug a unidirectional link failure

Steps

Check the Net-Chat Assistant for OSPF support

Using the web UI, ask the Net-Chat Assistant for OSPF support:

```
Do you have any tools related to OSPF protocol?
```

Get ready for source code editing on the remote VM

There are three options available:

1. You can use a remote terminal editor such as: *vim, nano, emacs, mc*
2. You can use the VS Code with Remote Development Extension (see VS Code configuration steps for details)
3. You can also skip code editing and apply pre-prepared code by executing dedicated scripts. In such a case, go directly to the *In case of problems* section.

Edit the first Python file

1. Open the following file in your favorite editor:

```
~/src/code_edit/add_ospf_tools.py
```

2. Find appropriately marked places in the Python code and provide valid **`show ip ospf neighbor detail`** commands for FRR (PE) and VyOS (CE) devices:

Python
```python
def frr_show_ip_ospf_neighbor_detail(device: str) -> str:

    # WORKSHOP-PART3 CODE MODIFICATION - START
    # UNCOMMENT THE NEXT LINE
    # command = "vtysh -c 'show ip ospf neighbor detail'"
    # WORKSHOP-PART3 CODE MODIFICATION - END

    # _base_method: opens ssh connection, executes command on the device and returns output
    return _base_method(vendor=Vendor.FRR, device=device, command=command,
                        output_phrases_to_clear=PHRASES_TO_CLEAR_LIST)
```

Python
```python
def vyos_show_ip_ospf_neighbor_detail(device: str) -> str:

    # WORKSHOP-PART3 CODE MODIFICATION - START
    # UNCOMMENT THE NEXT LINE
    # command = "show ip ospf neighbor detail"
    # WORKSHOP-PART3 CODE MODIFICATION - END

    # _base_method: opens ssh connection, executes command on the device and returns output
    return _base_method(vendor=Vendor.VYOS, device=device, command=command,
                        output_phrases_to_clear=[])
```

3. Find the appropriately marked place in the Python code and provide the **description** in docstring for the tool delivering output for commands **`show ip ospf neighbor detail`** for the list of devices:

Python
```python
@tool
def show_ip_ospf_neighbor_detail_for_many_devices(devices: list[str], empty_str: str) -> str:
    # WORKSHOP-PART3 CODE MODIFICATION - START
    # PROPOSE DESCRIPTION WHAT THE TOOL IS DOING
    # BASED ON THIS LLM WILL SELECT THE TOOL FOR USER QUERY ANSWERING
    # START WITH THE FOLLOWING SENTENCE AND ADD DETAILS WHAT THE COMMAND IS RETURNING
```

```
    """Returns the output of the command 'show ip ospf neighbor detail' on many network
devices.

<PUT YOUR DETAILS HERE>

Args:
    devices (list[str]): list of shortnames of the devices
    empty_str (str): should be always '' for langchain compatibility

Returns (str): Returns details of ip ospf neighbors on specified devices.
"""
# WORKSHOP-PART3 CODE MODIFICATION - END

# _base_parallel_executor: parallel execution of scripts for many devices and collection of
outputs
return _base_parallel_executor(devices=devices, method=show_ip_ospf_neighbor_detail)
```

Below is our description, but you can provide your own proposal.

Unset

```
"""
Returns the output of the command 'show ip ospf neighbor detail' on many network
    devices.

It provides comprehensive information about OSPF (Open Shortest Path First) neighbors
    on each network device.
It lists neighboring routers involved in the OSPF process, detailing their IP
    addresses, router IDs, and the interfaces used for the adjacency.
Key OSPF metrics, such as the neighbor state (e.g., Full, Init) and the neighbor role
    (e.g., DROther),are displayed to help understand the relationship between
    routers.
The output also includes priority settings for elections, Dead timers (which indicate
    how long until a neighbor is declared unreachable), and BFD (Bidirectional
    Forwarding Detection) status to track neighbor health.
Additional information, like OSPF database synchronization lists, area details, and
    interface MTU, ensures the consistency and reliability of routing data.
For troubleshooting purposes, this command is essential for verifying correct OSPF
    neighbor relationships and resolving adjacency issues.
If Graceful Restart Helper is enabled, details about non-disruptive OSPF restarts are
    also included.


Args:
   devices (list[str]): list of shortnames of the devices
   empty_str (str): should be always '' for langchain compatibility

Returns (str): Returns details of ip ospf neighbors on specified devices.
"""
```

4. Find the appropriately marked places in the Python code and provide valid **`show ip ospf interface`** commands for FRR and VyOS devices

Python
```python
def frr_show_ip_ospf_interface(device: str) -> str:

    # WORKSHOP-PART3 CODE MODIFICATION - START
    # UNCOMMENT THE NEXT LINE
    # command = "vtysh -c 'show ip ospf interface'"
    # WORKSHOP-PART3 CODE MODIFICATION - END

    # _base_method: opens ssh connection, executes command on the device and returns
     output
    return _base_method(vendor=Vendor.FRR, device=device, command=command,
                        output_phrases_to_clear=PHRASES_TO_CLEAR_LIST)
```

Python
```python
def vyos_show_ip_ospf_interface(device: str) -> str:

    # WORKSHOP-PART3 CODE MODIFICATION - START
    # UNCOMMENT THE NEXT LINE
    # command = "show ip ospf interface"
    # WORKSHOP-PART3 CODE MODIFICATION - END

    # _base_method: opens ssh connection, executes command on the device and returns
     output
    return _base_method(vendor=Vendor.VYOS, device=device, command=command,
                        output_phrases_to_clear=[])
```

5. Find the appropriately marked place in the Python code and provide a **description** in docstring for the tool delivering output for commands **`show ip ospf interface`** for the list of devices:

Python
```python
@tool
def show_ip_ospf_interface_for_many_devices(devices: list[str], empty_str: str) -> str:
    # WORKSHOP-PART3 CODE MODIFICATION - START
    # PROPOSE DESCRIPTION WHAT THE TOOL IS DOING
    # BASED ON THIS LLM WILL SELECT THE TOOL FOR USER QUERY ANSWERING
    # START WITH THE FOLLOWING SENTENCE AND ADD DETAILS WHAT THE COMMAND IS RETURNING

    """Returns the output of the command 'show ip ospf interface' for each network device provided
     in devices argument.
```

```
    <PUT YOUR DETAILS HERE>

    Args:
        devices (list[str]): list of shortnames of the devices
        empty_str (str): should be always '' for langchain compatibility

    Returns (str): Returns details of ip ospf interface on specified devices.

    """
    # WORKSHOP-PART3 CODE MODIFICATION - END

     # _base_parallel_executor: parallel execution of scripts for many devices and collection of
     outputs
    return _base_parallel_executor(devices=devices, method=show_ip_ospf_interface)
```

Below is our description, but you can provide your own proposal.

Unset

```
"""
Returns the output of the command 'show ip ospf interface' for each network device
    provided in devices argument.

The output provides detailed insights into the OSPF (Open Shortest Path First)
    configuration for each network interface.
This includes the status of the interface (e.g., whether it's up or down), the
    interface-specific details such as MTU size, bandwidth, and IP address, along
    with the OSPF area and Router ID associated with each interface.
Additionally, the command highlights important OSPF parameters like network type,
    cost,Hello and Dead timers, and the current neighbor status, including adjacency
    details.
It also provides information about multicast group memberships and protocols like
    BFD, used to monitor the health of OSPF sessions. This information is essential
    for understanding OSPF operation and behavior on network interfaces.

Args:
   devices (list[str]): list of shortnames of the devices
   empty_str (str): should be always '' for langchain compatibility

Returns (str): Returns details of ip ospf interface on specified devices.

"""
```

6. Save the file.

Edit the second Python file

1. Open the following file in your favorite editor:

`~/src/code_edit/agents.py`

2. import the prepared **`add_ospf_tools`** module

Python

```
from langchain.agents import AgentExecutor
from langchain.agents import StructuredChatAgent

from net_agents.custom_prompt import PREFIX, SUFFIX, FORMAT_INSTRUCTIONS,
StructuredChatOutputParser
from net_tools import multi_vendor_tools, topology_tools



# WORKSHOP-PART3 CODE MODIFICATION - START
# UNCOMMENT THE NEXT LINE
# from net_tools import add_ospf_tools
# WORKSHOP-PART3 CODE MODIFICATION - END
```

3. Add the prepared tools to the agent's tool_list

Python

```
tool_list = [
    multi_vendor_tools.run_ping_for_many_devices,
    multi_vendor_tools.run_curl_for_many_devices,
    multi_vendor_tools.show_interfaces_details_for_many_devices,
    multi_vendor_tools.show_bgp_summary_for_many_devices,
    multi_vendor_tools.show_bgp_neighbors_for_many_devices,
    multi_vendor_tools.show_bfd_sessions_for_many_devices,
    multi_vendor_tools.show_ip_route_for_many_devices,
    multi_vendor_tools.show_version_for_many_devices,
    topology_tools.topology_get_info_network_topology,
    topology_tools.topology_get_info_on_links,
    topology_tools.topology_get_links_between_devices,
    topology_tools.topology_get_vendor_of_devices,
```

```
    topology_tools.topology_get_list_of_shortnames_of_network_devices,
    topology_tools.topology_get_list_shortnames_of_endpoints
]

# WORKSHOP-PART3 CODE MODIFICATION - START
# UNCOMMENT THE NEXT 2 LINES
# tool_list.append(add_ospf_tools.show_ip_ospf_neighbor_detail_for_many_devices)
# tool_list.append(add_ospf_tools.show_ip_ospf_interface_for_many_devices)
# WORKSHOP-PART3 CODE MODIFICATION - END
```

4. Save the file

Restart the Net-Chat Assistant

Issue the following command in the VM terminal:

```
$ sudo ~/scripts/restart_net_chat.sh
Restarting net-chat ... done
```

In case of problems

Note: If you get lost, you can always use pre-prepared Python code by following the steps below:

```
$ sudo ~/scripts/fix_code_3.sh
$ sudo ~/scripts/restart_net_chat.sh
Restarting net-chat ... done
```

By default, the first script returns no output.

Re-check the Net-Chat Assistant for OSPF support

Using the web UI, ask the Net-Chat Assistant for OSPF support again:

```
Do you have any tools related to OSPF protocol?
```

Introduce a failure

In the terminal, run the script:

```
$ sudo ~/scripts/failure_3.sh
+-----------+-------+--------+-------------+-------------+
| Interface | Delay | Jitter | Packet Loss | Rate (kbit) |
+-----------+-------+--------+-------------+-------------+
| eth2      | 0s    | 0s     | 100.00%     |           0 |
+-----------+-------+--------+-------------+-------------+
```

The `failure_3.sh` script simulates an unidirectional network failure on the PE-red node by introducing 100% packet loss on interface eth2, effectively dropping all packets sent through this interface (RX packets will be fine). This can disrupt connectivity between devices that rely on PE-red for data transmission, potentially causing routing issues, loss of service, and degraded network performance for paths that traverse this interface.

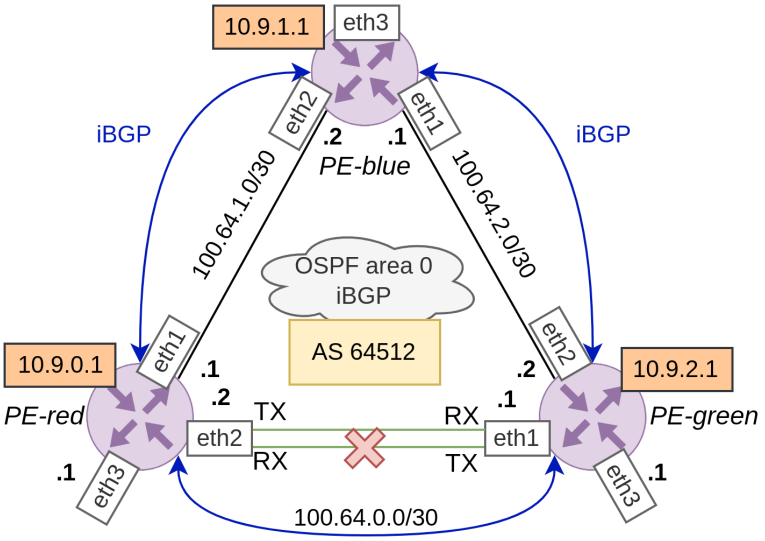


Figure 3: Simulated failure in Part 3

Check the impact of the failure on the network:

To check if there is indeed a problem between devices, issue the following commands and analyze the output:

```
$ ssh -t root@PE-red vtysh
PE-red# show ip ospf neighbor eth2
Neighbor ID    Pri State          Up Time    Dead Time …
10.9.2.1         1 Init/-         34.166s    35.833s    …
PE-red#

$ ssh -t root@PE-green vtysh
PE-green# show ip ospf neighbor eth1
Neighbor ID    Pri State          Up Time    Dead Time …
PE-green#
```

Use the Net-Chat Assistant to debug issues

- Switch to the Net-Chat Assistant in the browser and ask the following questions
- Observe the results and verify them against the examined network topology affected by the failure
- If the output is not satisfactory, try repeating the same query or rephrasing it
- Example queries:

```
Check each network device. Are there any problems in the
network?

Which OSPF neighbors have problems? Check all devices.

Show me all issues with OSPF on the PE-green device.

What is suspicious about OSPF protocol on PE devices?
Check each PE device..
```

Restore the network to a normal state

Execute the following script:

```
$ sudo ~/scripts/recovery_3.sh
+-----------+-------+--------+-------------+-------------+
| Interface | Delay | Jitter | Packet Loss | Rate (kbit) |
+-----------+-------+--------+-------------+-------------+
| eth2      | 0s    | 0s     | 0.00%       |           0 |
+-----------+-------+--------+-------------+-------------+
```

Part 4 - Adding a config check capability & debugging misconfigurations

Objectives

- Add support for checking current configuration on devices
- Debug misconfiguration issue between CE and PE

Note: The process of adding a new tool is similar to the one described in Part 3. In order to save time here we will apply changes via script instead of manual file editing. As a result, the new tool `show_configuration_for_many_devices` will give the Net-Chat Assistant the ability to retrieve running configurations from network devices. If you are curious about the internals, after applying code changes, please examine the new file:

```
~/src/code_edit/add_configuration_tool.py.
```

Steps

Automatically apply code changes

Change the source code by executing the script:

```
$ sudo ~/scripts/fix_code_4.sh
```

By default, the script returns no output.

Restart the Net-Chat Assistant container afterward:

```
$ sudo ~/scripts/restart_net_chat.sh
Restarting net-chat ... done
```

Introduce a network failure

In the console, execute the script introducing misconfiguration between CE-red and PE-red:

```
$ sudo ~/scripts/failure_4.sh
INFO[0002] Executed command "bash
/tmp/scripts/ospf_area_20.sh" on the node "CE-red". stdout:
```

The script changes OSPF configuration by changing area 10 on the eth1 interface on CE-red and to area 20, effectively breaking the existing OSPF adjacency. This change will disrupt the routing paths between PE-red and CE-red.

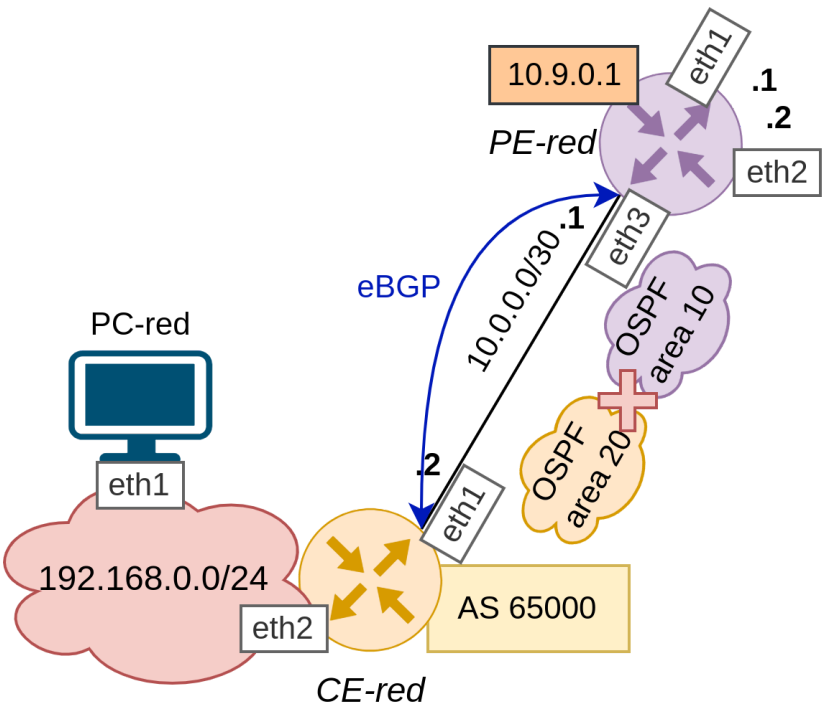


Figure 4: OSPF misconfiguration in Part 4

Check the impact of the failure on the network

To check if there is indeed a problem between devices, issue the following commands and analyze the output:

```
$ ssh -t root@PE-red vtysh
PE-red# show ip ospf interface eth3
eth3 is up
  ifindex 441, MTU 9500 bytes, BW 10000 Mbit …
  Internet Address 10.0.0.1/30, Broadcast 10.0.0.3, Area 0.0.0.10
  MTU mismatch detection: enabled
```

```
$ ssh -t vyos@CE-red
CE-red$ show ip ospf interface eth1
eth1 is up
 ifindex 442, MTU 9500 bytes, BW 10000 Mbit …
 Internet Address 10.0.0.2/30, Broadcast 10.0.0.3, Area 0.0.0.20
 MTU mismatch detection: enabled
```

Use the Net-Chat Assistant to debug issues

- Switch to Net-Chat Assistant in the browser and ask the following questions
- Observe the results, and verify them against the examined network topology affected by the misconfiguration
- If the output is not satisfactory, try repeating the same query or rephrasing it
- Example queries:

```
Analyze all issues with OSPF on each network device.
```

```
Show me more details about issues with OSPF neighbors on
the PE-red device.
```

```
Are there any configuration issues in the network? Check
each network device.

Is there any misconfiguration between CE-red and PE-red?

Show me more detailed configuration of the PE-red device.
Outline the part related to OSPF.

Are the OSPF areas the same on PE-red and CE-red devices?

Do the PE-red and the CE-red devices match the OSPF area?

Show me all details on OSPF configuration on the CE-red
device.
```

Restore the network to a normal state

Execute the following script:

```
$ sudo scripts/recovery_4.sh
INFO[0002] Executed command "bash
/tmp/scripts/ospf_area_10.sh" on the node "CE-red". stdout:
```

[OPTIONAL] Part 5 - Working on your own

If there is some time left or if you finish before others, feel free to play with the Net-Chat Assistant or device configuration yourself.

You have completed the hands-on successfully. Thank you for your commitment 🙂

Please leave your feedback:

https://codilime.typeform.com/autocon