blueplanet®
a division of Ciena

BPO Service Design and Development

BPO212ILT, Version 1.2

Blue Planet Orchestrate

Hands-on Training Lab Guide

LEGAL NOTICES

THIS DOCUMENT CONTAINS CONFIDENTIAL AND TRADE SECRET INFORMATION OF CIENA CORPORATION AND ITS RECEIPT OR POSSESSION DOES NOT CONVEY ANY RIGHTS TO REPRODUCE OR DISCLOSE ITS CONTENTS, OR TO MANUFACTURE, USE, OR SELL ANYTHING THAT IT MAY DESCRIBE. REPRODUCTION, DISCLOSURE, OR USE IN WHOLE OR IN PART WITHOUT THE SPECIFIC WRITTEN AUTHORIZATION OF CIENA CORPORATION IS STRICTLY FORBIDDEN.

EVERY EFFORT HAS BEEN MADE TO ENSURE THAT THE INFORMATION IN THIS DOCUMENT IS COMPLETE AND ACCURATE AT THE TIME OF PUBLISHING; HOWEVER, THE INFORMATION CONTAINED IN THIS DOCUMENT IS SUBJECT TO CHANGE.

While the information in this document is believed to be accurate and reliable, except as otherwise expressly agreed to in writing CIENA PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OR CONDITION OF ANY KIND, EITHER EXPRESS OR IMPLIED. The information and/or products described in this document are subject to change without notice. For the most up-to-date technical publications, visit www.ciena.com.

Security

Ciena cannot be responsible for unauthorized use of equipment and will not make allowance or credit for unauthorized use or access.

Contacting Ciena

Corporate headquarters	410-694-5700 or 800-921-1144	www.ciena.com
Customer technical support/warranty		
In North America	1-800-CIENA24 (243-6224) 410-865-4961	Email: CIENA24@ciena.com
In Europe, Middle East, and Africa	800-CIENA-24-7 (800-2436-2247) +44-207-012-5508	Email: CIENA24@ciena.com
In Asia-Pacific	800-CIENA-24-7 (800-2436-2247) +81-3-6367-3989 +91-124-4340-600	Email: CIENA24@ciena.com
In Caribbean and Latin America	800-CIENA-24-7 (800-2436-2247) 410-865-4944 (USA)	Email: CIENA24@ciena.com
Sales and General Information	410-694-5700	E-mail: sales@ciena.com
In North America	410-694-5700 or 800-207-3714	E-mail: sales@ciena.com
In Europe	+44-207-012-5500 (UK)	E-mail: sales@ciena.com
In Asia	+81-3-3248-4680 (Japan)	E-mail: sales@ciena.com
In India	+91-124-434-0500	E-mail: sales@ciena.com
In Latin America	011-5255-1719-0220 (Mexico City)	E-mail: sales@ciena.com
Training		E-mail: learning@ciena.com

For additional office locations and phone numbers, please visit the Ciena website at www.ciena.com

Change History

Blue Planet Release	Revision	Publication Date	Reason for Change
23.08	1.0	February 2024	Initial release.
23.08	1.1	September 2024	Minor update.
23.08	1.2	March 2025	Minor update.

Contents

Lab 1: Prepare a Development Environment

Objectives

- Set up an environment for BPO service design and development
- Deploy a custom PyPi repository that contains the Python modules needed for this set of labs

Documentation

Remote Script virtualenv - *https://developer.blueplanet.com/docs/bpocore-docs/references/type_layer/remote_scripts.html#remote_scripts_virtualenv*

Task 1: Prepare the Development Environment

Throughout this set of labs, you will be working on an L2VPN service. The labs approach the development in an iterative way – each lab will focus on a specific problem and add another piece to the final service. In this initial task, you perform some basic steps that allow you to prepare and use your development environment.

1. Open the **Terminal** from the left side.

2. Go to **~/training** folder.

```
student@POD-XX:~$
student@POD-XX:~$ cd training/
```

3. Clone the repository: *https://gitlab.bptrn.com/studentX/bpo-sdd.git*. Change *X* to your pod number (e.g. POD01 > student1, POD15 > student15).

```
student@POD-XX:~/training$ git clone https://gitlab.bptrn.com/studentXX/bpo-sdd.git
Cloning into 'bpo-sdd'...
remote: Enumerating objects: 4, done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 4
Receiving objects: 100% (4/4), done.
```

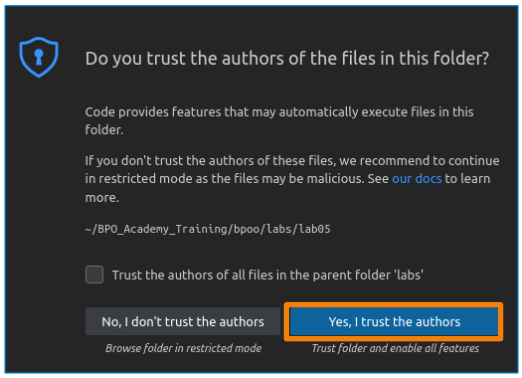
4. Move into the cloned repository **bpo-sdd**.

```
student@POD-XX:~/training$ cd bpo-sdd
student@POD-XX:~/training/bpo-sdd$
```

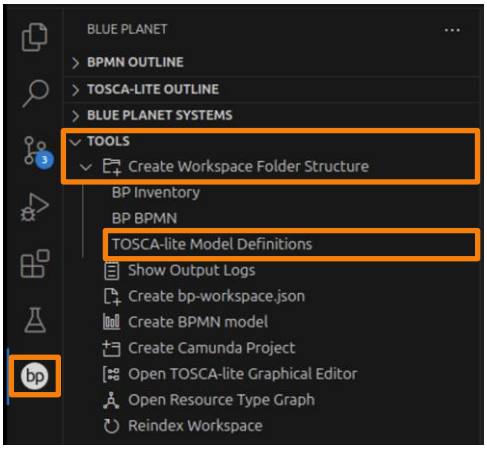
5. Use **code .** command to open the folder in VS Code.

```
student@POD-XX:~/training/bpo-sdd$ code .
```

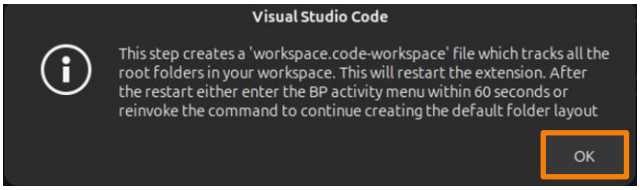
6. In VS Code, if a window for trusting the file authors shows up, confirm it.



7. Now use the BP extension to create the layout for TOSCA Model Definitions. In the BP extension, go to **TOOLS > Create Workspace Folder Structure > Tosca-Lite Model Definitions**. Accept the pop-up about the choice of the primary root folder.

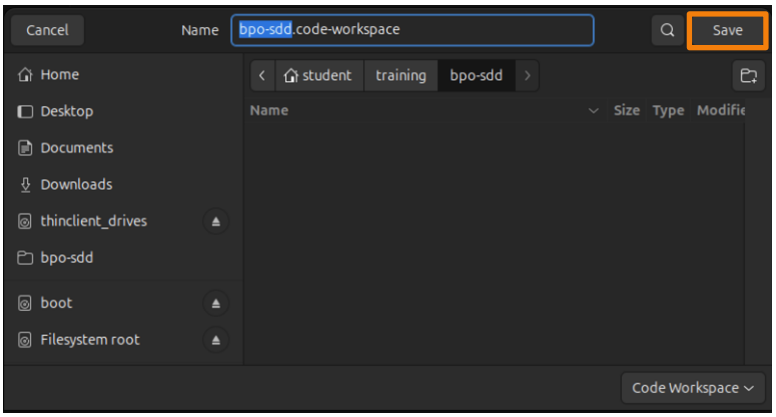


8. Read and accept the pop-up about the *workspace.code-workspace* file and the timing of the BP activity.

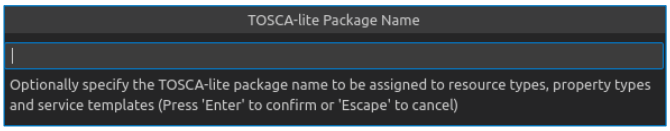


NOTE: After accepting the pop-up, a window will open with the content of the user's (student) home directory. You will create a subdirectory (sales-project) to act as the root folder of your workspace. The lab guide assumes that you will not enter the BP activity menu within 60 seconds and therefore you will reinvoke the command (Create Workspace Folder Structure) to create the folder layout.

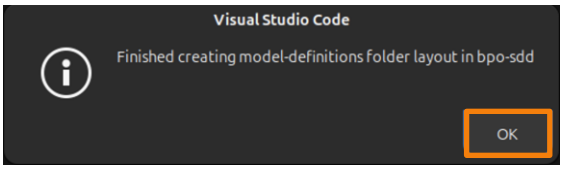
9. In the new window, click **Save** in the top-right corner.



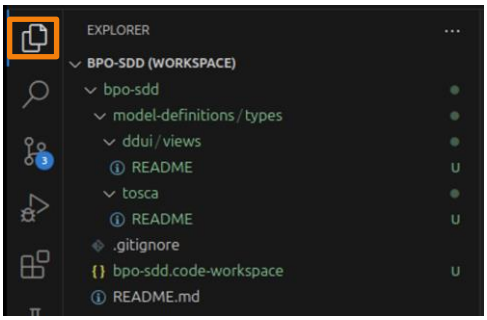
10. Click into the popup asking for TOSCA-lite Package Name and press **Escape (Esc)** on your keyboard.



11. Confirm the popup with **OK**.

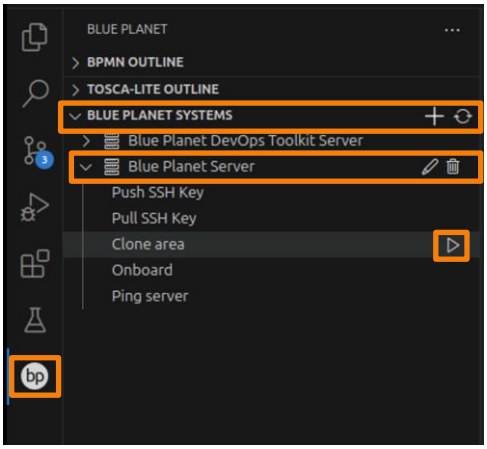


12. In the **Visual Studio Code Explorer**, expand your workspace and investigate the folder layout.

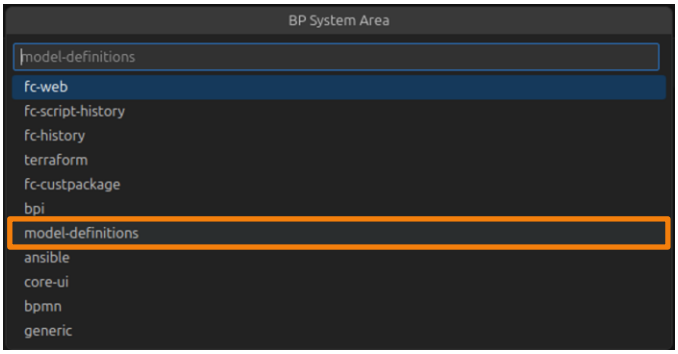


NOTE: The folder layout of your workspace (bpo-sdd) is complete. The root folder (bpo-sdd) contains model-definitions/types directories with further subfolders (ddui/views and tosca). The subfolders are empty, except for the README files.

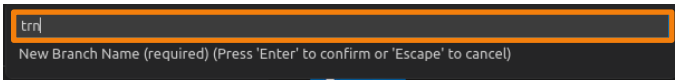
13. In the next steps, you will clone the model-definitions. Adding the cloned repository to the workspace will complete the multi-root workspace structure. Still in VS Code, open the **BP extension**.
Under BLUE PLANET SYSTEMS, expand the **Blue Planet Server** and click the **Play** button next to Onboard (the button will show up once you hover over the Onboard line).



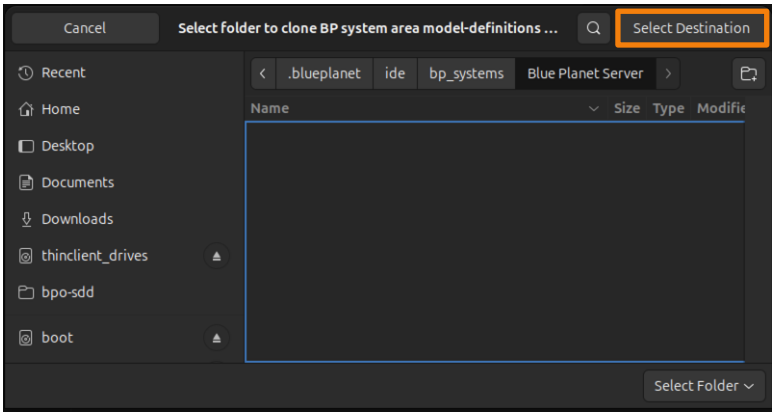
14. Choose the **model-definitions** area. The drop-down menu presents a list of all git repositories available on the server.



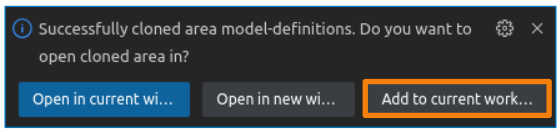
15. Enter **trn** for branch name and press **Enter**.



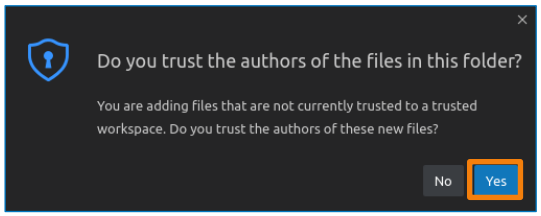
16. In the window that specifies the /home/student/.blueplanet/ide/bp_systems/BPO Server, click **Select Destination** in the top-right corner to create a subdirectory for the cloned repository.



17. A popup will show up in bottom right corner of VS Code. Click **Add to current workspace**.

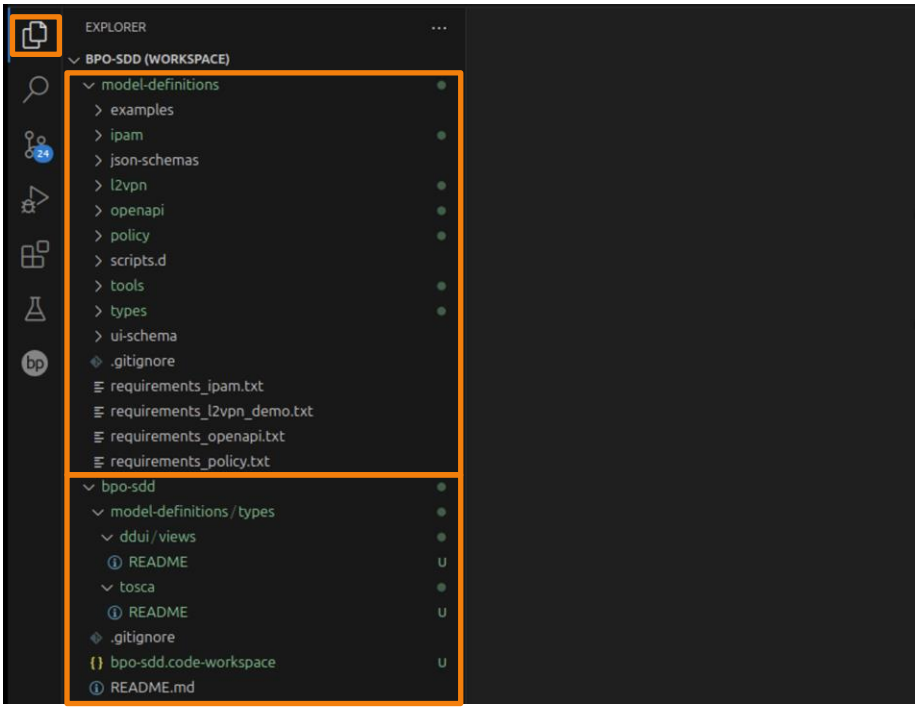


18. Confirm to trust the file authors.



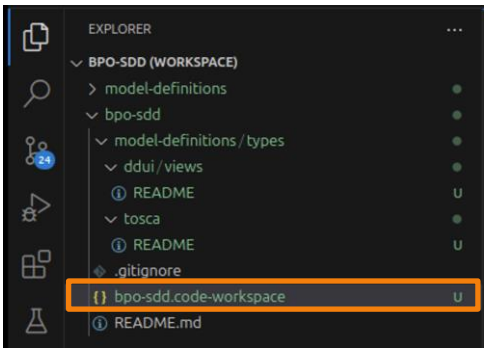
NOTE: You have now created a multi-root workspace. It contains a primary workspace (bpo-sdd) folder and a cloned repository of an asset area (model-definitions).

19. Open the explorer and examine the workspace structure.



NOTE: The onboarding process will synchronize your files from **bpo-sdd** to the source repository on the server and to the **cloned repository folder** in your workspace. You use this process in the next tasks.

20. In the next steps you will change the order of cloned repository **model-definitions** and working directory **bpo-sdd** in the Explorer view. In the VS Code Explorer, expand your workspace and click on the **bpo-soo.code-workspace** file.



21. Change the **json structure** in the code-workspace file and save it.

```
{
        "folders": [
                {
                        "path": "."
                },
                {
                        "name": "model-definitions",
                        "path": "../../.blueplanet/ide/bp_systems/Blue Planet Server/model-definitions"
                }
        ],
        "settings": {}
}
```

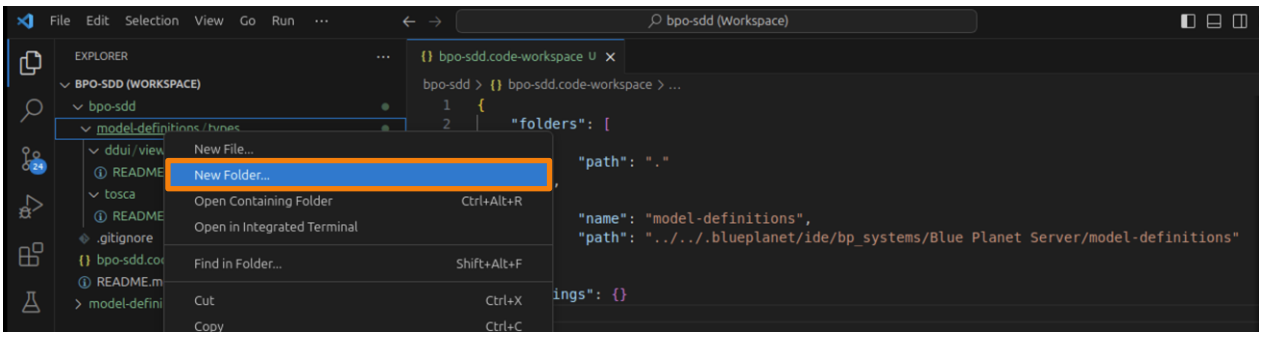
VS Code will arrange the folders in the Explorer view as specified in the **.code-workspace** file.

22. Leave VS Code open, you will use it in the next lab.

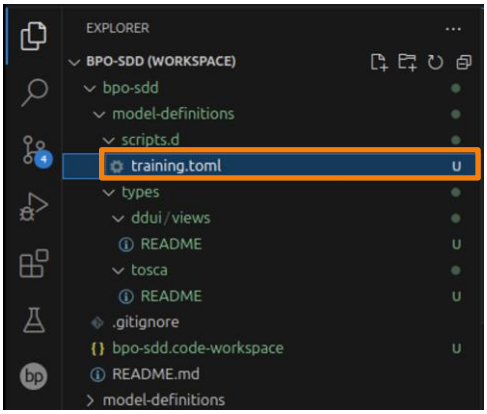
Task 2: Create a Python Virtual Environment

A common part of service development is often the execution of remote scripts that take care of different parts of service automation. These scripts use Python and may have different dependencies. For this reason, it is important to prepare virtual environments (using Python's *virtualenv*) to isolate their execution environments. In this task, you create a virtualenv for the purpose of this set of labs, however, you can read more about these environments in the link provided in the *Documentation* section of this lab.

1. In VS Code, create the **scripts.d** folder in **bpo-sdd/model-definitions** directory.



2. Create a **training.toml** file inside the script.d folder. The name of that file determines what scripts the virtual environment will cover. In your case, the *training.toml* file creates a virtual environment for Python scripts located in the *bpo-sdd/model-definitions/training* folder.

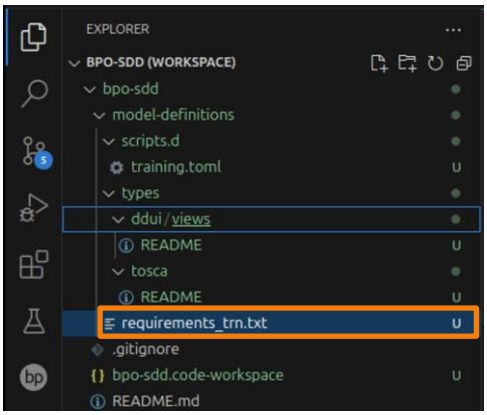


NOTE: If you, for example, named the file training.test.toml, it would create a virtual environment for scripts located in training/model-definitions/training/test folder.

3. Edit the **training.toml** file and add the following contents.
   The python key lets BPO know which Python version should be used, and the requirements key describes the file name that contains Python module dependencies.

```
[virtualenv]
python = "py310"
requirements = "requirements_trn.txt"
```

4. Create the **requirements_trn.txt** file in the **bpo-sdd/model-definitions** folder.
   This file is a classic Python requirements file, in which you define Python modules, and optionally their versions, remote PyPi repositories, and other parameters.

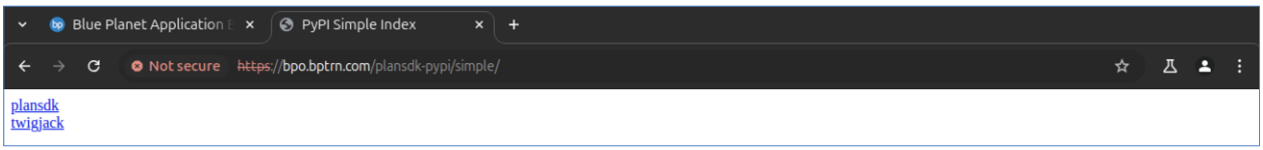


5. Modify the **requirements_trn.txt** file.
   With this modification, you add the **plansdk** Python module as a module that should be used in your virtual environment whenever you run any scripts located in the training folder. The *http://blueplanet/plansdk-pypi/simple/* URL points to a local PyPi repository that is already set up with the current BPO deployment.

```
-i http://blueplanet/plansdk-pypi/simple/
plansdk
```

6. You can verify the available packages via the web browser. Fist open **Chromium**.

7. Before connecting to the URL pointing to the local PyPi repository, make sure you are logged into BPO. Go to *bpo.bptrn.com* and login.
8. Open a new tab and open *https://bpo.bptrn.com/plansdk-pypi/simple/.*



NOTE: The reason we are using the *http://blueplanet* URL in the requirements file is that a number of apps in the BPO ecosystem define a blueplanet hostname that resolves to the IP addresses of the Blue Planet load balancer. The load balancer—implemented by the HAProxy app in the platform solution—treats requests from within the site to the HTTP port (80) as internal and bypasses the API Gateway. Conversely, the load balancer treats requests to the HTTPS port (443) as external.

Task 3: Create a Custom PyPI Repository

While a standard BPO deployment already comes with a built-in PyPi repository, it only contains two different Python modules. If you want to add additional packages, you can either add them to this repository, or better yet – create your own PyPi repository. Blue Planet provides a tool called *mkpypi* for this purpose. The tool is used to create a PyPi repository in the form of a Blue Planet solution – a standardized way of deploying custom applications within a Blue Planet ecosystem.

1. Open the **Terminal**.

2. Navigate to **~/training**.

```
student@POD-XX:~$ cd ~/training
```

3. First, obtain the required Python modules from Gitlab. Clone the **http://gitlab.lab.local/labs/pypi-modules** repository, which contains all the Python modules you will need for this set of labs.

```
student@POD-XX:~/training$ git clone https://gitlab.bptrn.com/bptrn/trn-pypi.git
Cloning into 'trn-pypi'...
remote: Enumerating objects: 13, done.
remote: Total 13 (delta 0), reused 0 (delta 0), pack-reused 13
Receiving objects: 100% (13/13), 453.36 KiB | 56.67 MiB/s, done.
```

4. Inspect the folder contents with the **ls** command. All these Python modules (.whl files) are Blue Planet proprietary, and you will not find them in any public PyPi repository.

```
student@POD-XX:~/training$ cd trn-pypi
student@POD-XX:~/training/trn-pypi$ ls
fig.yml                              SOLUTION.md
openapisdk-2308.0.0-py3-none-any.whl  spiffworkflow-2.0.2-py3-none-any.whl
planext-1.2.5-py3-none-any.whl       tmfsdk-0.0.1.10-py3-none-any.whl
plansdk-4.9.2-py3-none-any.whl       twigjack-5.4.2-py2.py3-none-any.whl
README.md                            workflowext-1.0.0.204-py3-none-any.whl
requirements_2308
```

5. Enter **python3.10 -m venv env** to create a virtual environment.

```
student@POD-XX:~/training/trn-pypi$ python3.10 -m venv env
```

6. Next, activate the environment with the **source env/bin/activate** command. You will notice the environment is active by the **(env)** at the beginning of the prompt in the terminal.

```
student@POD-XX:~/training/trn-pypi$ source env/bin/activate
(env) student@POD-XX:~/training/trn-pypi$
```

7. Install **wheel** (command line tool for manipulating Python wheel file) package in the envitonment.

```
(env) student@POD-XX:~/training/trn-pypi$ pip install wheel
Collecting wheel
...
```

8. Prepare the packages by packaging all required modules in the **requirements_2308** file and their dependencies in the **dist** folder.

```
(env) student@POD-XX:~/training/trn-pypi$ pip wheel --wheel-dir dist --find-links . -r requirements_2308
Looking in links: .
Processing ./openapisdk-2308.0.0-py3-none-any.whl
Processing ./planext-1.2.5-py3-none-any.whl
...
```

9. Inspect the help pages for the **mkpypi** command. You will notice that the command produces an output that matches a .tar file of a Blue Planet solution container.

```
(env) student@POD-XX:~/training/trn-pypi$ mkpypi --help
Usage of mkpypi:
  -a value
```

```
        name of Blue Planet app to output as tar layers
  -f    force overwrite of output directory
  -o string
        output directory for the PEP 503 simple index
  -p string
        path of a directory which contains the Python sdist and wheel distributions to use
  -s value
        name of Blue Planet solution to include in the app tar layers
  -v    show version information
```

10. Create the application image with the **mkpypi** command and pipe it into **docker load** command.

```
(env) student@POD-XX:~/training/trn-pypi$ mkpypi -p dist -a registry.bptrn.com/bptrn/trn-pypi:2308.0.0 |
docker load
```

11. In addition to those Docker images, a Blue Planet solution also needs a solution metadata file – fig.yml. You will learn more about those in a different lab. For now, inspect the **fig.yml** file in your current directory and enter the configuration below.

```
(env) student@POD-XX:~/training/trn-pypi$ cat fig.yml
__version__: 1
solution_name: trn_pypi
solution_version: 2308.0.0
apps:
    trn-pypi:
        image: registry.bptrn.com/bptrn/trn-pypi:2308.0.0
        environment:
          - BP2RemoveDataOnRestore=true
        volumes:
          - /dev/log:/dev/log
          - /etc/hostname:/etc/physical_hostname:ro
          - /etc/bp2/site:/etc/bp2/site:ro
```

12. Use usolmaker to create the Solution Data Image (SDI) from the fig.yml file. The tag must match the solution_version in fig.yml file.

```
(env) student@POD-XX:~/training/trn-pypi$ usolmaker -f fig.yml registry.bptrn.com/bptrn/solution-trn_pypi
2308.0.0 | docker load
```

13. Verify that the image has been loaded. You should find two new docker images there – a PyPi app image and a Blue Planet solution image.

```
(env) student@POD-XX:~/training/trn-pypi$ docker images | grep pypi
registry.bptrn.com/bptrn/solution-trn_pypi    2308.0.0     8694e0198c10   31 seconds ago       4.9kB
registry.bptrn.com/bptrn/trn-pypi             2308.0.0     4a1b84b9325a   About a minute ago   15.4MB
```

14. Since the solution and its images are stored on your Jumphost, you need to push it to the BPO server. First, save the solution with the **docker save** command.

```
(env) student@POD-XX:~/training/trn-pypi$ docker save -o solution-trn_pypi_2308.0.0
registry.bptrn.com/bptrn/solution-trn_pypi:2308.0.0 registry.bptrn.com/bptrn/trn-pypi:2308.0.0
```

15. Install your public key as an authorized key on the BPO server.

```
(env) student@POD-XX:~/training/trn-pypi$ ssh-copy-id bpuser@bpo.bptrn.com
```

NOTE: If the ssh key was already copied on the server, you will get a warning saying the keys were skipped.

16. Now transfer the solution to the BPO server with the **scp** command.

```
(env) student@POD-XX:~/training/trn-pypi$ scp solution-trn_pypi_2308.0.0 bpuser@bpo.bptrn.com:
```

17. Good job, you managed to create your first Blue Planet solution! However, when working in a real scenario, this entire process should be automated (for example, in the form of a script or a CI/CD pipeline).

Task 4: Onboard the Custom PyPI Repository

It is time to deploy your custom PyPi repository solution on the BPO server. This way, it can be used as a PyPi repository for any dependencies your service code and plans might need.

1. Connect to the BPO server.

```
student@POD-XX:~/training/trn-pypi$ ssh bpuser@bpo.bptrn.com
```

2. Using the **docker load** command, load the solution images on the BPO server. Make sure that the command completes successfully.

```
[bpuser@bpoXX~]$ docker load -i solution-trn_pypi_2308.0.0
Loaded image: registry.bptrn.com/bptrn/solution-trn_pypi:2308.0.0
23648c0c18bb: Loading layer [==================================================>]  15.4 MB/15.4 MB
Loaded image: registry.bptrn.com/bptrn/trn-pypi:2308.0.0
```

3. You can now deploy the PyPi solution. This is done with **solman** – the Blue Planet solution manager. Enter the solman CLI with the **solman** command.

```
[bpuser@bpoXX~]$ solman
Connecting smcli to solutionmanager_0

(Cmd)
```

4. Deploy the PyPi solution with the **solution_deploy** command.

```
(Cmd) solution_deploy registry.bptrn.com.bptrn.trn_pypi:2308.0.0
Deploying registry.bptrn.com.bptrn.trn_pypi:2308.0.0
    Installing registry.bptrn.com.bptrn.trn_pypi:2308.0.0
        SDI Already Downloaded
        Installing SDC
            'registry.bptrn.com.bptrn.trn_pypi:2308.0.0'
        App Images Already Download
        Creating App Containers
            ['trn-pypi_2308.0.0_0']
    Starting registry.bptrn.com.bptrn.trn_pypi:2308.0.0
        Starting App Containers
            ['trn-pypi_2308.0.0_0']
(Cmd)
```

5. The PyPi solution should now be running. Exit the **solman** tool with the **exit** command.

```
(Cmd) exit
```

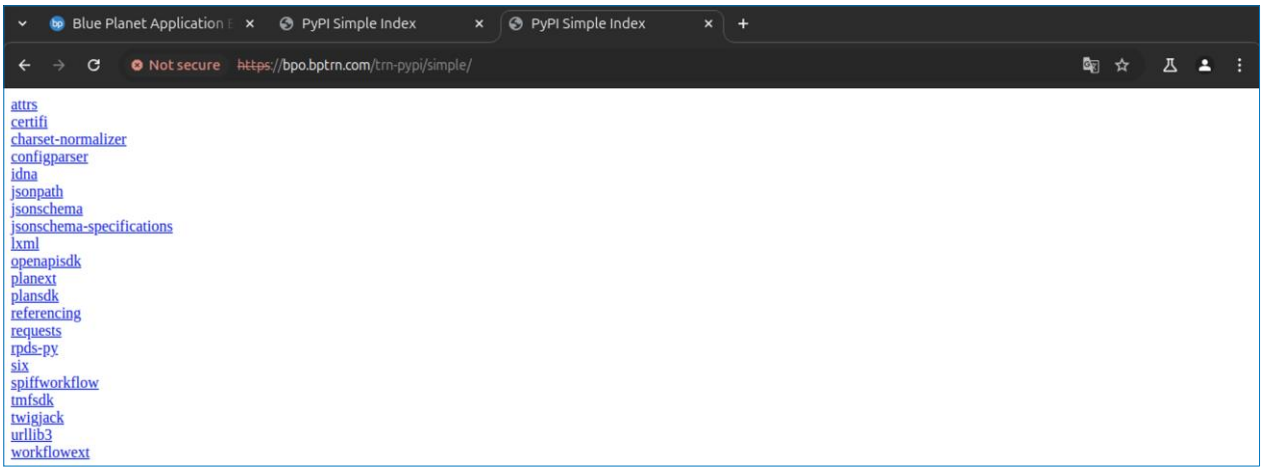
6. Find the Docker container that is running as your solution container. It should be up and running.

```
[bpuser@bpoXX~]$ docker  ps | grep trn-pypi
        registry.bptrn.com/bptrn/trn-pypi:2308.0.0                                  "pypi-httpd
/trn-pypi"   19 seconds ago        Up 17 seconds
trn-pypi_2308.0.0_0
```

7. Disconnect from the BPO server.

```
[bpuser@bpoXX~]$ exit
logout
Connection to bpo.bptrn.com closed.
```

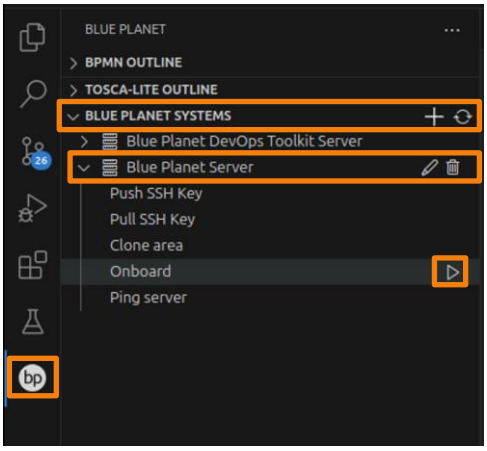
8. Your custom PyPi repository solution is now available. Open the browser and got to *https://bpo.bptrn.com/trn-pypi/simple/*. Make sure you are logged into *bpo.bptrn.com* before accessing the PyPi site.



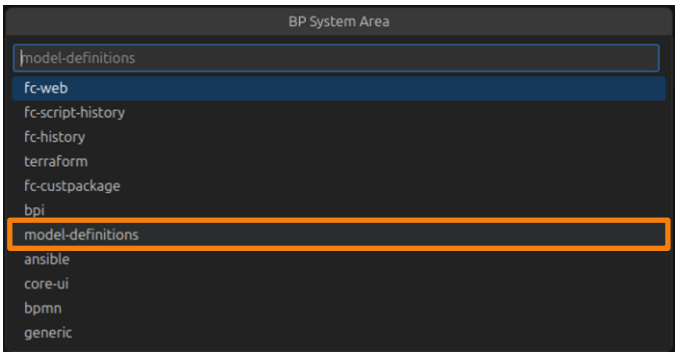
9. Update your Python requirements with the new PyPi registry. In VS Code, open the **requirements_trn.txt**.
10. Update the **requirements_trn.txt** file. With the content displayed below. Use the *http://blueplanet/trn-pypi/simple/* – this is the custom repository with Blue Planet proprietary modules.

```
-i http://blueplanet/trn-pypi/simple/
 plansdk
```

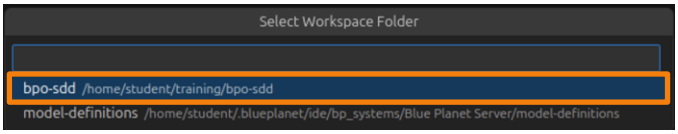
11. Onboard the **bpo-sdd/model-definitions** using the BP extension. Open the **BP extension** and in the BLUE PLANET SYSTEMS, expand the **Blue Planet Server** and click the **Play** button in the Onboard line.



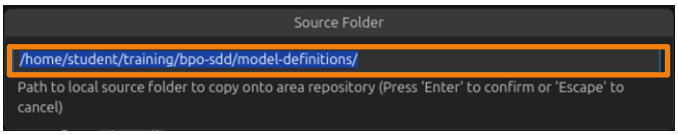
12. In the popup, select **model-definitions**.



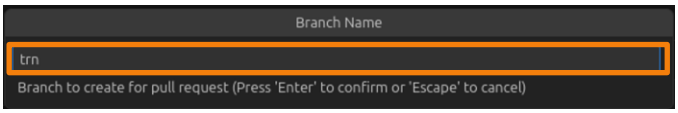
13. Next, select the **bpo-sdd** workspace folder.



14. Confirm the path to the **bpo-sdd/model-definitions** folder.

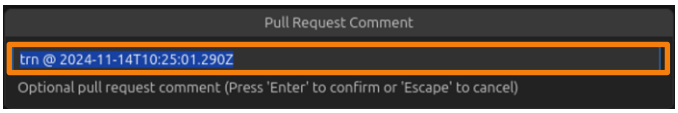


15. Set the branch name to **trn**. This branch name must match the branch name of the local model-definitions repository.

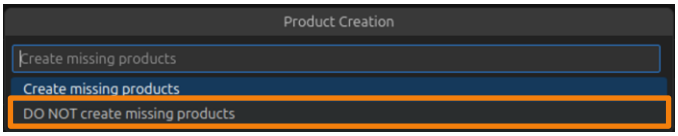


NOTE: Failing to match the two branch names will result in a failed onboarding.

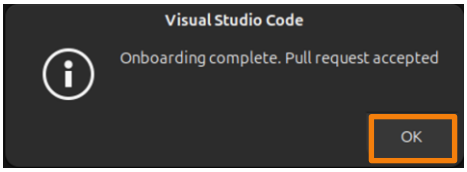
16. Confirm the default commit message (pull request comment).



17. Select **DO NOT create missing products**.



18. Confirm the popup noting that the onboarding was completed and the pull request accepted.



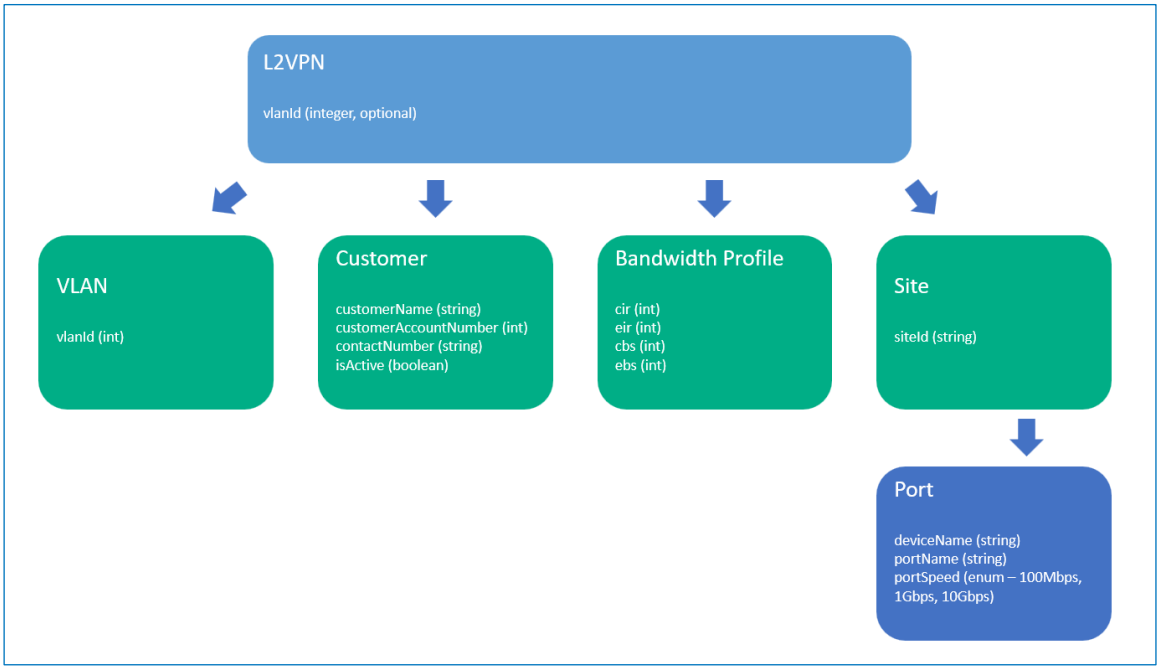
19. Congratulations! You have finished this set of tasks and are now ready to start designing and developing services.
20. Leave the VS Code open, you will use it in the next task.

End of Lab

Lab 2: Create Resource Type Definitions

Objectives

- Create Resource Type definitions.
- Create a Declarative Service Template.
- Learn how to utilize built-in resource types.
- Implement the following structure using custom resource definitions:



Documentation

TOSCA definitions - *https://developer.blueplanet.com/docs/bpocore-docs/references/type_layer/README.html*

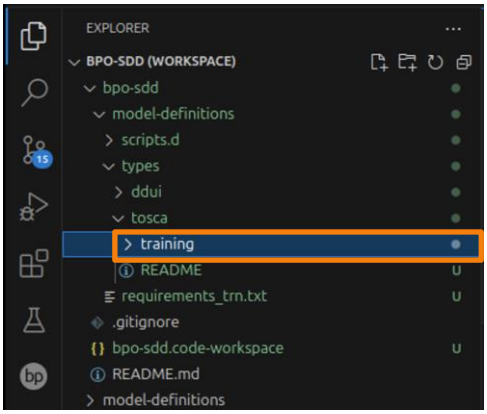
Template directives - *https://developer.blueplanet.com/docs/bpocore-docs/references/type_layer/st_directives.html#_getattr*

NumberPool allocation - *https://developer.blueplanet.com/docs/bpocore-docs/apis/components/number-pool.html*

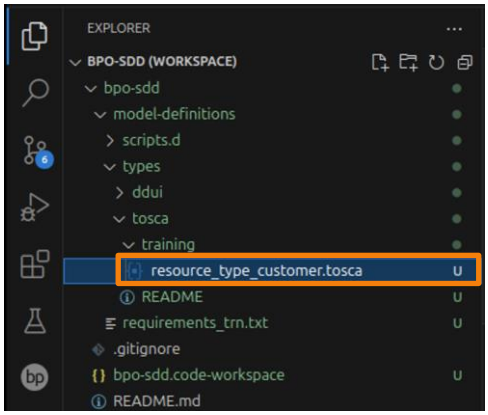
Task 1: Create Resource Types Definitions

In this initial task, you create the basic resource type definitions for Customer, BandwidthProfile, Site, Port, and L2VPN, which will be used in the final service.

1. Study the diagram under the **Objectives** section of this lab. Your goal is to implement the same structure using custom resources in Blue Planet Orchestrator.
2. Create a **training** folder in the **bpo-sdd/model-definitions/types/tosca**.



3. In the **types/tosca/training** folder, create a file for the customer resource. Name the file **resource_type_customer.tosca**. By having the Blue Planet Extension installed for VS Code, some template text for resource types should get added automatically for this file.



4. First, edit the headers of the TOSCA file and change the **package**, **title**, **author**, and **description** values.

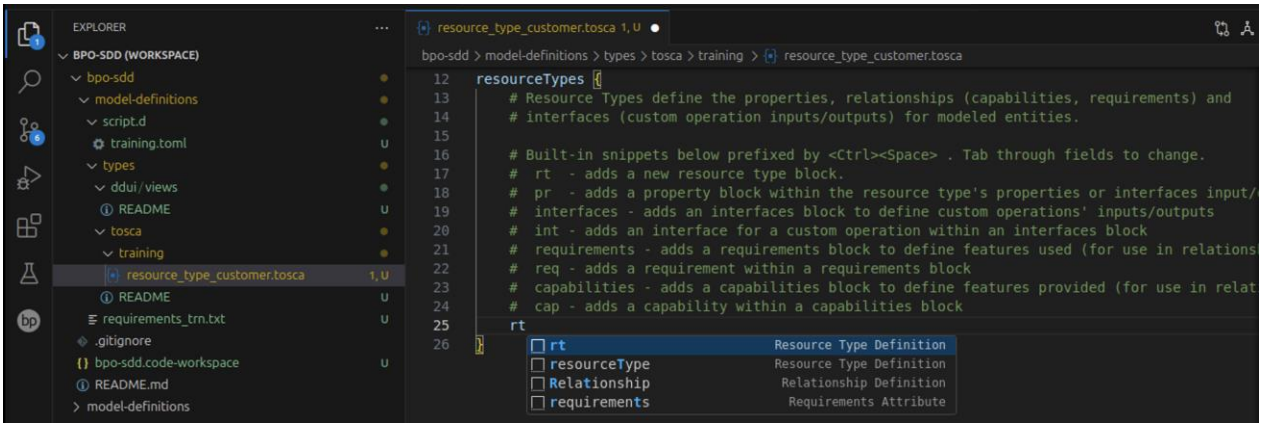
```
"$schema"    = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title        = "Customer resource type definition"
package      = training
version      = "1.0"
description = "This TOSCA document defines the Customer resource type."
authors      = [ "Developer (developer@bptrn.com)" ]
```

5. Add an import statement that imports the Root type using the **tosca.resourceTypes.Root** type.

```
imports {
  Root = tosca.resourceTypes.Root
}
```

6. Delete the **propertyTypes** and **serviceTemplates** statements since you do not need them yet in this part of the lab.

7. Start editing the **resourceTypes**. You can start typing **rt** and then choose the ResourceType to use the Blue Planet Extension to create an empty resource type.



You will end up with the following **resourceType** template.

```
resourceTypes {
  ResourceTypeExample {
    title       = Resource Type Example
    description = Resource type example
    derivedFrom = tosca.resourceTypes.Root
    properties {

    }
  }
}
```

8. Name the resourceType **Customer**, which should be derived from the Root import, and set its description.

```
resourceTypes {
  Customer {
    title = "Customer"
    description = "The Customer resource is used to define customer information"
    derivedFrom = Root

    properties {

    }
}
```

9. Finally, set the resource type's properties. Add the following properties:
    - **customerName** (string)
    - **customerAccountNumber** (integer)
    - **contactNumber** (string)
    - **isActive** (Boolean, updateable=true, optional=true)

```
    properties {
      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
     }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as an integer"
        type = integer
      }
```

```
      contactNumber {
        title = "Contact Phone Number"
        description = "Contact phone number of the customer"
        type = string
      }

      isActive {
        title = "Active"
        description = "Whether the Customer is active or not"
        type = boolean
        updatable = true
        optional = true
      }
    }
```

10. This is how your **Customer** resource type file should look like in the end.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer resource type."
authors     = [ "Developer (developer@bptrn.com)" ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  Customer {
    title = "Customer"
    description = "The Customer resource is used to define customer information"
    derivedFrom = Root

    properties {
      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as an integer"
        type = integer
      }

      contactNumber {
        title = "Contact Phone Number"
        description = "Contact phone number of the customer"
        type = string
      }

      isActive {
        title = "Active"
        description = "Whether the Customer is active or not"
        type = boolean
        updatable = true
        optional = true
      }
    }
  }
}
```

11. Next, create a **Port** resource type in a new file called **resource_type_port.tosca**. Modify the header in a similar way you did for the Customer, import the **Root** resource type, and add the following properties:
    - **deviceName** (string)
    - **portName** (string)
    - **portSpeed** (enum – 100Mbps, 1Gbps, 10Gbps)
12. The finished Port resource type file should look like this.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Port resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Port resource type."
authors     = [ "Developer (developer@bptrn.com)" ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  Port {
    derivedFrom = Root
    title = "Port"
    description = "The Port resource is used to define port information"

    properties {
      deviceName {
        title = "Device Name"
        description = "Name of the device"
        type = string
      }

      portName {
        title = "Port Number"
        description = "Representing the port interface number"
        type = string
      }

      portSpeed {
        title = "Port Speed"
        description = "Speed of the port"
        type = string
        enum = [100Mbps, 1Gbps, 10Gbps]
      }
    }
  }
}
```

13. Create a resource type for **BandwidthProfile** using a **resource_type_bandwidthprofile.tosca** file. Use the following properties, where the value for each property is greater than 8,000 and lower than 100,000:
    - cir (integer), meaning Committed Rate
    - eir (integer), meaning Excessive Rate
    - cbs (integer), meaning Committed Burst
    - ebs (integer), meaning Excessive Burst
14. The finished **BandwidthProfile** resource type file should look like this.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Bandwidth Profile resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the BandwidthProfile resource type."
authors     = [ "Developer (developer@bptrn.com)" ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  BandwidthProfile {
    derivedFrom = Root
    title = "Bandwidth Profile"
    description = "The Bandwidth Profile resource is used to define the bandwidth information of the L2VPN
service"

    properties {
      cir {
        title = "Committed Rate"
        description = "Committed rate of the bandwidth profile"
        type = integer
        minimum = 8000
        maximum = 100000
      }

      eir {
        title = "Excessive Rate"
        description = "Excessive rate of the bandwidth profile"
        type = integer
        minimum = 8000
        maximum = 100000
      }

      cbs {
        title = "Committed Burst"
        description = "Committed burst of the bandwidth profile"
        type = integer
        minimum = 8000
        maximum = 100000
      }

      ebs {
        title = "Excessive Burst"
        description = "Excessive burst of the bandwidth profile"
        type = integer
        minimum = 8000
        maximum = 100000
      }
    }
  }
}
```

15. Next, create a resource type for **Site** using a **resource_type_site.tosca** file. Use the following properties:
    - siteId (string)

    However, due to the hierarchy described in the Objectives image for this lab, the Site resource should also create the Port resource. For this reason, you need to add the Port properties to the Site resource type as well:
    - deviceName (string)
    - portName (string)
    - postSpeed (enum – 100Mbps, 1Gbps, 10Gbps)
16. The finished **Site** resource type file should look like this.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Site resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Site resource type."
authors     = [ "Developer (developer@bptrn.com)" ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  Site {
    derivedFrom = Root
    title = "Site"
    description = "The Site resource is used to define the site information of the L2VPN service"

    properties {
      portName {
        title = "Port Name"
        description = "Label of the first Port that is to be used for service creation"
        type = string
      }

      siteId {
        title = "Site Identifier"
        description = "Description of the Site that the port is from"
        type = string
      }

      deviceName {
        title = "Device Name"
        description = "Name of the device"
        type = string
      }

      portSpeed {
        title = "Port Speed"
        description = "Speed of the port"
        type = string
        enum = [100Mbps, 1Gbps, 10Gbps]
      }
    }
  }
}
```

17. Finally, create a resource type for **L2VPN** using a **resource_type_l2vpn.tosca** file. In the same file create also the **VLAN** resource type. The idea is to add a **vlanId** parameter to the top-level resource, which can be either input manually when creating the resource or generated dynamically if the input field is left empty.

    a. For the VLAN resource type use the following property:

    - **vlanId** (integer)

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)" ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  VLAN {
    derivedFrom = Root
    title = "VLAN"
    description = "Designated VLAN"

    properties {
      vlanId {
        title = "VLAN"
        description = "Specify or allocate the VLAN to be used by the L2VPN"
        type = integer
      }
    }
  }
}
```

    b. For the L2VPN resource type Use the following properties:

    - **vlanId** (integer, optional=true)
    - **customerName** (string)
    - **customerAccountNumber** (integer)
    - **contactNumber** (string)
    - **isActive** (Boolean, updateable=true, optional=true)
    - **deviceName** (string)
    - **portName** (string)
    - **portSpeed** (enum – 100Mbps, 1Gbps, 10Gbps)
    - **cir** (integer)
    - **eir**(integer)
    - **cbs** (integer)
    - **ebs** (integer)
    - **portName** (string)
    - **siteId** (string)

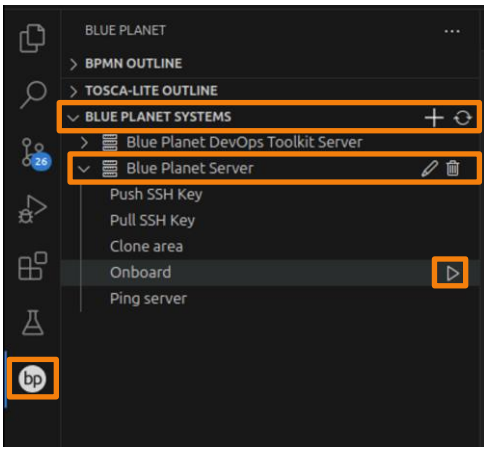
    As you can see, most of the L2VPN properties (except vlanId) relate to resources or which you have already created resource types, meaning they will get used in the creation of those sub-resources. Make sure to apply any limitations (such as min and max value) to these properties.

18. The finished **L2VPN** resource type file should look like this.

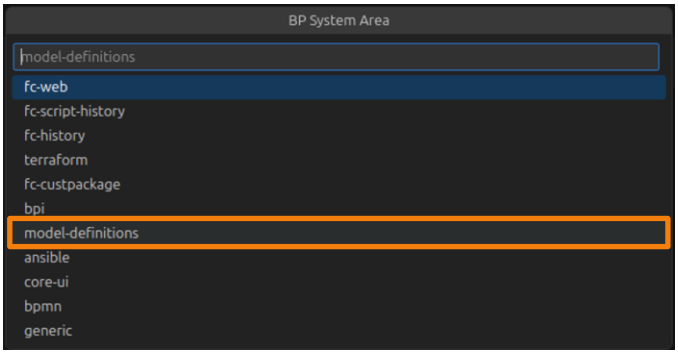
```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)" ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  VLAN {
    derivedFrom = Root
    title = "VLAN"
    description = "Designated VLAN"

    properties {
      vlanId {
        title = "VLAN"
        description = "Specify or allocate the VLAN to be used by the L2VPN"
        type = integer
      }
    }
  }

  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {
      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
      }

      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as an integer"
        type = integer
      }

      contactNumber {
        title = "Contact Phone Number"
        description = "Contact phone number of the customer"
        type = string
      }

      isActive {
        title = "Active"
        description = "Whether the Customer is active or not"
        type = boolean
        updatable = true
        optional = true
      }
```

```
      portName {
        title = "Port Name"
        description = "Label of the first Port that is to be used for service creation"
        type = string
      }

      siteId {
        title = "Site Identifier"
        description = "Description of the Site that the port is from"
        type = string
      }

      deviceName {
        title = "Device Name"
        description = "Name of the device"
        type = string
      }
      portName {
        title = "Port Number"
        description = "Representing the port interface number"
        type = string
      }

      portSpeed {
        title = "Port Speed"
        description = "Speed of the port"
        type = string
        enum = [100Mbps, 1Gbps, 10Gbps]
      }

      cir {
        title = "Committed Rate"
        description = "Committed rate of the bandwidth profile"
        type = integer
        minimum = 8000
        maximum = 100000
      }

      eir {
        title = "Excessive Rate"
        description = "Excessive rate of the bandwidth profile"
        type = integer
        minimum = 8000
        maximum = 100000
      }

      cbs {
        title = "Committed Burst"
        description = "Committed burst of the bandwidth profile"
        type = integer
        minimum = 8000
        maximum = 100000
      }

      ebs {
        title = "Excessive Burst"
        description = "Excessive burst of the bandwidth profile"
        type = integer
        minimum = 8000
        maximum = 100000
      }
    }
  }
}
```

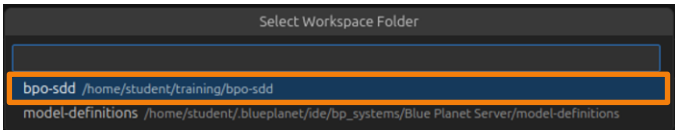
19. Onboard the **bpo-sdd/model-definitions** using the BP extension. Open the **BP extension** and in the BLUE PLANET SYSTEMS, expand the **Blue Planet Server** and click the **Play** button in the Onboard line.



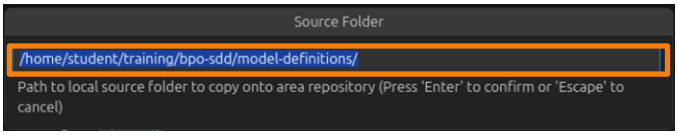
20. In the popup, select **model-definitions**.



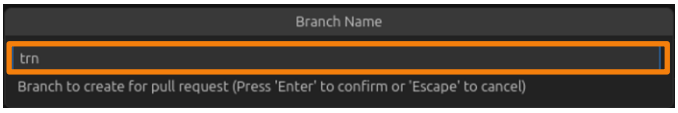
21. Next, select the **bpo-sdd** workspace folder.



22. Confirm the path to the **bpo-sdd/model-definitions** folder.

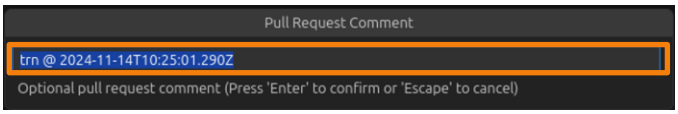


23. Set the branch name to **trn**. This branch name must match the branch name of the local model-definitions repository.

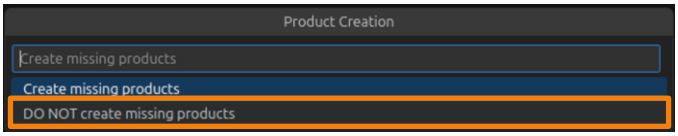


NOTE: Failing to match the two branch names will result in a failed onboarding.

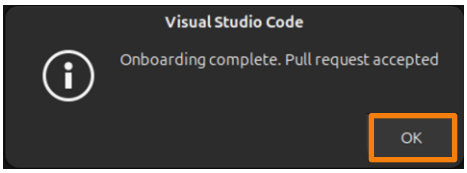
24. Confirm the default commit message (pull request comment).



25. Select **DO NOT create missing products**.



26. Confirm the popup noting that the onboarding was completed and the pull request accepted.



27. Once completed, the request should be marked in green as *accepted*. In any other case (for example, the request is rejected), you can click on the request and inspect the error message. At this point, the issue in most cases will be the incorrect formatting of the TOSCA files you've edited. Debug the issue by fixing the error in the TOSCA file, re-committing the changes, and creating a new pull request.

Task 2: Create a Declarative Service Template

Having your resource types defined, it is now time to implement service templates for those types. This is needed for you to be able to instantiate resources from your resource types. During this task, you learn how to implement service templates in a declarative way. In addition, you also add a VLAN resource type, which supports either custom assignment of VLAN or dynamic assignment from a pool of free VLANs. For this VLAN assignment, you learn how to utilize built-in resource types such as NumberPool and PooledNumber.

1. Navigate back to your VS Code and create a **service_template_customer.tosca** file in the same folder as the resource type definitions. Add a simple **serviceTemplates** segment to your TOSCA file. You can either add this in this newly created file or add it to the resource types file. From the BPO point of view, it does not matter in which file the definitions are located – they will all get loaded and be available for you to use.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer service template."
authors     = [ "Developer (developer@bptrn.com)" ]

serviceTemplates = {
  Customer {
    title = "Customer"
    description = "A service template for Customer Workflow and operations."
    implements = training.resourceTypes.Customer
  }
}
```

2. Next, add a service template for the bandwidth profile to **service_template_bandwidthprofile.tosca** file.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Bandwidth Profile service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the BandwidthProfile service template."
authors     = [ "Developer (developer@bptrn.com)" ]

serviceTemplates = {
  BandwidthProfile {
    title = "BandwidthProfile"
    description = "A service template for BandwidthProfile."
    implements = training.resourceTypes.BandwidthProfile
  }
}
```

3. Do the same for the port in the **service_template_port.tosca** file.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Port service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Port service template."
authors     = [ "Developer (developer@bptrn.com)" ]

serviceTemplates = {
  Port {
    title = "Port"
    description = "A service template for Port Workflow and operations."
    implements = training.resourceTypes.Port
  }
}
```

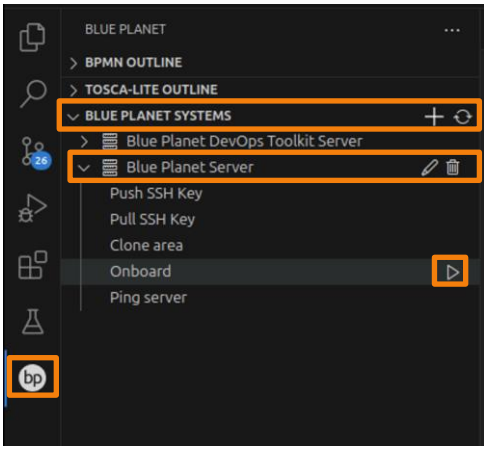
4. And for the site as well, using the **service_template_site.tosca** file.

```
"$schema"    = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Site service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Site service template."
authors     = [ "Developer (developer@bptrn.com)" ]

serviceTemplates = {
  Site {
    title = "Site"
    description = "A service template for Site Workflow and operations."
    implements = training.resourceTypes.Site
  }
}
```

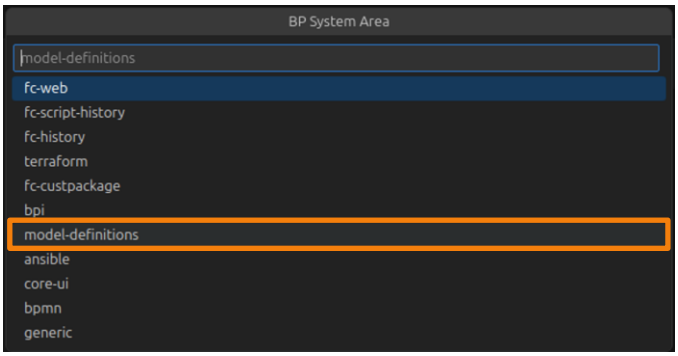
5. According to the image under Objectives, you can see that the Site resource is a parent to the Port resource, meaning the Port sub-resource should get created automatically whenever you create a Site. To make the Site create Port sub-resources, you need to add a **resources** segment under serviceTemplates. This segment should contain the *type* of the sub-resource and the *properties* that get passed from the parent resource to create it.

```
serviceTemplates = {
  Site {
    title = "Site"
    description = "A service template for Site Workflow and operations."
    implements = training.resourceTypes.Site

    resources {
      Port {
        type = training.resourceTypes.Port
        properties {
          deviceName = { getParam = deviceName }
          portName = { getParam = portName }
          portSpeed = { getParam = portSpeed }
        }
      }
    }
  }
}
```

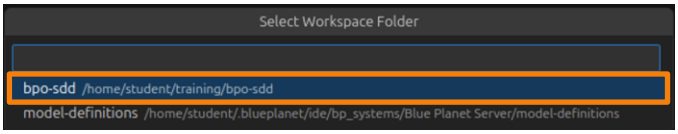
6. You have now created service templates for the customer, site, bandwidth profile, and port resources. This means that you can create these resources in BPO.
7. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension. In the BLUE PLANET SYSTEMS, expand the **Blue Planet Server** and click the **Play** button in the Onboard line



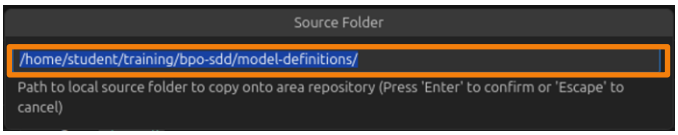
8. In the popup, select **model-definitions**.



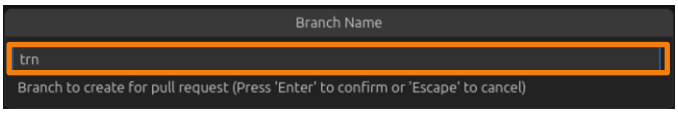
9. Next, select the **bpo-sdd** workspace folder.



10. Confirm the path to the **bpo-sdd/model-definitions** folder.

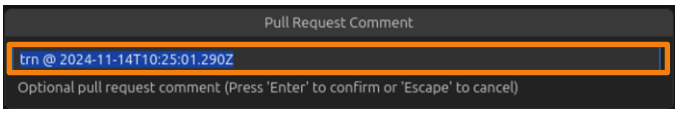


11. Set the branch name to **trn**. This branch name must match the branch name of the local model-definitions repository.

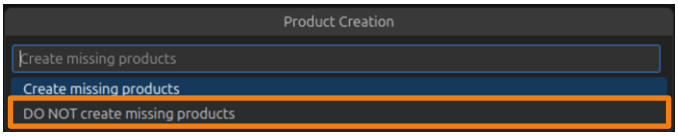


NOTE: Failing to match the two branch names will result in a failed onboarding.

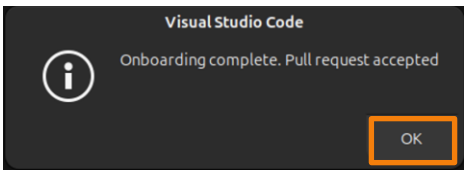
12. Confirm the default commit message (pull request comment).



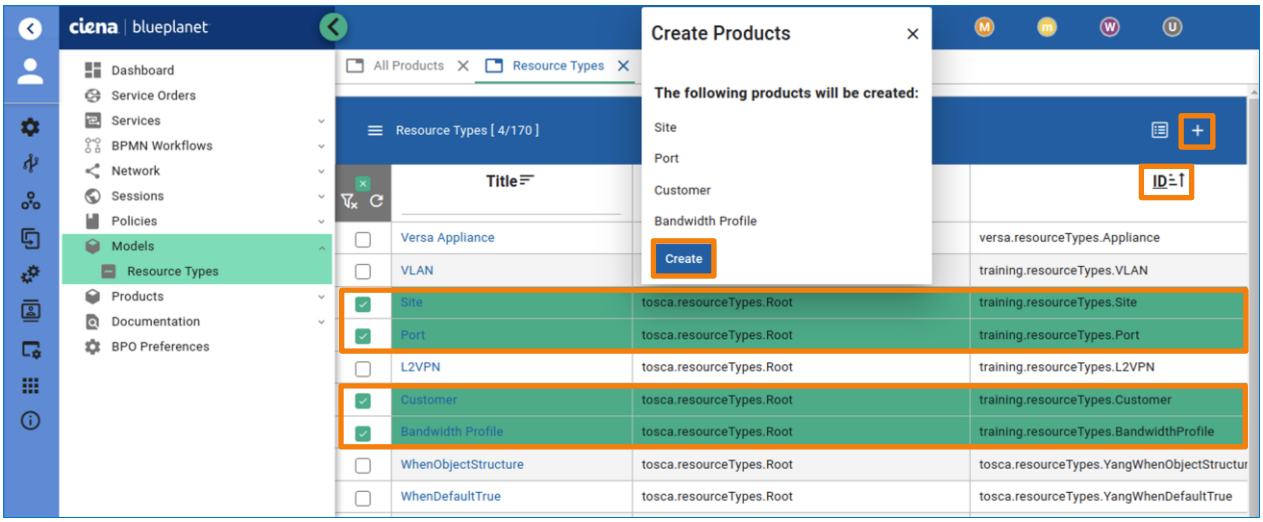
13. Select **DO NOT create missing products**.



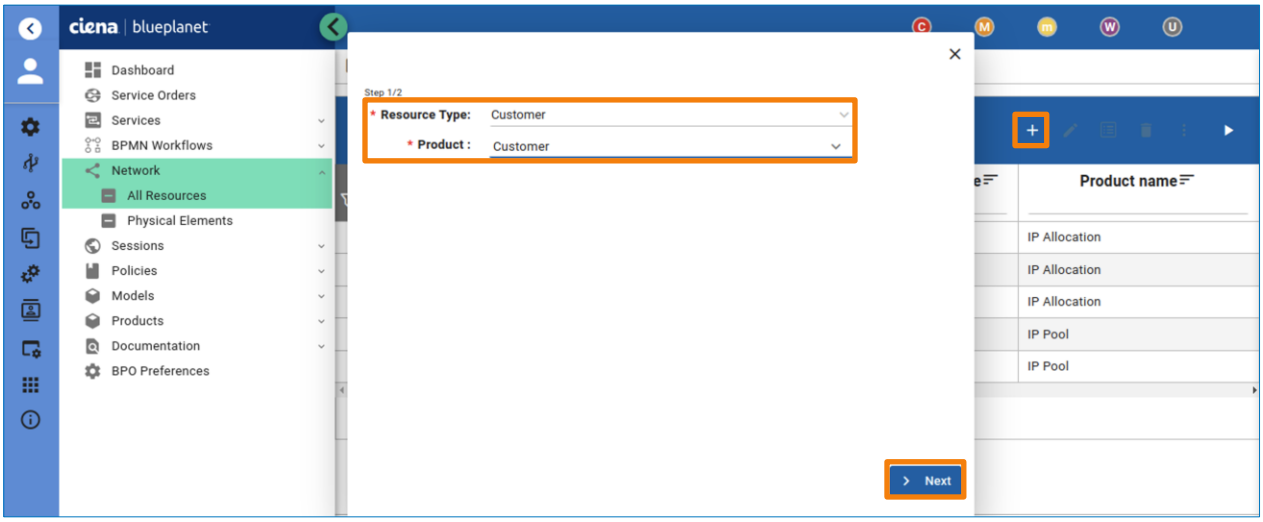
14. Confirm the popup noting that the onboarding was completed and the pull request accepted.



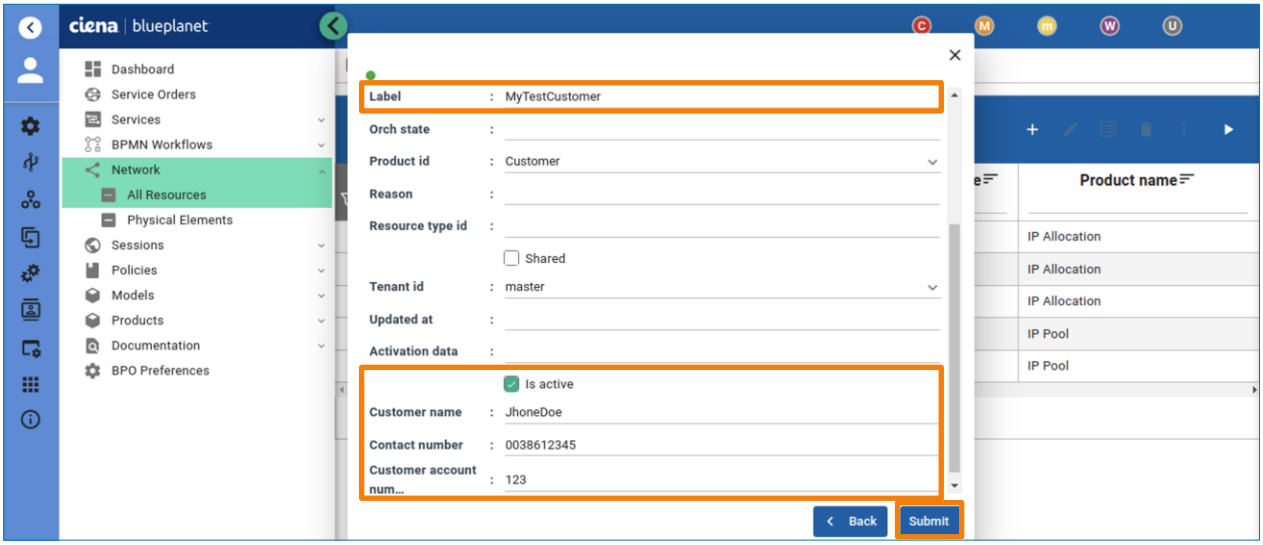
15. To create resources, you first need to create a product out of those resource definitions and service templates you've just created. Open the browser and go to **bpo.bptrn.com**. Login as **admin** with **adminpw** password.
16. Open the BPO UI, navigate to the **Models > Resource Types** and find your custom resource types. You can sort by **ID** (click on it) and look for the resource types starting with **training.resourceTypes** to find them easily.
    Choose the following resource types: **Site**, **Port**, **Customer**, and **Bandwidth Profile**, and create a product based on them by clicking the **+** symbol in the right corner.
    Choose **Create** to create the products. This operation should be successful with no errors.



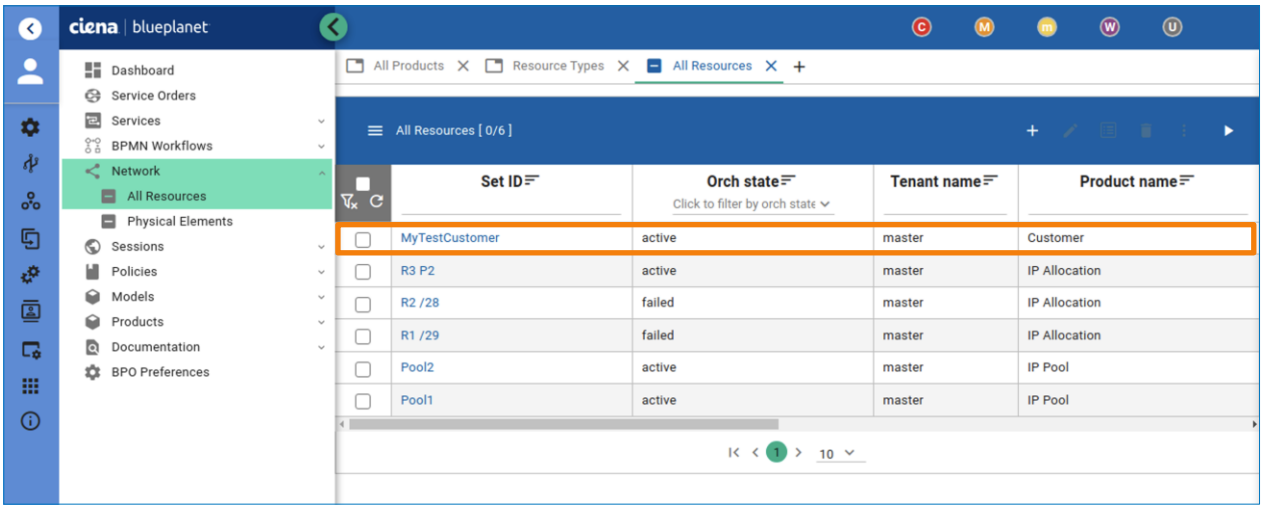
17. Try creating a Customer resource now. Navigate to the **Network > All Resources** tab. Click the **+** symbol in the right corner to create a new resource.
    Choose a **Customer** resource type based on the Customer product and click **Next**.



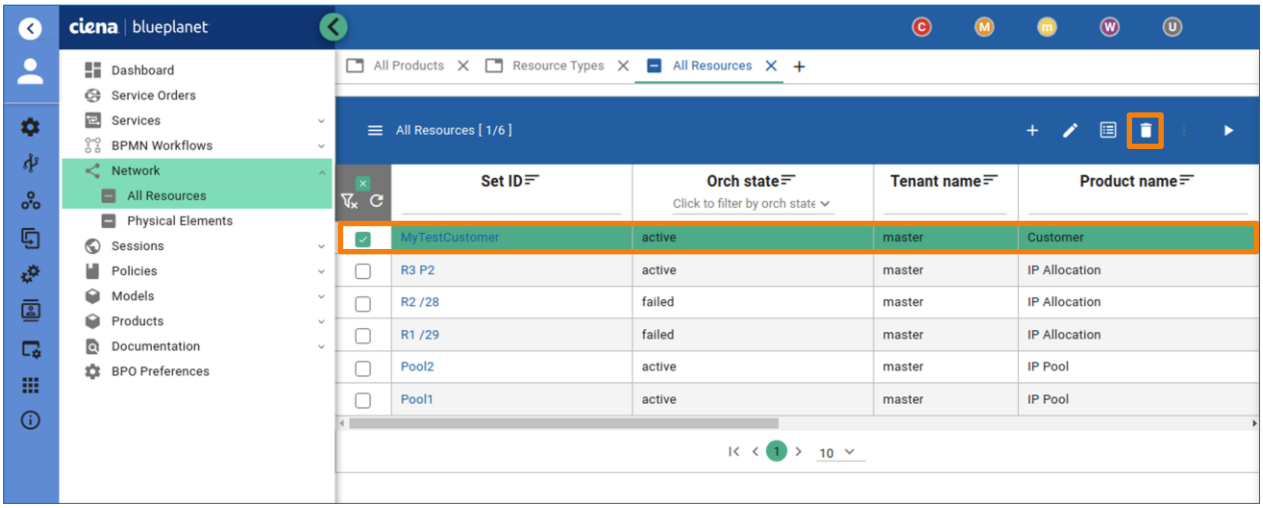
18. Fill out the following properties with arbitrary data to create a new resource:
    - Label
    - Check the **Is active** checkbox.
    - Customer Name
    - Contact Number
    - Customer Account Number
    - Click the **Submit** button to create a new customer.



19. The resource should now exist in an *active* state. If this is not the case, click on the resource and inspect the error message to debug and fix it before continuing.



20. Delete the customer resource by selecting your customer resource and clicking the **Delete** icon in the upper right corner.



21. With the products for sub-resources now created, you can focus on creating a service template for the L2VPN resource. Since you are only using that top-level resource for the creation of sub-resources, the only thing that you will have to do is map the input parameters to those sub-resources accordingly. First, create the **service_template_l2vpn.tosca** file and add a **serviceTemplates** segment.

```
"$schema"    = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN service template."
authors     = [ "Developer (developer@bptrn.com)"  ]

serviceTemplates {
  L2VPN {
    title  = "L2VPN"
    description = "A service template for L2VPN"
    implements = training.resourceTypes.L2VPN
  }
}
```

22. As is, this service will only create the l2vpn resource. To make it create sub-resources, you need to add a **resources** segment under serviceTemplates, and add a sub-resource for each resource you want to create by specifying their type. Add the **Customer**, **Site**, and **Bandwidth Profile** sub-resources. There is no need to add a Port as well since it will get created automatically when you create a Site.

```
serviceTemplates {
  L2VPN {
    title  = "L2VPN"
    description = "A service template for L2VPN"
    implements = training.resourceTypes.L2VPN

    resources = {
      Customer {
        type = training.resourceTypes.Customer
      }

      Site {
        type = training.resourceTypes.Site
      }

      BandwidthProfile {
```

```
        type = training.resourceTypes.BandwidthProfile
      }
    }
  }
}
```

23. These sub-resources also require some input parameters for them to be created. Since you have already defined these parameters in the l2vpn resourceType, you only need to map them to sub-resources using directives such as *getParam*, which returns the value of the specified parameter. Make sure you also pass the Port parameters to the Site resource, since they are used in the creation of the Port sub-resource by the Site.

```
serviceTemplates {
  L2VPN {
    title  = "L2VPN"
    description = "A service template for L2VPN Workflow and operations."
    implements = training.resourceTypes.L2VPN

    resources = {
      Customer {
        type = training.resourceTypes.Customer
        properties {
          customerName = { getParam = customerName }
          customerAccountNumber = { getParam = customerAccountNumber }
          contactNumber = { getParam = contactNumber }
          isActive = { getParam = isActive }
        }
      }

      Site {
        type = training.resourceTypes.Site
        properties {
          siteId = { getParam = siteId }
          deviceName = { getParam = deviceName }
          portName = { getParam = portName }
          portSpeed = { getParam = portSpeed }
        }
      }

      BandwidthProfile {
        type = training.resourceTypes.BandwidthProfile
        properties {
          cir = { getParam = cir }
          eir = { getParam = eir }
          cbs = { getParam = cbs }
          ebs = { getParam = ebs }
        }
      }
    }
  }
}
```

24. Next, take care of the VLAN assignment for your L2VPN service. The idea is to add a **vlanId** parameter to the top-level resource, which can be either input manually when creating the resource or generated dynamically if the input field is left empty. First, view the **resource_type_l2vpn.tosca** file and check the **VLAN** resource type with the **vlanId**(integer) property to it.

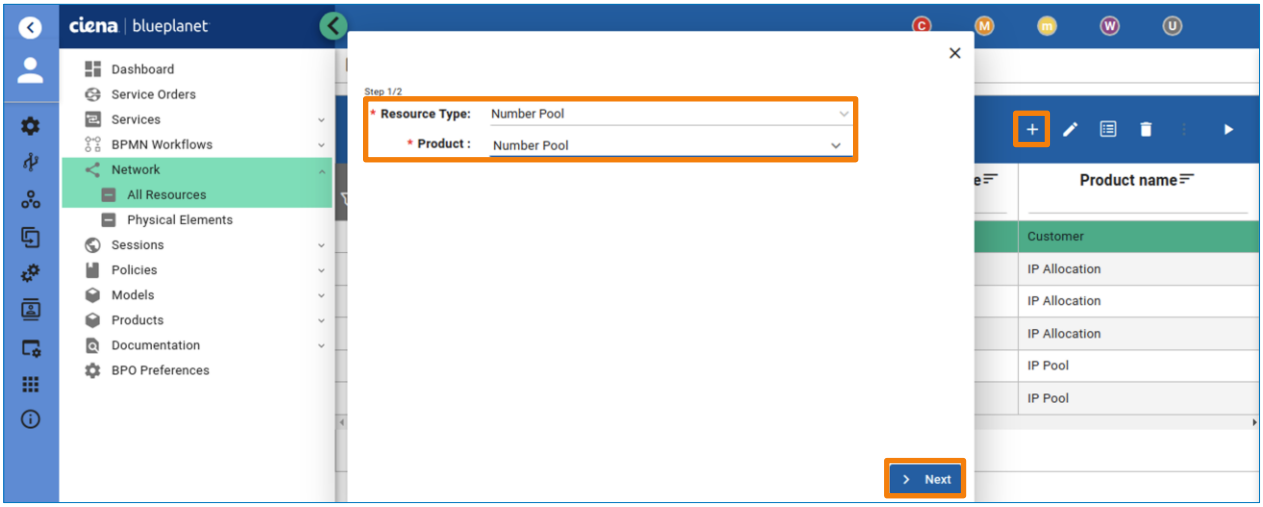
```
resourceTypes {
  VLAN {
    derivedFrom = Root
    title = "VLAN"
    description = "Designated VLAN"
    properties {
      vlanId {
        title = "VLAN"
        description = "Specify or allocate the VLAN to be used by the L2VPN"
```

```
        type = integer
      }
    }
  }
...
```

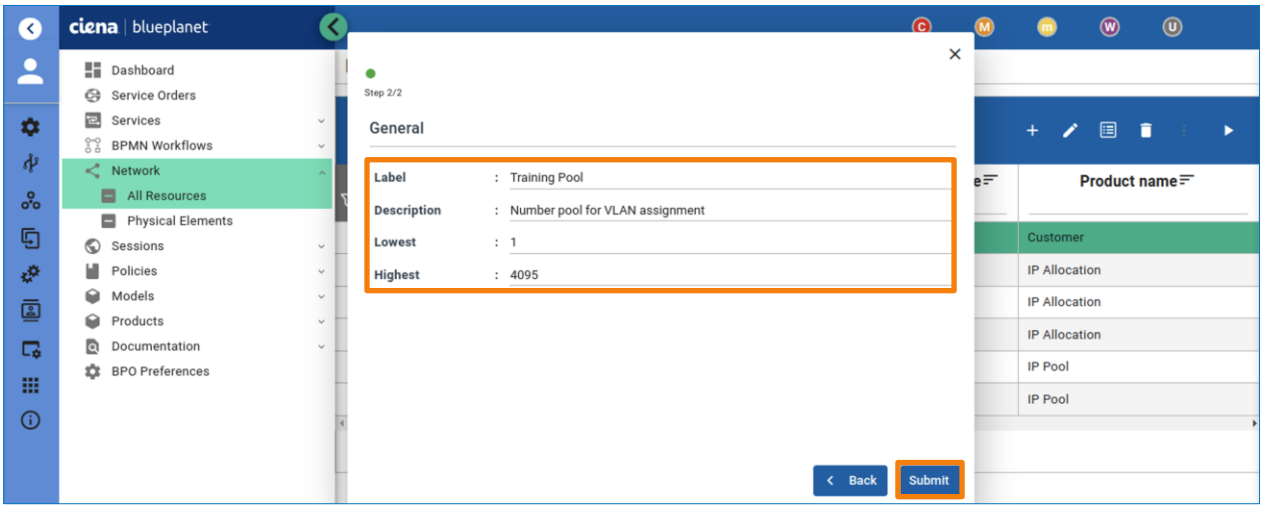
25. Add a basic service template for VLAN as well to the **service_template_l2vpn.tosca** file.

```
serviceTemplates {
  VLAN {
    title  = "VLAN"
    description = "A service template for VLAN"
    implements = training.resourceTypes.VLAN
  }

  L2VPN {
    title  = "L2VPN"
    description = "A service template for L2VPN Workflow and operations."
    implements = training.resourceTypes.L2VPN

    resources = {
      VLAN {
        type = training.resourceTypes.VLAN
      }

      Customer {
```

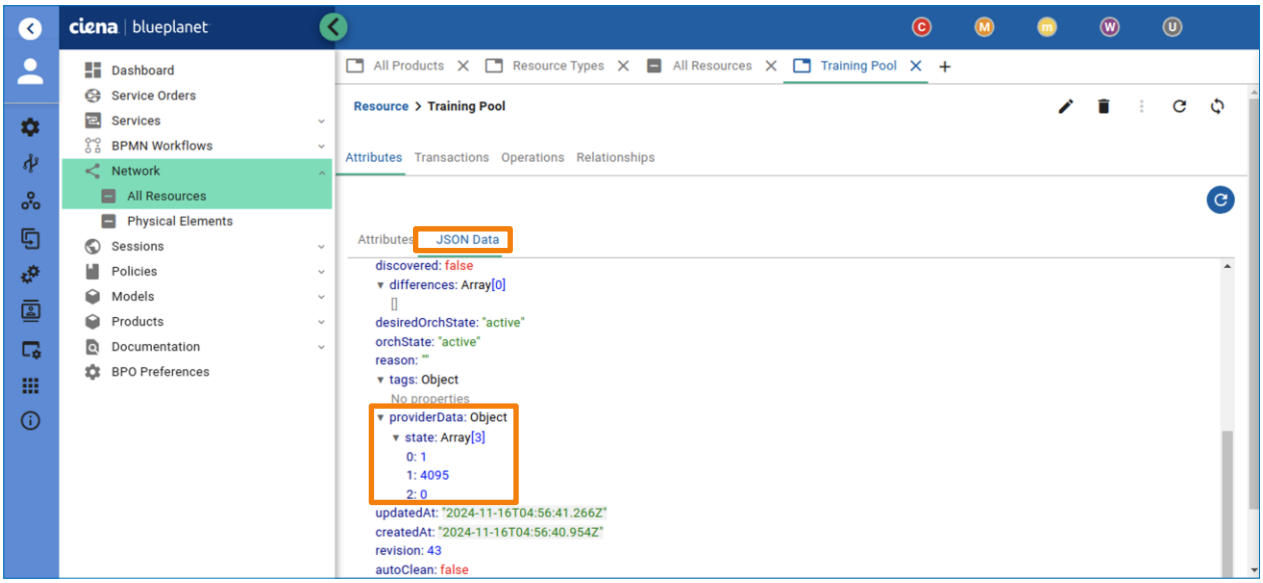
26. Now you need to create a pool of numbers, that will allow you to assign sequential available numbers that will represent VLANs. Open the **BPO UI** and navigate to the **Network > All Resources** tab. Click the **+** (plus) button. Choose **Number Pool** as your desired resource type.



27. Use **Training Pool** as the name of the pool. This is important since you will use the name to specify the pool in your declarative plans. Use the range of 1-4095 as the standard VLAN pool range. Click **Submit** and make sure that the Number Pool has been successfully created – the state should be marked as *active*.



28. Click on the pool, navigate to **Attributes > JSON Data** and inspect it. Here, you can see the properties that you specified for the pool range and a bunch of different IDs.



NOTE: The NumberPool could also be created through the declarative plan. You will create the number pool programmatically in one of the following labs.

29. The Number Pool resource can create PooledNumber resources (type tosca.resourceTypes.PooledNumber). These are resources containing a single positive integer, allocated from the pool whose domain you specify. This integer should then be used in the creation of the VLAN resource. However, the user should have the option to bypass this VLAN assignment from the pool and specify a VLAN ID if they wish. Add a **PooledNumber** sub-resource to the resources segment in the **service_template_l2vpn.tosca** file.

```
L2VPN {
  title = "L2VPN"
  description = "A service template for L2VPN"
```

```
    implements = training.resourceTypes.L2VPN

    resources = {
      PooledNumber {
          type = tosca.resourceTypes.PooledNumber
      }

      VLAN {
...
```

30. You now need to make sure that the PooledNumber resource is created within the domain of the Training Pool resource. You can do this by specifying a **product** segment under your sub-resource and setting the **domain** parameter within to the ID of the desired domain. Study the documentation for template directives and form a plan on how to use these directives for this use case.
31. There are multiple solutions to this problem. For example, you can find the Training Pool resource, read its ID, get the domain of the resource with this ID, and finally read the ID of the domain. Such a solution can be written as follows. Make sure that the label matches the one you set on your Number Pool.

```
      PooledNumber {
          type = tosca.resourceTypes.PooledNumber
          product { domain = { getDomain = { getResourceWith = { label = "Training Pool" } } } }
      }
```

32. Having added these statements, you now force the creation of a PooledNumber resource every time you create an L2VPN resource. However, you only want this to happen if the vlanId parameter is empty. This can be solved using the **createIf** directive.

```
       PooledNumber {
          type = tosca.resourceTypes.PooledNumber
          product { domain = { getDomain = { getResourceWith = { label = "Training Pool" } } } }
          createIf = { ifElse = [ { getOptionalParam = vlanId }, false, true] }
      }
```

33. Finally, you want to assign either this allocated VLAN number, or manually set the VLAN number to the **vlanId** property of the VLAN sub-resource. For this, you need to use the **ifElse** directive and read the **vlanId** parameter as an optional input parameter if it was set or read the **allocatedValue** parameter from your PooledNumber sub-resource.

```
      VLAN {
        type = training.resourceTypes.VLAN
        properties {
          vlanId = { ifElse = [ { getOptionalParam = vlanId },
                                { getOptionalParam = vlanId },
                                { getAttr = [ PooledNumber , allocatedValue ] } ] }
        }
      }
```

34. Additionally, the order is important! While most of the sub-resources can be created in parallel, the VLAN sub-resource depends on a PooledNumber sub-resource being created if it must allocate a VLAN from the pool. You can set the order using the **activateAfter** section under your sub-resource and specify dependent sub-resources there.

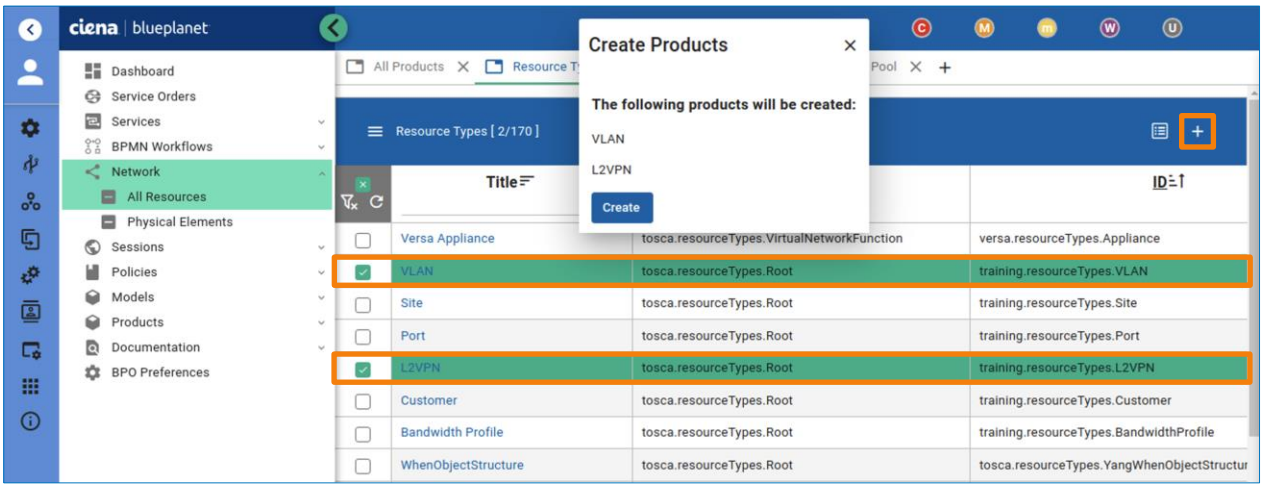
```
      VLAN {
        type = training.resourceTypes.VLAN
        properties {
          vlanId = { ifElse = [ { getOptionalParam = vlanId },
                                { getOptionalParam = vlanId },
                                { getAttr = [ PooledNumber , allocatedValue ] } ] }
        }
        activateAfter = [ PooledNumber ]
      }
```

35. This is how the **service_template_l2vpn.tosca** file should look like in the end.

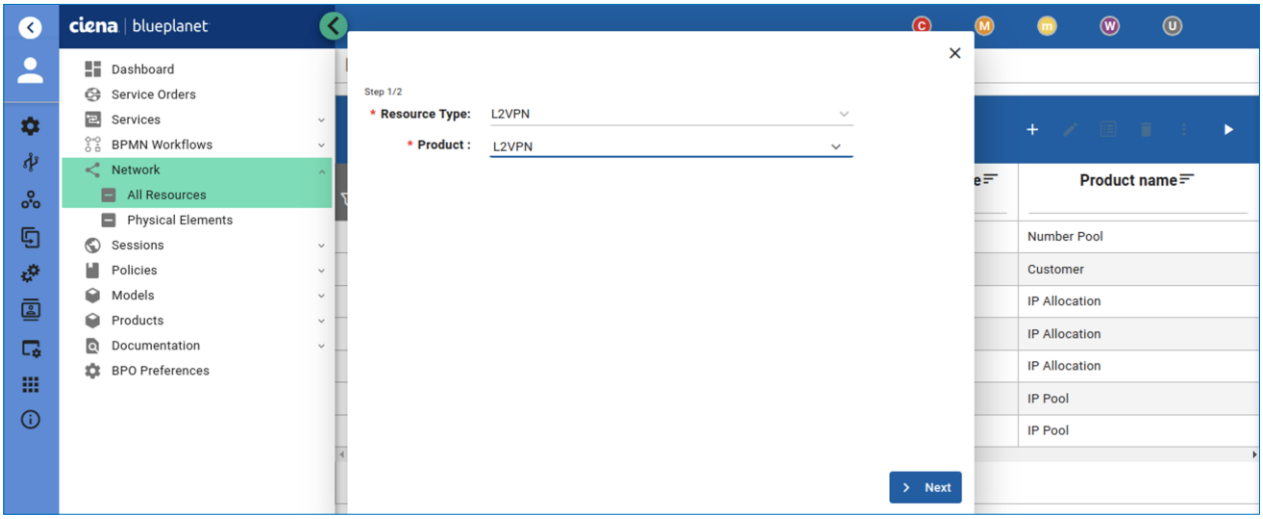
```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN service template."
authors     = [ "Developer (developer@bptrn.com)"  ]

serviceTemplates {
  VLAN {
    title = "VLAN"
    description = "A service template for VLAN."
    implements = training.resourceTypes.VLAN
  }

  L2VPN {
    title = "L2VPN"
    description = "A service template for L2VPN."
    implements = training.resourceTypes.L2VPN

    resources = {
      PooledNumber {
          type = tosca.resourceTypes.PooledNumber
          product { domain = { getDomain = { getResourceWith = { label = "Training Pool" } } } }
          createIf = { ifElse = [ { getOptionalParam = vlanId }, false, true ] }
      }

      VLAN {
        type = training.resourceTypes.VLAN
        properties {
          vlanId = { ifElse = [ { getOptionalParam = vlanId },
                                { getOptionalParam = vlanId },
                                { getAttr = [ PooledNumber , allocatedValue ] } ] }
        }
        activateAfter = [ PooledNumber ]
      }

      Customer {
        type = training.resourceTypes.Customer
        properties {
          customerName = { getParam = customerName }
          customerAccountNumber = { getParam = customerAccountNumber }
          contactNumber = { getParam = contactNumber }
          isActive = { getParam = isActive }
        }
      }

      Site {
        type = training.resourceTypes.Site
        properties {
          siteId = { getParam = siteId }
          deviceName = { getParam = deviceName }
          portName = { getParam = portName }
          portSpeed = { getParam = portSpeed }
        }
      }

      BandwidthProfile {
        type = training.resourceTypes.BandwidthProfile
        properties {
          cir = { getParam = cir }
          eir = { getParam = eir }
          cbs = { getParam = cbs }
          ebs = { getParam = ebs }
        }
      }
```

```
    }
  }
}
```

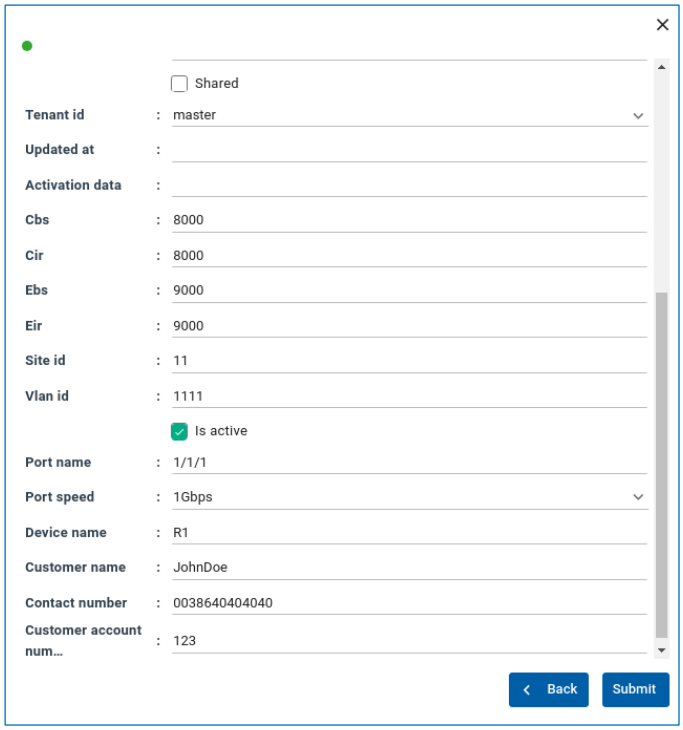
36. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
37. In the BPO UI, create the products out of the two remaining resource types that you have not created yet (L2VPN, VLAN). There should be six products starting with **training.resourceTypes** under your **Products** tab now.



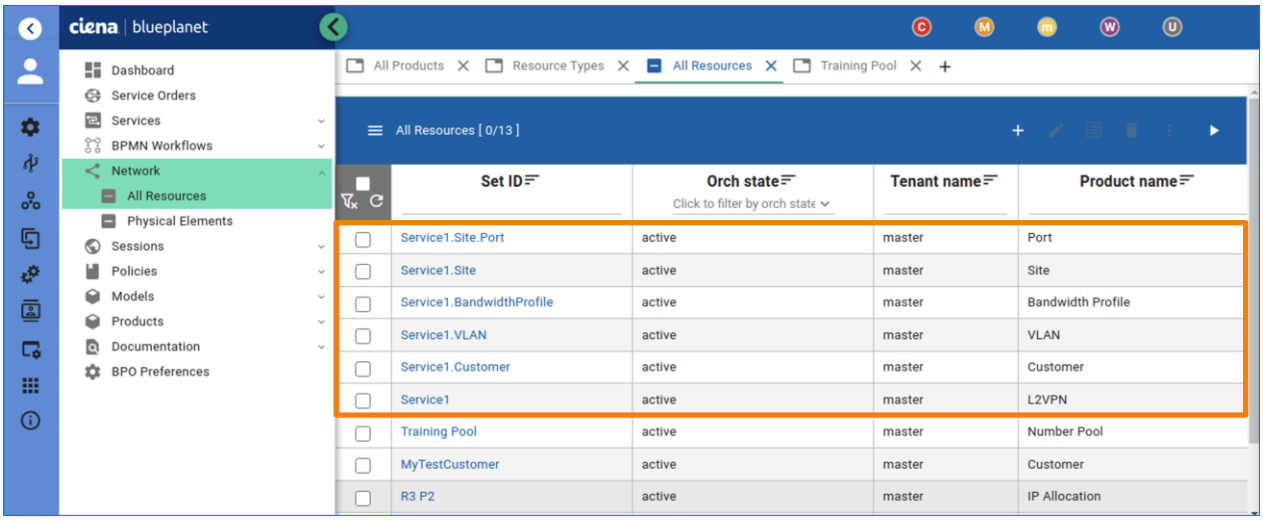
38. You can now finally test out your L2VPN template which uses a declarative plan to implement the resources. Navigate to the **Network > All Resources** tab and make sure that the Training Pool resource, which you created a few steps back is present there.
39. You will now create two L2VPN resources – one where you manually set the VLAN and one where you let the VLAN be allocated from the Number Pool.
40. Create the first **L2VPN** resource by choosing the **+** symbol in the right corner of the resources table and choosing the **L2VPN** resource type and **L2VPN** product.



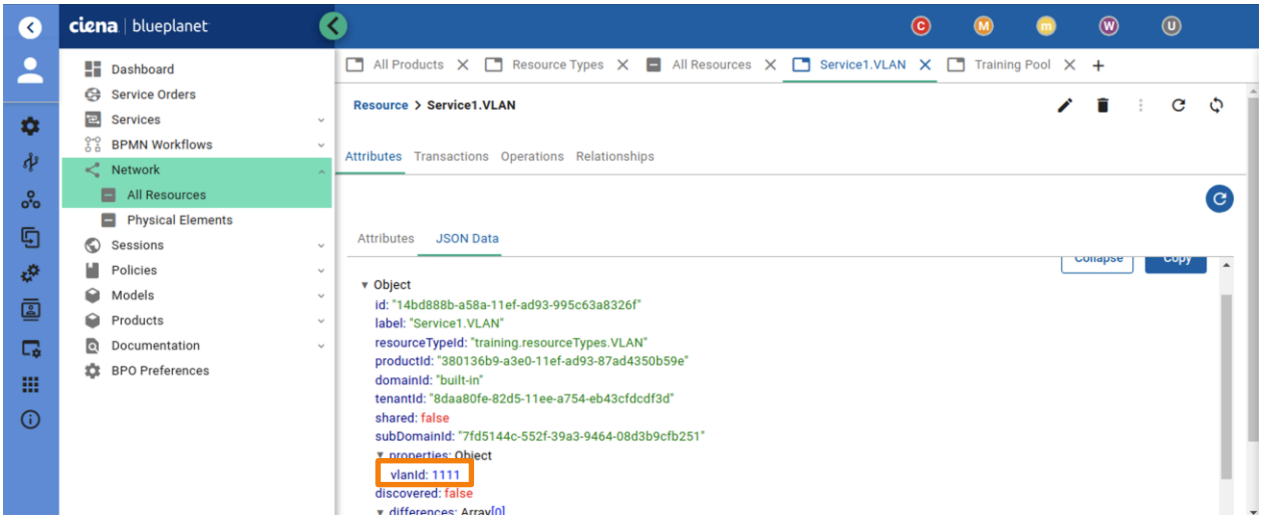
Fill in the required parameters. For the label, set **Service1**. For the **Vlan id** parameter, enter an arbitrary integer.



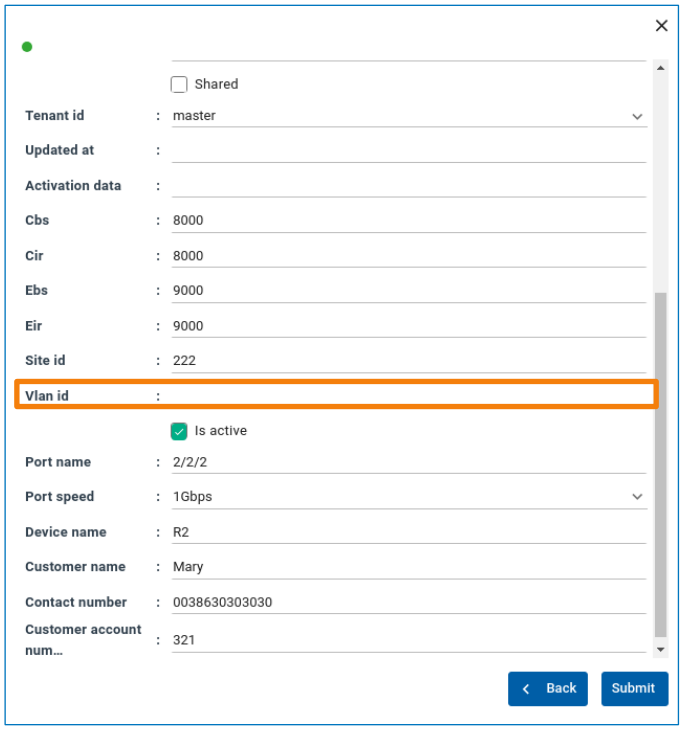
41. Six new resources should be created – L2VPN, VLAN, Customer, BP, Port, and Site. Make sure all of them are in *an active* state. If not, click on them, inspect the error message, and fix the errors.



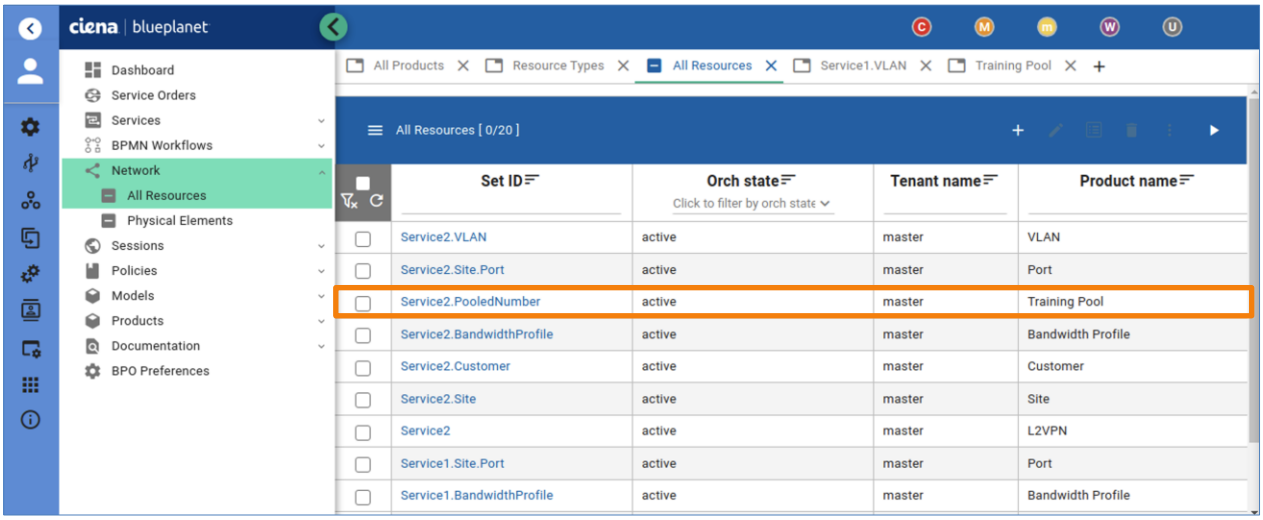
42. Inspect the **Service1.VLAN** resource and its **JSON Data**. You will notice that the vlanId parameter there is the same as the one you entered in the resource creation form.



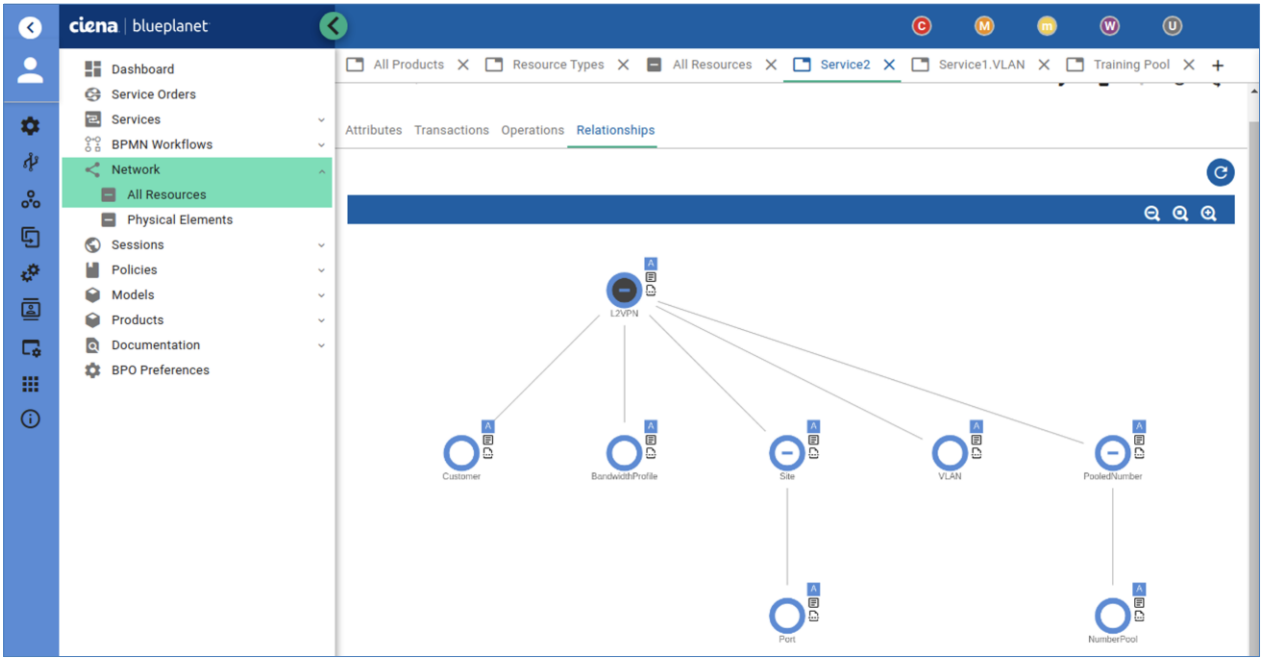
43. Create the second **L2VPN** resource named **Service2** (label). This time, leave the **Vlan id** parameter empty.



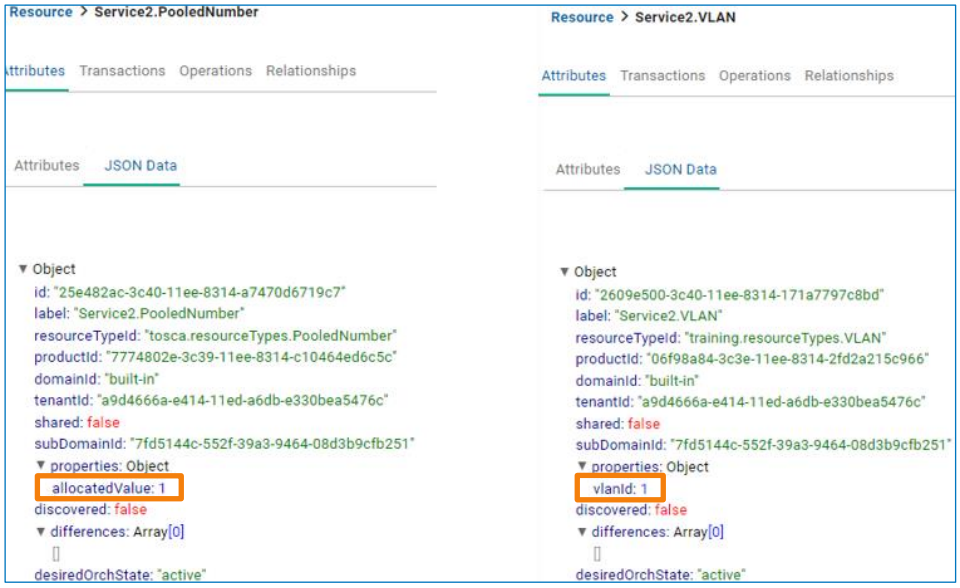
44. \In addition to the resources that were created with the first resource, you can now also notice the *Service2.PooledNumber* resource being created.



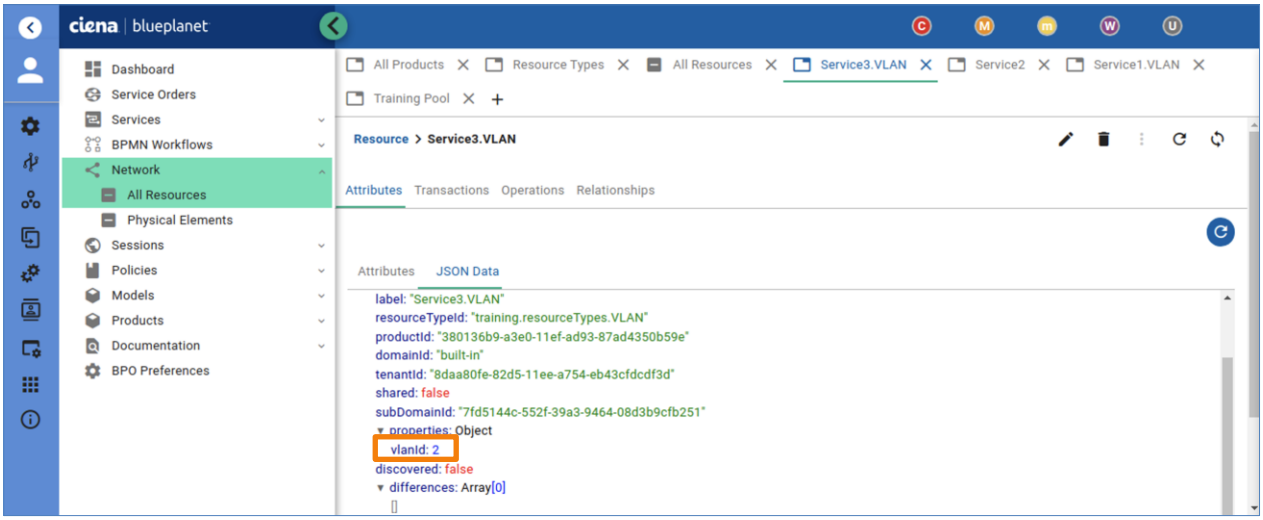
45. Open the **Service2** resource. Observe the **Relationship** tab under the L2VPN resource. It will show you what the relationships between resources look like.



46. Compare the JSON parameters from the *Service2.PooledNumber* resource and the *Service2.VLAN* resource. Both have the same number allocated (in this case – *1*).



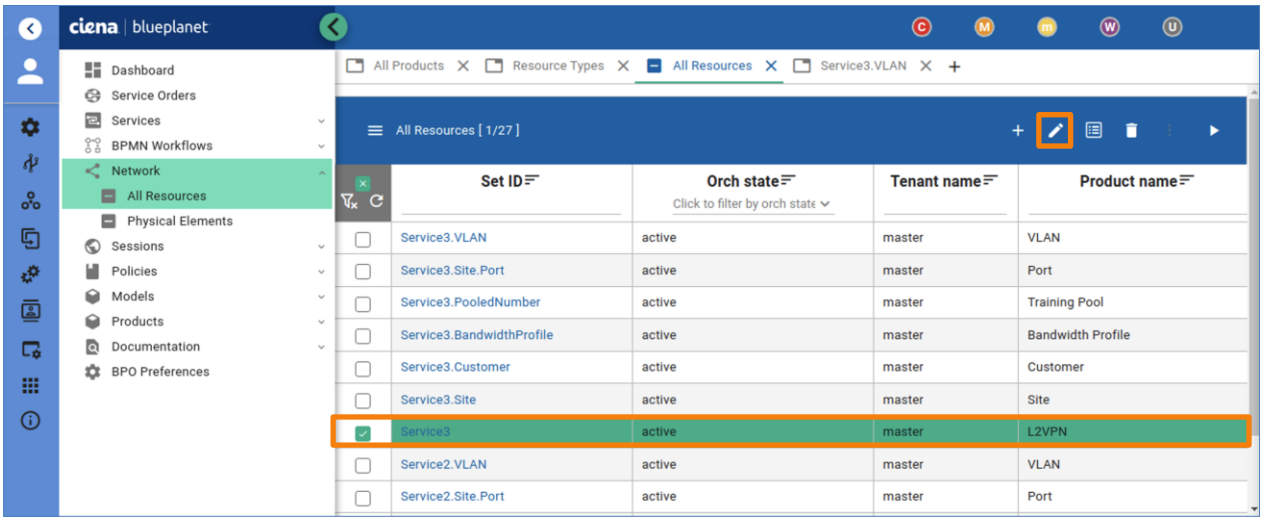
47. To confirm that the number allocation works, create a third L2VPN resource – **Service3**, which also uses dynamic VLAN allocation. You can expect the next available number (in this case – *2*) to be allocated next.



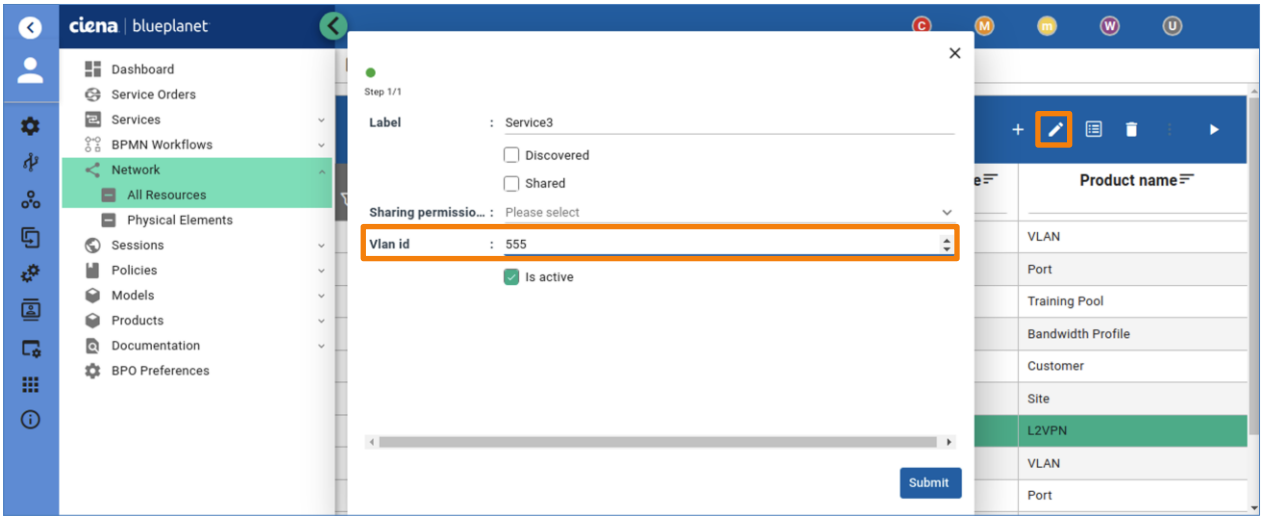
48. Sometimes, some parameters for a network service need to be updated as well. In this case, it is required for any property to use the **updateable = true** statement. Modify the **resource_type_l2vpn.tosca** file and set the updateable property to both **VLAN:vlanId** and **L2VPN:vlanId** properties.

```
resourceTypes {
  VLAN {
    derivedFrom = Root
    title = "VLAN"
    description = "Designated VLAN"

    properties {
      vlanId {
        title = "VLAN"
        description = "Specify or allocate the VLAN to be used by the L2VPN"
        type = integer
        updatable = true
      }
    }
  }

  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {
      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
        updatable = true
      }
```

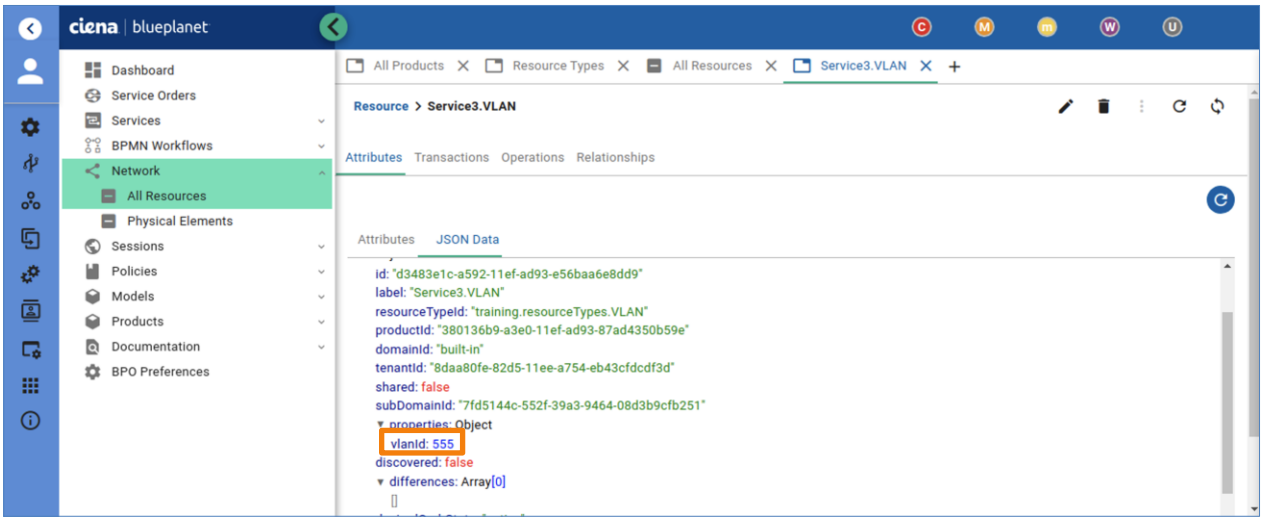
49. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
50. Update the **vlanId** property for the **Service3** resource by choosing it and clicking the **Edit** button in the upper-right corner.



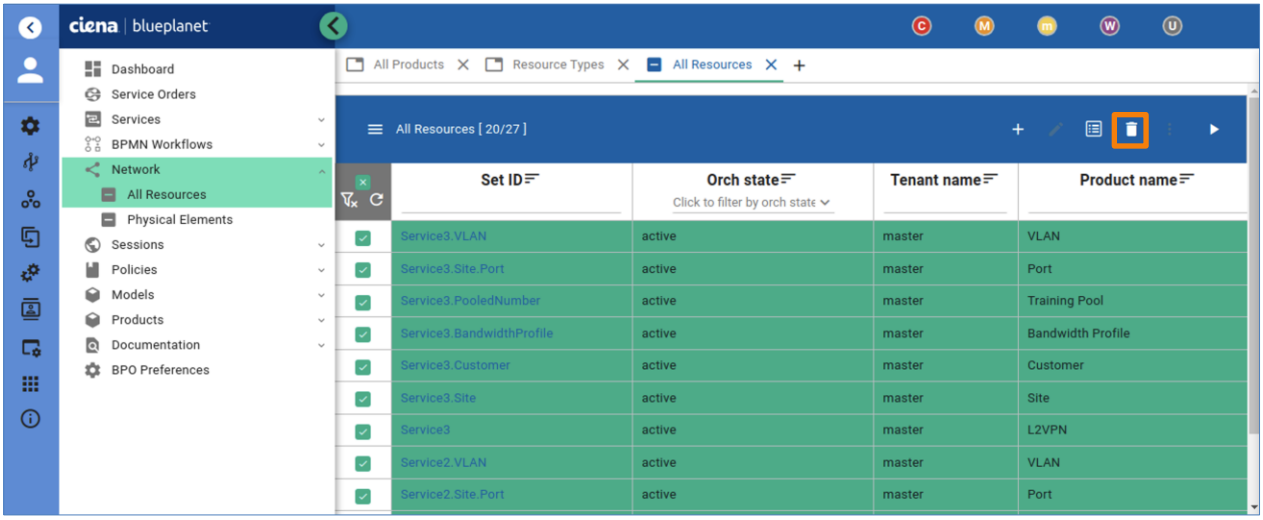
51. Set the **vlanId** property to an arbitrary number and click **Submit** to update the resource**.**



52. Inspect the resources again. The Service3.PooledNumber resource has disappeared (refresh the Resources table if not), showing that declarative plans automatically take care of update and delete actions as well. This is a big benefit, especially when compared to the imperative plans. Inspect the **Service3.VLAN** resource's JSON data. The manually set VLAN should be visible there.



53. Delete **all** the resources (**Service1**, **Service2**, **Service3** and **MyTestCustomer**). To delete resources, simply choose all the resources that you want to delete and click the **Delete** button on the top right. Due to BPO trying to delete all resources at once, you might get an error warning you about not being able to delete some resource because it has dependencies. In this case, just try again, since these dependencies will not be deleted.



NOTE: To force the deletion of a resource with active dependencies, you can use the BPO API. This is very useful for development, but can have unintended side effects, so avoid using it in production. To force delete a resource, use the **Market PATCH /resources/{resourceId}** API call with the following payload:

```
{
  "desiredOrchState": "terminated",
  "orchState": "terminated"
}
```

End of Lab

Lab 3: Create Default Service Lifecycle Operations

Objectives

- Update the L2VPN service with an imperative template using default operations
- Create the required Python code to implement these imperative plans
- Learn how to utilize built-in resource types using PlanSDK

Documentation

TOSCA definitions - *https://developer.blueplanet.com/docs/bpocore-docs/references/type_layer/README.html*

PlanSDK documentation - https://developer.blueplanet.com/docs/plansdk/index.html

Lifecycle operations - *https://developer.blueplanet.com/docs/bpocore-docs/references/lifecycle_operations.html*

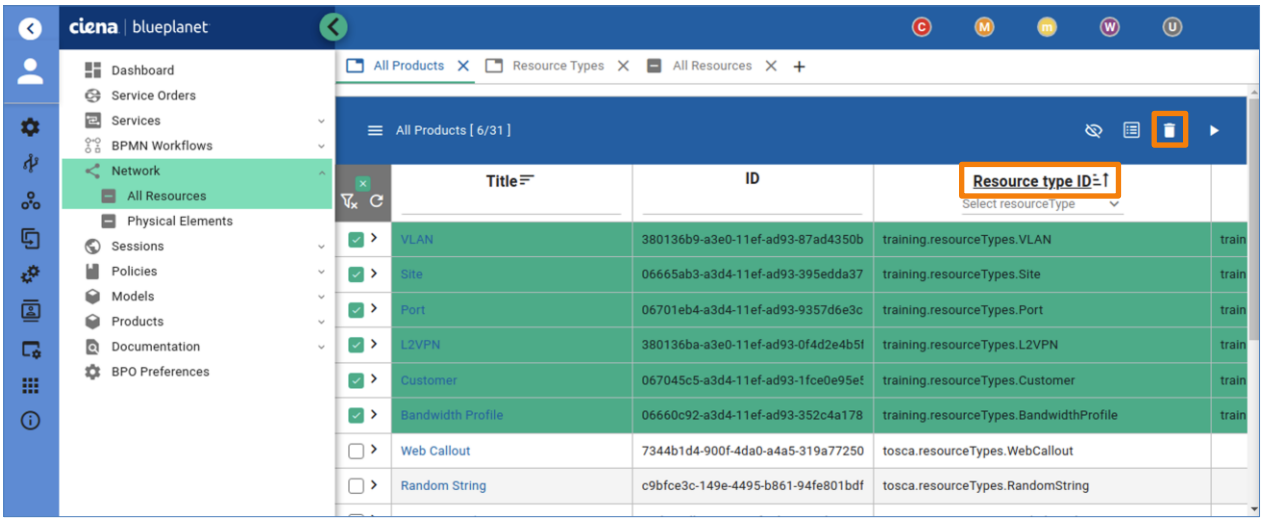
Task 1: Update Service Templates with an Imperative Plan

In this initial task, you update your existing service templates so that they use an imperative plan instead of a declarative one.

1. This lab continues the work done in the previous labs. Make sure that you have successfully completed the previous lab. If you did not complete the previous lab, make sure to onboard the resource definitions, service templates, and code from that lab to bring your BPO server into the initial state required for this lab. You can find them in the **~/Desktop/learning/bpo-sdd/solutions** folder.
2. First, ensure all the resources in the BPO UI, under **Network > All Resources**, are *deleted*. You will be modifying existing service templates and resource types, which can cause unintended consequences and backward incompatible changes on existing resources. You will learn exactly how to handle such changes in one of the following labs.

   NOTE: Refer to the last step from the previous lab for deleting the resources.

3. Also, you need to delete all the Products you have created from your training resource types. Go to **Products > All Products** and delete all the products that use the **training.resourceTypes.*** ID. You will re-create them later.



4. You now need to add imperative plan definitions to the service template files. These plan definitions get added under a specific resource's service templates with a *plans* segment. Each plan interface represents a specific lifecycle or a custom operation within BPO. In this task, you focus on creating the default operations – *activate*, *terminate*, and in some cases *update* as well.
5. In VS Code, open the **service_templates_customer.tosca** file. Add a **plans** object. The object should contain *activate* and *terminate* plans. Each plan contains a reference to a script, which implements the operation. Set the type of script to **remote**, and the language to **python.** As for the path, the parameter specifies the fully qualified Python module name of the script in the Model Definitions area, relative to the root of the area. This means, that if you have, for example, a file called **customer.py** in the **training/model-definitions/training** folder, which contains the *Activate* class, you define the path as **training.customer.Activate**.

```
serviceTemplates = {
  Customer {
    title = "Customer"
    description = "A service template for Customer."
    implements = training.resourceTypes.Customer

    plans {
      activate {
        type = remote
        language = python
        path = training.customer.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.customer.Terminate
      }
    }
  }
}
```

6. Add the **activate** and **terminate** plans for the bandwidth profile to the **service_template_bandwidthprofile.tosca** next.

```
serviceTemplates = {
  BandwidthProfile {
    title = "BandwidthProfile"
    description = "A service template for BandwidthProfile"
    implements = training.resourceTypes.BandwidthProfile

    plans {
      activate {
        type = remote
        language = python
        path = training.bandwidthprofile.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.bandwidthprofile.Terminate
      }
    }
  }
}
```

7. Do the same for the port service template in the **service_template_port.tosca** file.

```
serviceTemplates = {
  Port {
    title = "Port"
    description = "A service template for Port Workflow and operations."
    implements = training.resourceTypes.Port

    plans {
      activate {
        type = remote
        language = python
        path = training.port.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.port.Terminate
      }
    }
  }
}
```

8. In the service template **service_template_site.tosca** file **delete resources** segment. The file should look like this.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Site service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Site service template."
authors     = [ "Developer (developer@bptrn.com)" ]

serviceTemplates = {
  Site {
    title = "Site"
    description = "A service template for Site Workflow and operations."
    implements = training.resourceTypes.Site
  }
}
```

9. In the same **service_template_site.tosca** file, also add the **activate** and **terminate** plans.

```
serviceTemplates = {
  Site {
    title = "Site"
    description = "A service template for Site Workflow and operations."
    implements = training.resourceTypes.Site

    plans {
      activate {
        type = remote
        language = python
        path = training.site.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.site.Terminate
      }
    }
  }
}
```

10. Finally, do the same for the *L2VPN* resource in the **service_template_l2vpn.tosca** file. First delete any declarative plans for resource creation, as well as the VLAN resource.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN service template."
authors     = [ "Developer (developer@bptrn.com)"  ]

serviceTemplates {
  L2VPN {
    title = "L2VPN"
    description = "A service template for L2VPN."
    implements = training.resourceTypes.L2VPN

  }
}
```

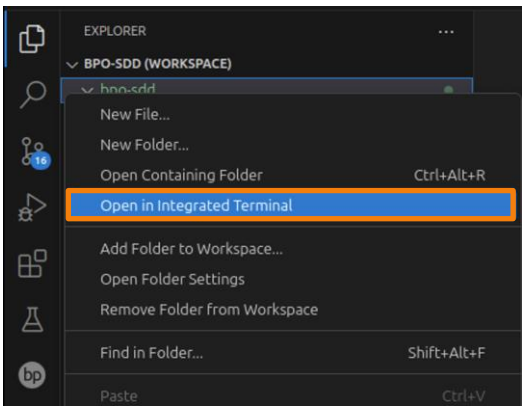
11. Then add the **activate** and **terminate** plans to the **service_template_l2vpn.tosca** file. This is how your file should look like.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN service template."
authors     = [ "Developer (developer@bptrn.com)"  ]

serviceTemplates {
  L2VPN {
    title = "L2VPN"
    description = "A service template for L2VPN."
    implements = training.resourceTypes.L2VPN

    plans {
      activate {
        type = remote
        language = python
        path = training.l2vpn.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.l2vpn.Terminate
      }
    }
  }
}
```

Task 2: Create Virtual Environment

In this task you will create a Python virtual environment, which provides isolation for projects, preventing dependency clashes and enabling experimentation without affecting system-wide configurations. It also ensures reproducibility by maintaining consistent development environments across different machines and team members. In the next labs you will leverage this environment for having code competition in VS Code for Python packages that you will get from the Git repository.

1. In VS code, right click the **bpo-sdd** folder and select **Open in Integrated Terminal**.



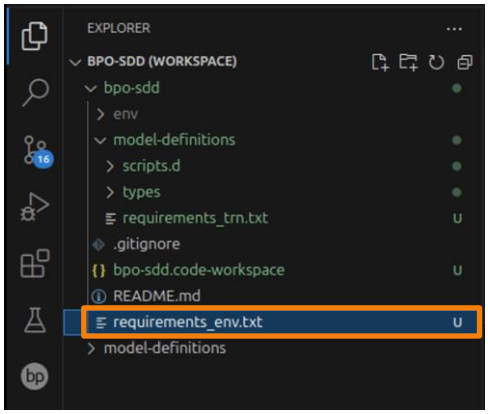
   In the integrated terminal enter **python3.10 -m venv env** to create a virtual environment.

```
student@POD-XX:~/training/bpo-sdd$ python3.10 -m venv env
```

2. Next, you will notice the environment is active by the **(env)** at the beginning of the prompt in the terminal. If not, activate the environment with the **source env/bin/activate** command.

```
student@POD-XX:~/training/bpo-sdd$ source env/bin/activate
(env) student@POD-XX:~/training/bpo-sdd$
```

3. Create a **requirements_env.txt** file in the **bpo-sdd** folder.



4. Open the created **requirements_env.txt** file and add the following lines to it. Save the file.

```
-i https://pypi.org/simple
--extra-index-url https://gitlab.bptrn.com/api/v4/projects/2/packages/pypi/simple
plansdk
```

NOTE: The -i (same as --index-url) defines the base URL if the Python package index, while the --extra-index-url defines the package index to use in addition to the base index. The plansdk package is on the GitLab repository and will be installed into the virtual environment.

5. Back in the integrated terminal, install the packages into the virtual environment. Use the **pip install -r requirements_env.txt** command.

```
(env) student@POD-XX:~/training/bpo-sdd$ pip install -r requirements_env.txt
Looking in indexes: https://pypi.org/simple,
https://gitlab.bptrn.com/api/v4/projects/2/packages/pypi/simple
...
```

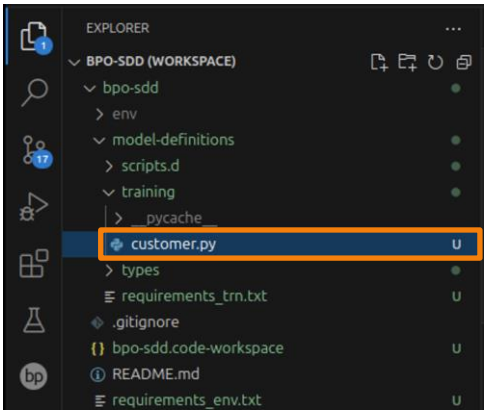
NOTE: You will get some warnings about untrusted repository which is expected. As long as there are no errors, the packages should be installed.

6. Now the virtual environment is ready. You will add it to the workspace in the next task, when you will start creating the Python files.

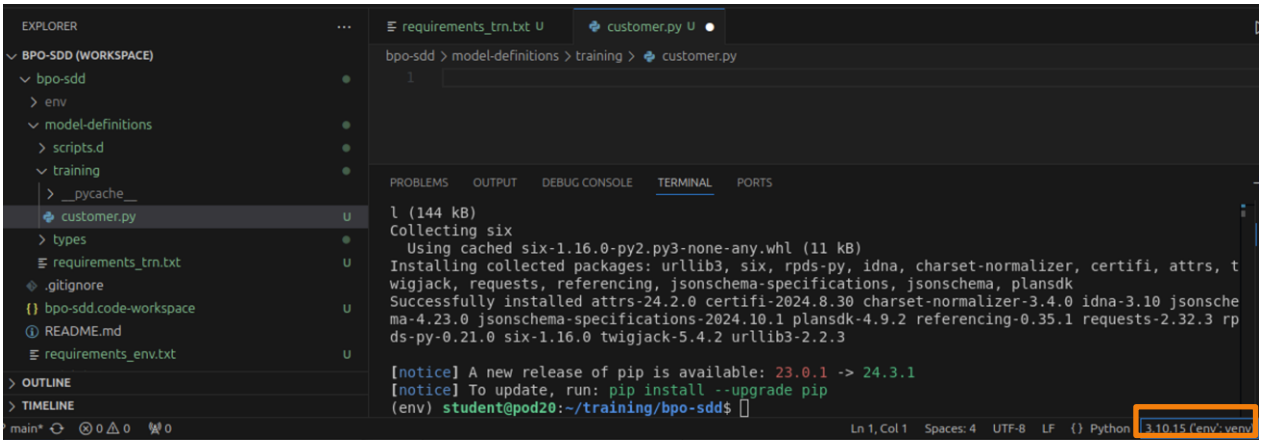
Task 3: Implement the Imperative Plans Using Python Scripts

In this task, you focus on implementing your imperative plans using remote Python scripts.

1. Start with the Customer's Activate and Terminate plans. Since you defined the path as **training.customer.Activate and training.customer.Terminate,** you need to create a Python file under the **bpo-sdd/model-definitions/training** folder that is called **customer.py**. First create the folder, then the python file.

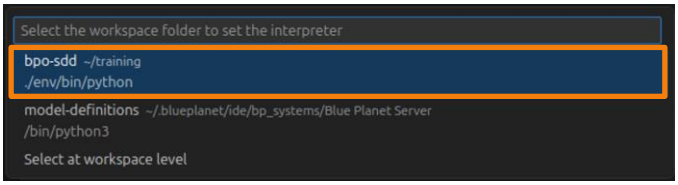


2. Set the Python Interpreter. In the bottom right corner of VS Code, click on **3.10.x**.

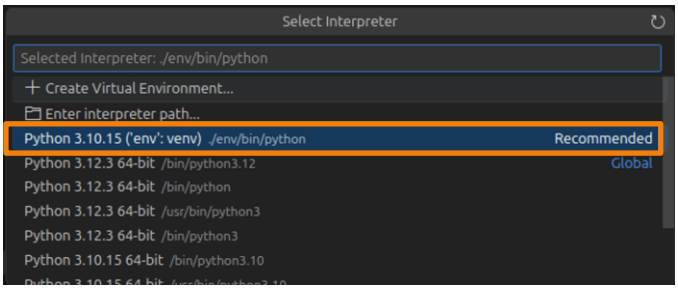


NOTE: The interpreter currently used will show up only if you have a Python file open.

3. From the drop-down menu select **bpo-sdd**.



4. Then select the Python Interpreter in the **env** folder (**./env/bin/python**).



5. Now you will have the autocomplete and documentation options in VS Code for the plansdk package that you have installed in the previous task.
6. In the **customer.py** file, add empty **Activate** and **Terminate** classes to this file. You can also add some comments to the classes – this example is using the docstring format. In fact, it's good practice to add such comments to any and all classes you implement.

```
class Activate():
    """
    Creates Customer resource in the market
    """

class Terminate():
    """
    Removes Customer resource from the market
    """
```

7. Now just defining classes will not work. When writing and executing these Python scripts, a lot of parameters and metadata are passed down from the BPO to the script execution engine. These parameters are inherited from the *Plan* superclass, which is a part of the *plansdk* framework, and is a Blue Planet proprietary Python package. You can read more about it using the link provided in the *Documentation* section of this lab.
   Modify the **customer.py** code so that you import the **Plan** superclass from the **plansdk.apis.plan** module and use it to initialize the **Activate** and **Terminate** subclasses**.**

```
from plansdk.apis.plan import Plan

class Activate(Plan):
    """
    Creates Customer resource in the market
    """

class Terminate(Plan):
    """
    Removes Customer resource from the market
    """
```

8. Each Plan subclass also expects a **run()** method. This method overrides the default run method from the Plan superclass. In other terms, this is where you write the code that implements specific operations. Similarly, you could use the **setup()** class for modifying the global state before the **run()** is run, or the **teardown()** for after the **run()** method. However, in this case, only the **run()** method will be needed.

```
from plansdk.apis.plan import Plan

class Activate(Plan):
    """
    Creates Customer resource in the market
    """
    def run(self):

class Terminate(Plan):
    """
```

```
    Removes Customer resource from the market
    """
    def run(self):
```

9. You do not need to take care of the resource creation – this happens automatically. You do, however, need to take care of the resource activation and provide the code for that operation. In the case of a customer, no additional code for activation is needed since you do not create any relationships or other resources when creating a Customer resource. For this reason, you simply log the new resource's parameters..

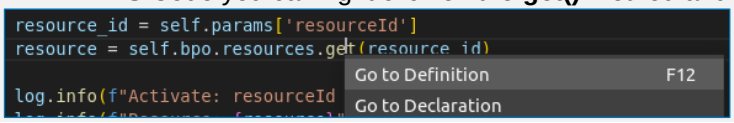
   First, import the **logging** module, set the log variable to the default logger.

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Creates Customer resource in the market
    """
    def run(self):

class Terminate(Plan):
    """
    Removes Customer resource from the market
    """
    def run(self):
```

10. For informational purposes, it is a good idea to log the input parameters that were used in the creation of the resource. You can access these resources by using the **self.params** object. You can then read the **resourceId** from those parameters and use it to read and log the Customer resource through the **self.bpo.resource** class (open the plansdk documentation link in the Documentation section of this lab and see how the **self.bpo.resource.get()** method works).

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Creates Customer resource in the market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Activate: DONE")

class Terminate(Plan):
    """
    Removes Customer resource from the market
    """
    def run(self):
```

HINT: In VS Code you can right click on the **get()** method and select **Go to Definition**.



11. The script engine executes these plans and monitors them for when they are completed. If the script exits with a zero status code, the script engine returns a successful result to BPO, together with the return value of the run() method. Non-zero status codes indicate failure, and the script engine returns a failed result with the sys.stderr output as the error message. Additionally, the **run()** method must return either a JSON encoded mapping for a Python dictionary structure, or **None** – meaning you can have a void method that returns nothing, and it will still be a viable return. Since you are still learning, return an empty dictionary at the end of the Activate **run()** method.

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Creates Customer resource in the market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Activate: DONE")
        return {}

class Terminate(Plan):
    """
    Removes Customer resource from the market
    """
    def run(self):
```

NOTE: This return dictionary can be used to pass additional data (for example, about a validation error) back to BPO as you will learn in one of the following steps.

12. Next, still in **customer.py** file, add a similar code to the **Terminate run()** method that will log the input parameters, resourceId, and the resource used in the terminate operation.

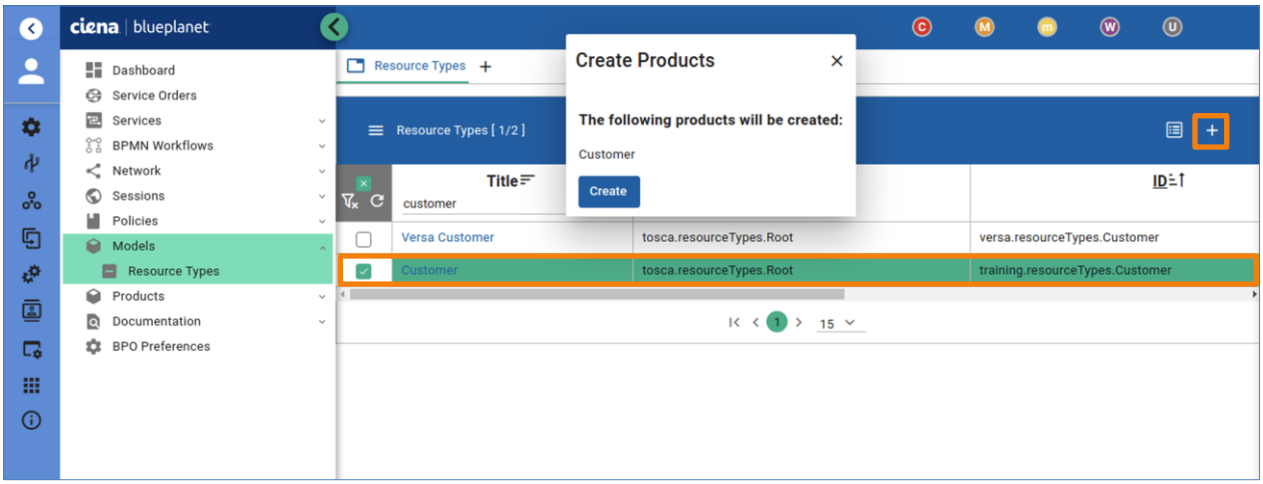
```
...

class Terminate(Plan):
    """
    Verify no other resources are present as dependents of the Customer resource
    Remove resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Terminate: DONE")
        return {}
```

13. Think about the edge cases of the terminate operation. When should a resource not be deleted? The answer is – when there are still resources that have dependencies on your resource. Because of this, you need to add an additional check to the Terminate operation. This check should verify whether dependencies for this resource exist and prevent termination in such cases (hint - open the **plansdk** documentation link in the *Documentation* section of this lab and see how the **self.bpo.resource.get_dependencies()** method works).

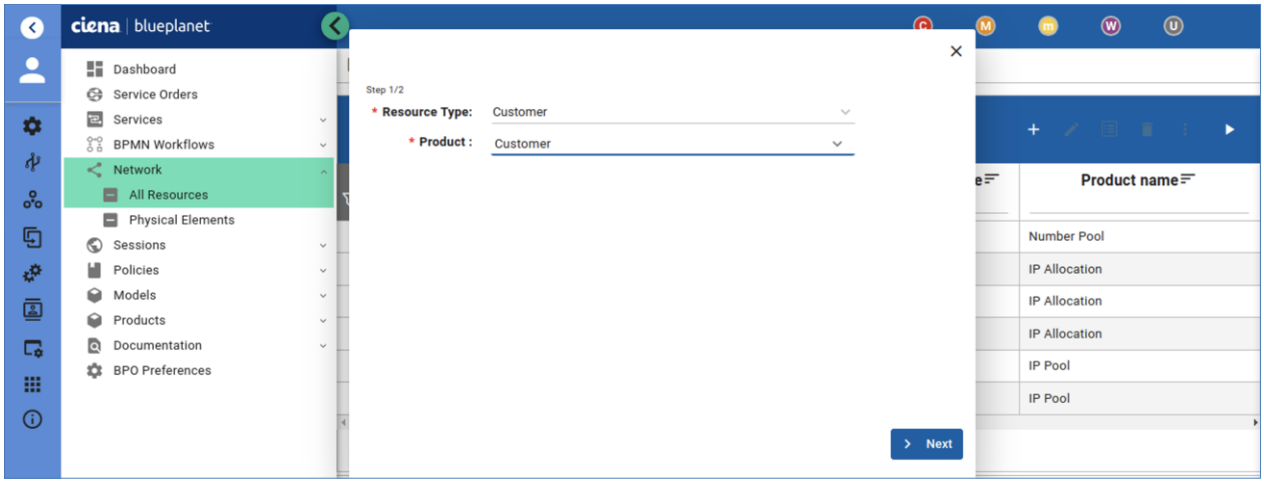
```
...

class Terminate(Plan):
    """
    Verify no other resources are present as dependents of the Customer resource
    Remove resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Customer has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

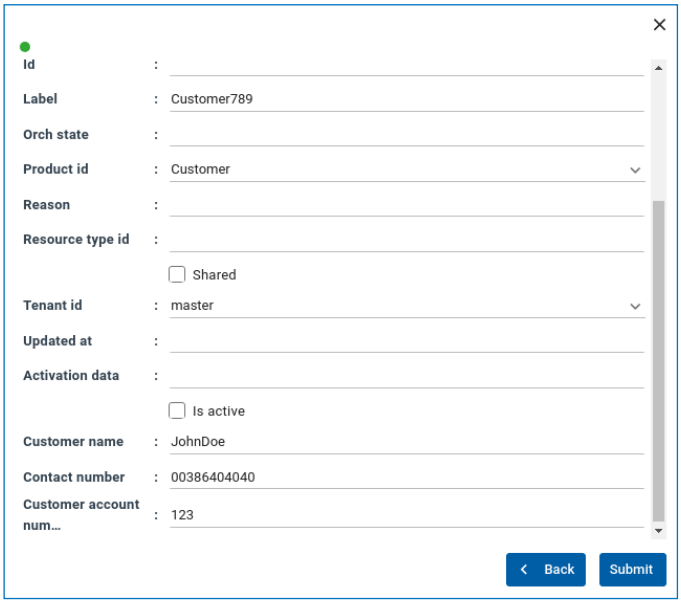
NOTE: As you can see in the documentation, most of the resources are referenced by their ID when dealing with them programmatically. This is because resource names can be duplicated, while IDs cannot.

14. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
15. In BPO-UI, create a product out of the Customer resource type. Got to **Models > Resource Types**. Find the **Customer** resource type, choose it, and click the **+** icon in the upper-right corner to create the related product.

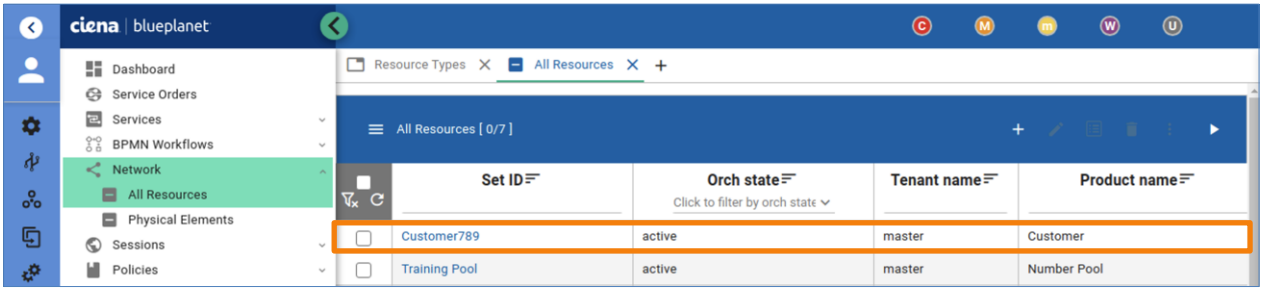


16. Create a new Customer resource to test your imperative plan remote scripts.





17. Make sure that the resource is in an *active* state.



18. Delete this Customer resource. Since no dependencies to the Customer have been implemented yet, the resource should be deleted with no issues. In the case of errors, fix the errors by clicking on the resource, inspecting the error message, and troubleshooting it.
19. You can only see the logs of remote script execution in the UI for Resources if something fails. If you wish to examine the full logs of each remote script you need to search for them either in the built-in monitoring tool Kibana or in the appropriate Docker container (or Kubernetes pod, depending on what distribution of BPO platform you are using).
20. To inspect the logs in the docker container, connect to your BPO server over SSH with user **bpuser** and password **bpuserpw**.

```
student@POD-XX:~/training/trn-pypi$ ssh bpuser@bpo.bptrn.com
Last login: Tue Feb 13 20:05:24 2024 from 10.154.20.181
[bpuser@BPO-XX ~]$
```

21. You can connect to the Docker container's shell. This canbe done with **solman** – the Blue Planet solution manager. Enter the solman CLI with the **solman** command.

```
[bpuser@BPO-XX ~]$ solman
Connecting smcli to solutionmanager_0

(Cmd)
```

22. Connect to the **scriptplan** container's shell by using **enter_container** command.

```
(Cmd) enter_container scriptplan_2.2.17-23.08-1_0
root@f735a73c82b9:/bp2/src#
```

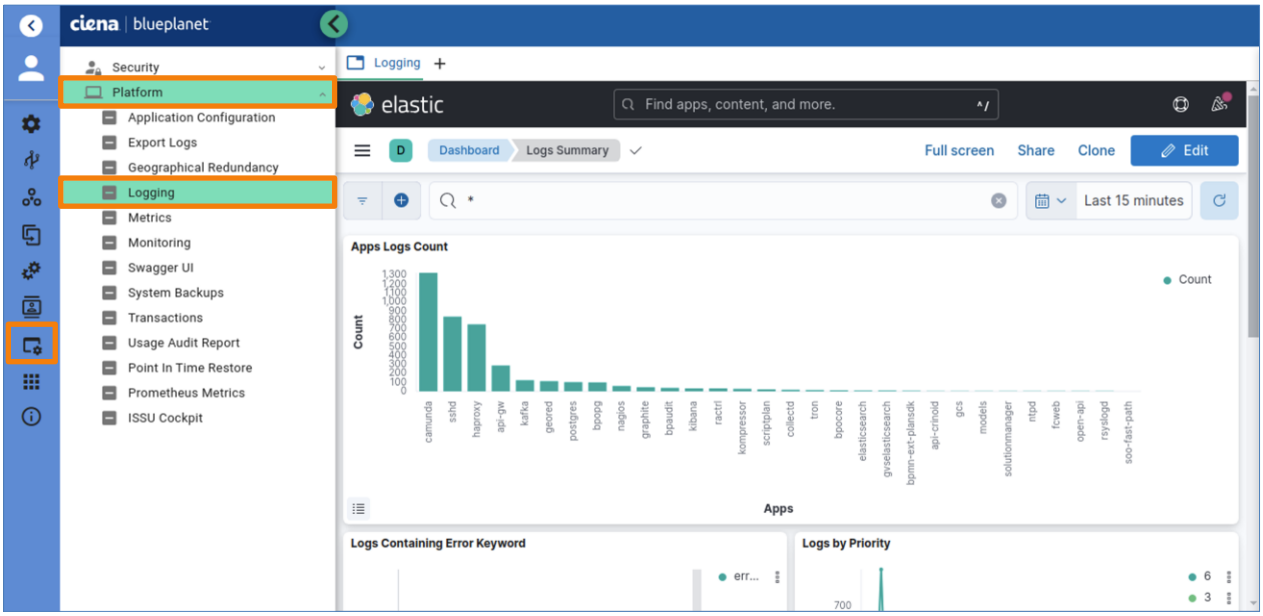
23. Navigate to the **/bp2/log/plan-log** directory and list the files there using the **ls-tr** (*-t* sort by time, *-r* reverse) command. This is the directory that stores the logs from Python remote script executions. At least 2 log files should be present there – one for the activation script and one for the deletion script. You can see which lifecycle operation got activated by the prefix in the log name (in other words, activate / terminate).

```
root@f735a73c82b9:/bp2/src# cd /bp2/log/plan-log/
root@f735a73c82b9:/bp2/log/plan-log# ls -tr
...
plan-script-83448837-a730-11ef-ad93-3be9759f2657-83543fa8-a730-11ef-ad93-db1be16e6a38-activate-2024-11-
20T11:13:47.642Z
plan-script-83448837-a730-11ef-ad93-3be9759f2657-74a5ba8a-a738-11ef-ad93-434039555fb2-terminate-2024-11-
20T12:10:38.990Z
```

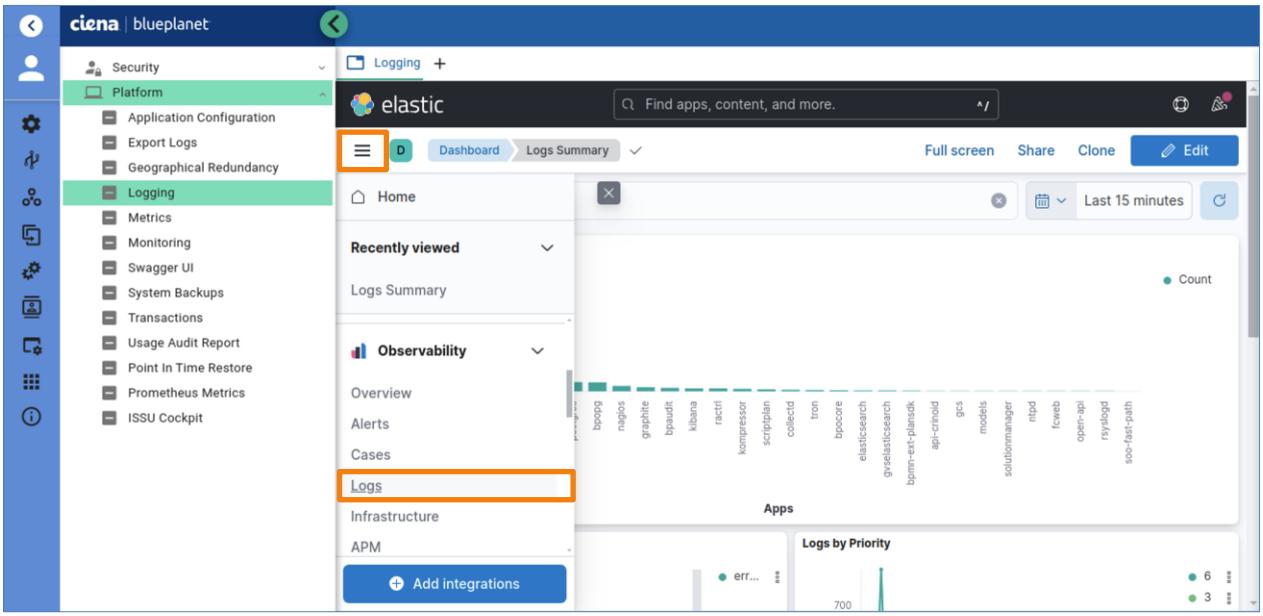
24. Open the activate log and find where the input parameters, resourceId, and resource are printed out.

```
root@c93616c3c285:/bp2/log/plan-log# cat plan-script-83448837-a730-11ef-ad93-3be9759f2657-83543fa8-a730-
11ef-ad93-db1be16e6a38-activate-2024-11-20T11:13:47.642Z
024-11-20 11:13:47,958 training.customer INFO Activate: Input params: {'marketwatchUrl':
'http://127.0.0.1:40797', 'marketwatchFeatures': ['resource1', 'resourceOperation1', 'minimum1'],
'operation': 'activate', 'operationId': '83543fa8-a730-11ef-ad93-db1be16e6a38', 'resourceId': '83448837-
a730-11ef-ad93-3be9759f2657', 'requestId': '387cc7cb-7bb1-41a0-9cab-e6ffec747653', 'bpUserId': '780b5348-
ec14-4511-a2eb-06ed163d7277', 'bpTenantId': '8daa80fe-82d5-11ee-a754-eb43cfdcdf3d', 'bpRoleId': 'f498faa2-
82d5-11ee-8629-dbc3340a1ee0,f4989904-82d5-11ee-8629-8be165298959,f499b5c8-82d5-11ee-8629-
1303168fa1c6,f497a206-82d5-11ee-8629-5b4616beb2e0,f49953bc-82d5-11ee-8629-131952f652ad,f886d9c9-9ff6-4436-
b8bc-492cea9b47d1,f49a5ad2-82d5-11ee-8629-53a6bb73224e,f49a0cf8-82d5-11ee-8629-f3ec6879ff25', 'logFile':
'/bp2/log/plan-log/plan-script-83448837-a730-11ef-ad93-3be9759f2657-83543fa8-a730-11ef-ad93-db1be16e6a38-
activate-2024-11-20T11:13:47.642Z', 'bpSubdomainId': '', 'uri': 'http://blueplanet', 'traceId': '3db9992a-
d8eb-413b-bc4a-ea9d07ab85dd'}
...
```

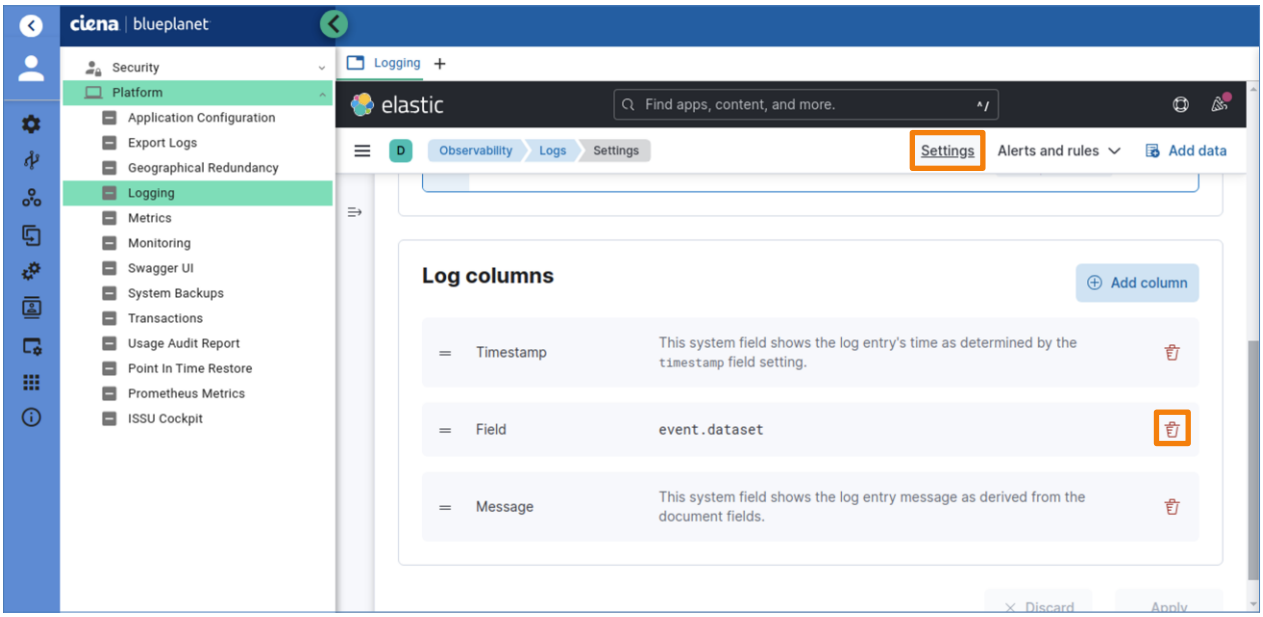
25. Another option is to inspect the logs which are aggregated on the BPO platform itself. Open the UI and navigate to **System > Platform > Logging**, which opens a built-in Kibana instance.



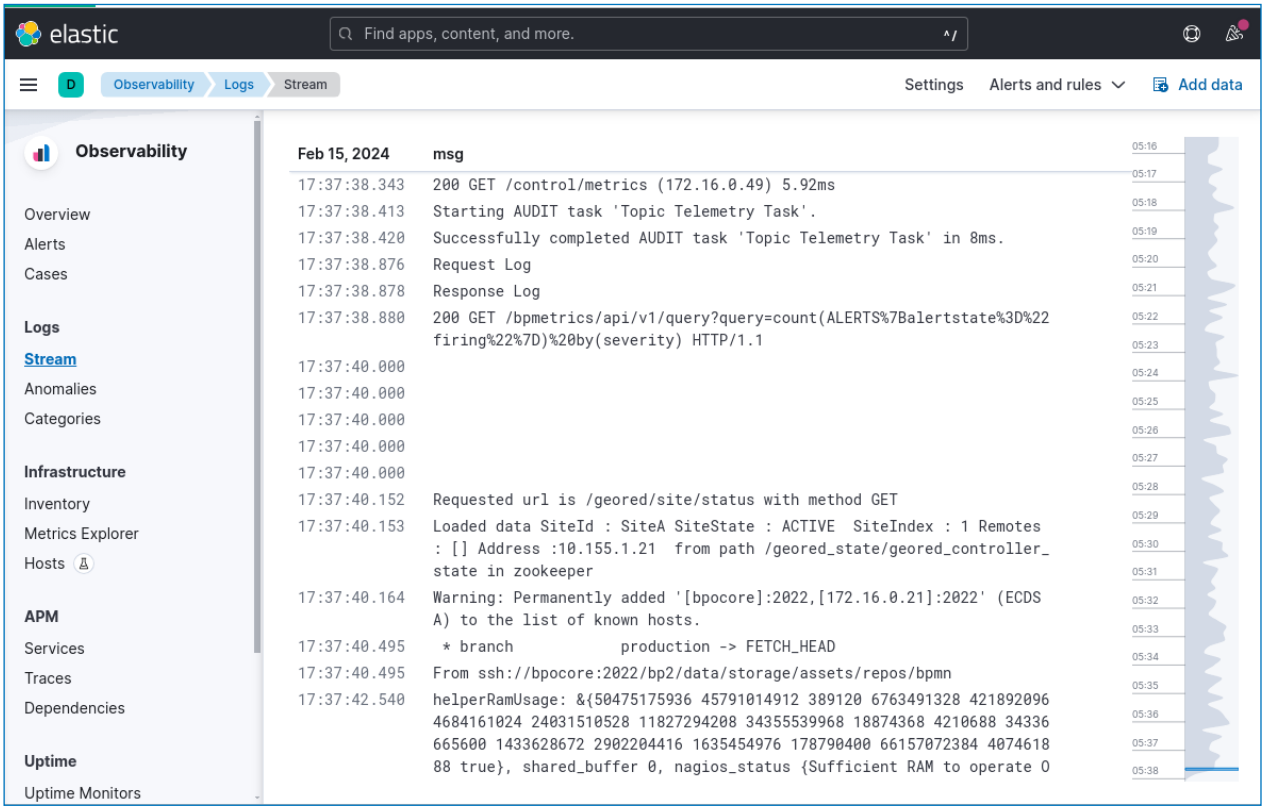
26. Within the Kibana platform, expand the menu and navigate to **Observability > Logs**.



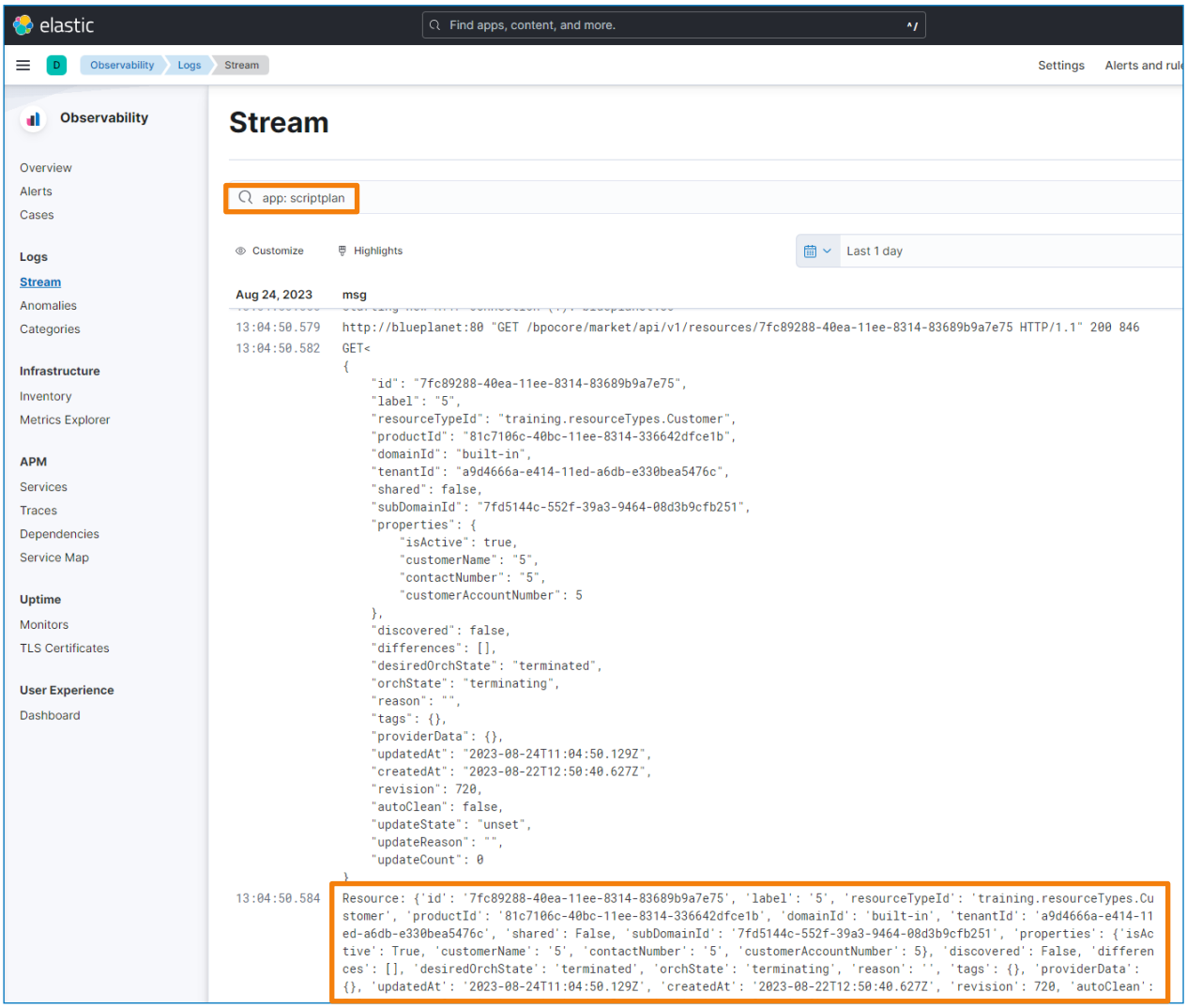
27. To improve the output of the log stream, click the **Settings** button in the upper right corner of Kibana. Under **Log Columns**, delete the **event.dataset** and **Message** columns, add the **msg** column, and click **Apply**.



28. Return to the log streaming page by navigating to **Menu (left side) > Logs > Stream**.



29. In the search bar, input **app: scriptsplan**. This will filter the log messages and only display the ones coming from that application. Once applied, you should see the same logs that you found in the scriptplan container.



30. Your next step is to create a similar remote script for the Bandwidth Profile. Start by creating the **bandwidthprofile.py** file in the **training/model-definitions/training** folder. Continue by importing the **Plan** class from the **plansdk.apis.plan** module, then define **Activate** and **Terminate** classes that contain the **run()** methods.

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Creates BandwidthProfile resource into market
    """
    def run(self):

class Terminate(Plan):
    """
    Remove BandwidthProfile from market
```

```
    """
    def run(self):
```

31. Add the Python code to the **run()** methods. Similarly, as before, log the input parameters, resourceId, and resource data. In the Terminate plan, make sure that no dependencies exist for the BandwidthProfile resource before deleting it.

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Creates BandwidthProfile resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Activate: DONE")
        return {}

class Terminate(Plan):
    """
    Remove BandwidthProfile from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"BandwidthProfile has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

32. Create the **port.py** file in the **training/model-definitions/training** folder next. Add similar classes and methods as you did for the previous two Python files.

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Create Port resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
```

```
        log.info(f"Resource: {resource}")

        log.info("Activate: DONE")
        return {}

class Terminate(Plan):
    """
    Delete Port resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Port has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

33. Create a remote script for the Site resource as well by creating the **site.py** file in the **training/model-definitions/training** folder. Add similar classes and methods as you did for the previous three Python files.

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Creates Site resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Activate: DONE")
        return {}

class Terminate(Plan):
    """
    Remove Site resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")
```

```
        log.info("Terminate: DONE")
        return {}
```

34. In addition to creating the **Site** resource, this remote script should also create the **Port** resource. Create an additional method called **create_port()** in the *Activate* class that takes care of Port creation during Site creation and returns the port object when finished.
    a. Inspect the **bpo.resources.create()** method in the PlanSDK documentation.

```
 create(parent_resource_id, data, wait_active=True, wait_time=300.0, interval=5.0, obfuscate=None,
log_bodies=True, validate=None)
```

    b. You can see that you need to add 2 mandatory parameters – **parent_resource_id**, which represents the ID of the Site resource, and **data**, which is a dictionary of properties that are used to create the resource. In the case of Port resource, you need to compose the **data** parameter out of:
        - **productId:** ID of the Port product.
        - **label:** The label for this Port resource.
        - **properties:** The properties of the port resource that you defined in the Port resource type – *deviceName*, *portName*, and *portSpeed*.
    c. Inspect the **bpo.market.get_produts_by_resource_type()** method in the PlanSDK documentation.

```
get_products_by_resource_type(resource_type_id)
```

    d. The only parameter needed here is the resource_type_id, which in the case of Port, is training.resourceTypes.Port.
    e. You can now start creating the **create_port()** method under the Activate class in the *site.py* file. First, read the Port product by its ID. The **get_products_by_resource_type** method returns a list since there can be multiple products with the same resource type. Make sure you only choose the first list element.

```
...
class Activate(Plan):
    """
    Creates Site resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Activate: DONE")
        return {}

    def create_port(self):
        port_product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Port')[0]
...
```

NOTE: You can also try to add validation for port products – make sure there is only one returned by this method.

    f. Add the code to create a new resource. To figure out where exactly to find the needed parameters, try looking at the logs from a few steps back when you created a Customer resource and inspected the logs both in the container and in Kibana. You will also need to add some additional input parameters to the method.

NOTE: To execute and test out the changes to your code, simply onboard the changes to BPO and create a Site resource either by hand or over the API.

g. The label of the Port resource should follow the same naming convention as the resources created over the declarative plan, meaning the format should look like this - **<parentResource>.Port**

h. This is how the finished **create_port()** method should look like.

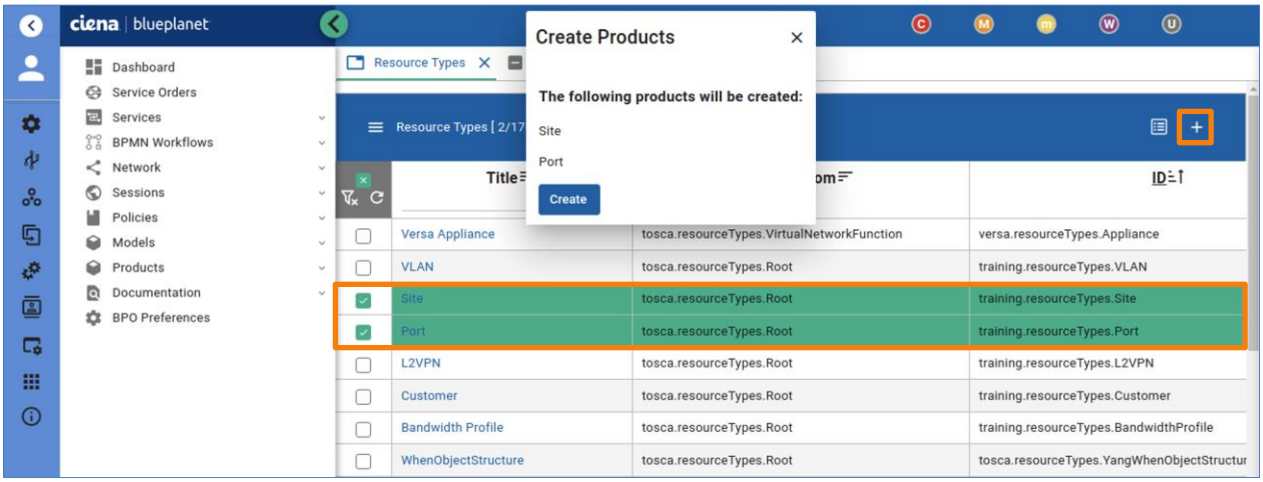
```
...
    def create_port(self, resource):
        properties = resource['properties']

        port_product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Port')[0]
        port = self.bpo.resources.create(self.params['resourceId'], {
            'productId': port_product['id'],
            'label': f"{resource['label']}.Port",
            'properties': {'deviceName': properties["deviceName"],
                           'portName': properties["portName"],
                           'portSpeed': properties["portSpeed"]}
        })
        return port
...
```

35. You can now include the **create_port()** method to the main **run()** method for the Site activation.

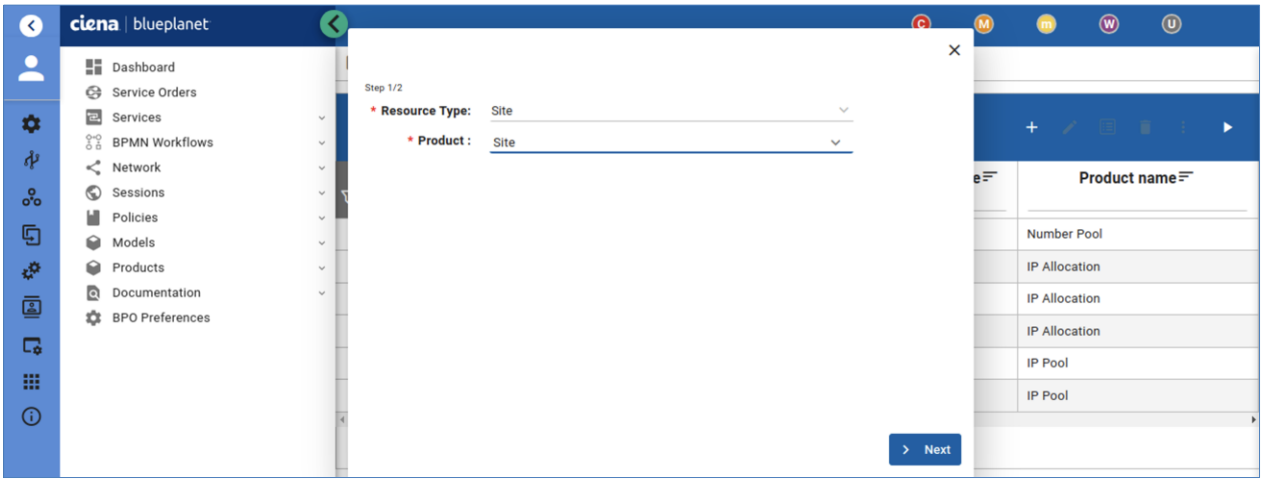
```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Creates Site resource into market
    Creates Port resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        port = self.create_port(resource)
        log.info(f"Create Port resource: {port}")

        log.info("Activate: DONE")
        return {}

    def create_port(self, resource):
        properties = resource['properties']

        port_product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Port')[0]
        port = self.bpo.resources.create(self.params['resourceId'], {
            'productId': port_product['id'],
            'label': f"{resource['label']}.Port",
            'properties': {'deviceName': properties["deviceName"],
                           'portName': properties["portName"],
                           'portSpeed': properties["portSpeed"]}
        })
        return port

class Terminate(Plan):
    """
    Remove Site resource from market
```

```
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

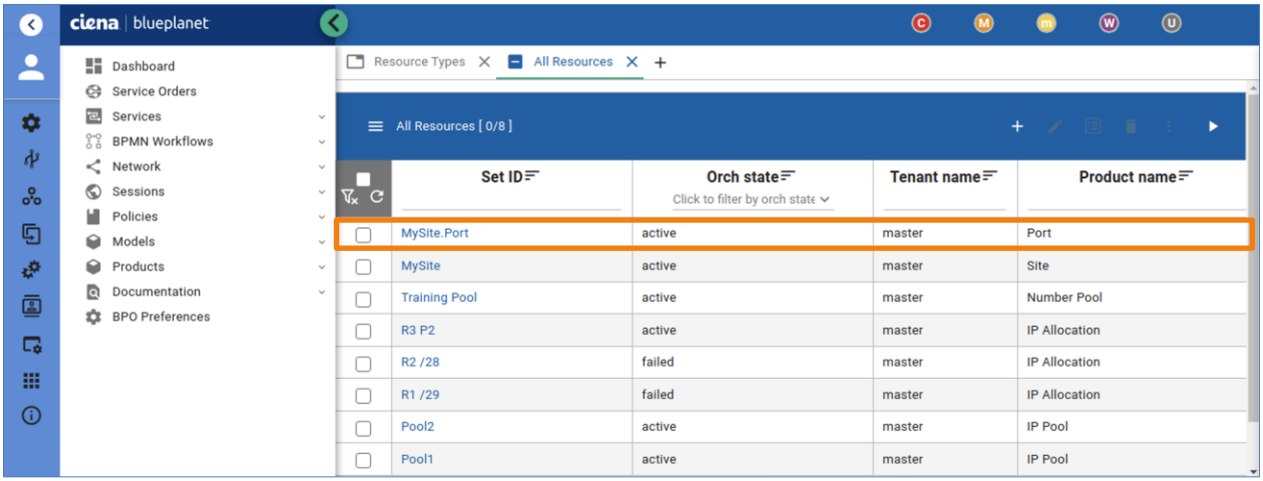
36. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
37. Create a Port and Site Products based on the updated resource types by navigating to **Models > Resource Types**, choosing the **Port** and **Site** resource type, clicking the **+** symbol in the upper-right corner, and clicking **Create.**



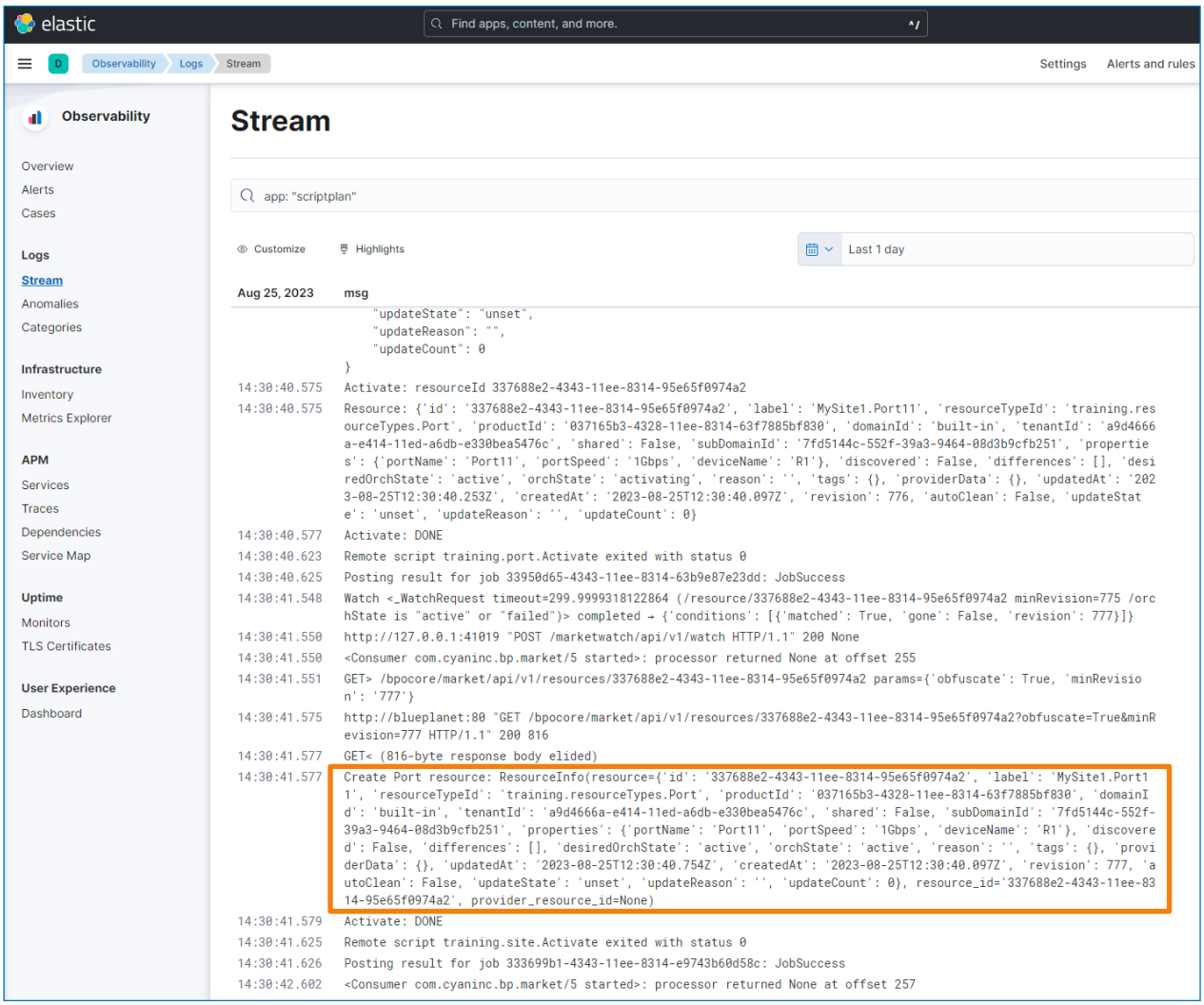
38. Create a new **Site** resource with label **MySite**.



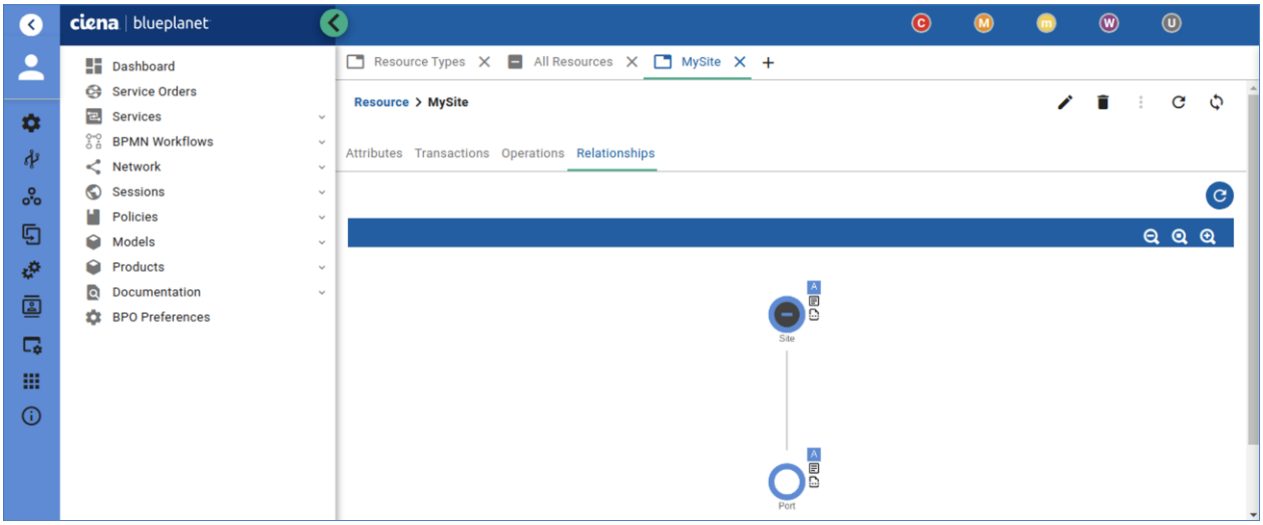
39. The Port resource should also be automatically created.



40. Inspect the **scriptplan** logs in Kibana and find the newly added log statements.



41. Back in BPO-UI, click on the created **Site** resource and open the **Relationships** tab. You will find that the relationship between the Site and Port resources has been created automatically since the Port was created during the Activate phase of the Site resource.



42. This takes care of the Port creation. However, you also need to take care of the Port resource when you delete the Site – the Port resource must be deleted during the Terminate phase as well.
    a. When working with PlanSDK, you usually have multiple ways to achieve your goals. In this case, one option would be to find the Port resource and delete it directly through **bpo.resources.delete()**. Another option would be to list the Site dependencies and delete all the Port resources that are found that way. In this example, the latter option will be shown. Inspect the **PlanSDK** documentation for the **delete_dependencies()** method and the mandatory parameters it requires.

```
delete_dependencies(parent_id, resource_type, dependencies, force_delete_relationship=False,
await_termination=True, timeout=300.0, validate=None)
```

    b. You can get the parent_id by passing the Site resource ID to this method, the resource_type for Port is a known string, and the dependencies list can be obtained with the **get_dependencies()** method you are already using in the *Terminate* class.
    c. Create the **delete_port()** method within the Terminate class. The final method should look something like this:

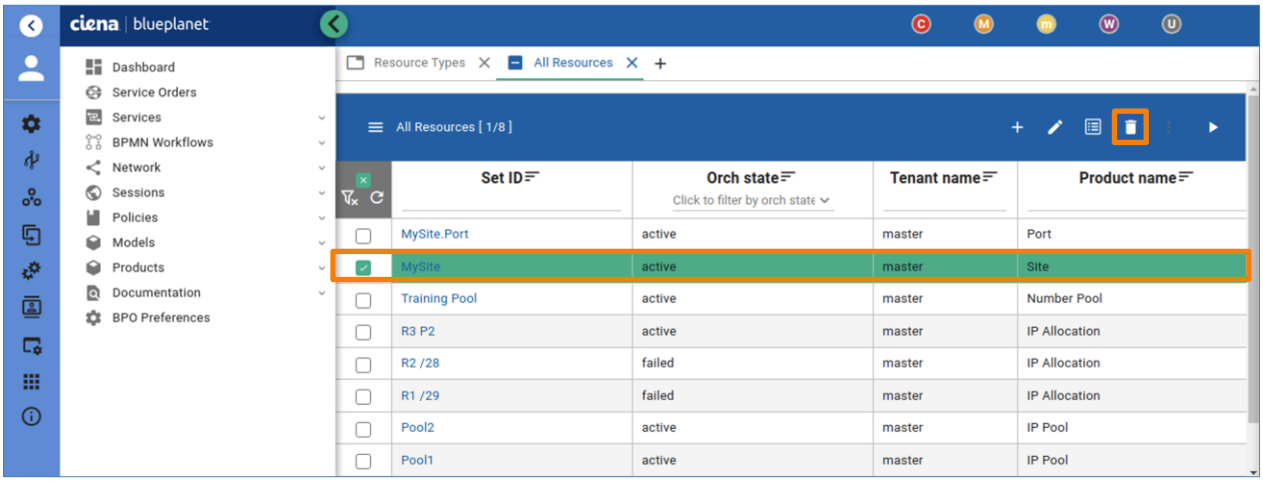
```
    def delete_port(self, resource_id):
        dependencies = self.bpo.resources.get_dependencies(resource_id)
        self.bpo.resources.delete_dependencies(
                            self.params['resourceId'],
                            'training.resourceTypes.Port',
                            dependencies)
```

    d. Call this method from the **run()** method in the Terminate class in the site.py file.

```
class Terminate(Plan):
    """
    Remove Site resource from market
    Remove dependent Port resources from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")
```

```
        self.delete_port(resource_id)
        log.info(f"Deleting Port dependencies for Site {resource['label']}")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

43. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
44. Navigate to the BPO UI, and delete *only* the previously created Site resource under **Network > All Resources**. If your newly added method works as intended, the Port resource will get deleted as well. If there is a failure, click on the failed resource, inspect the error message, and troubleshoot the issue.



NOTE: If you see additional resources with Resource Type ID starting with "ipam.resourceTypes", do not delete them, you will use them later in the labs.

45. As the final Python file, create the **l2vpn.py** file located in the **training/model-definitions/training** folder. This file will contain classes and methods that implement L2VPN lifecycle operations.
46. As with other Python remote script files, add the **Activate** and **Terminate** classes with the **run()** methods, and take care of imports and logging level.

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Create L2VPN Resource to market
    """
    def run(self):

class Terminate(Plan):
    """
    Remove L2VPN resource from market
    """
    def run(self):
```

47. Add the code to log the input parameters, resourceId, and resource data, and the return statements to both **run()** methods.

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)

class Activate(Plan):
    """
    Create L2VPN resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Activate: DONE")
        return {}

class Terminate(Plan):
    """
    Delete L2VPN resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Terminate: DONE")
        return {}
```

48. The L2VPN Activate operation should create the following resources:
    - Customer
    - Bandwidth Profile
    - Site

    Create methods in the *Activate* class for all of them.

```
class Activate(Plan):
    """
    Create L2VPN resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Activate: DONE")
        return {}

    def create_customer():
    def create_site():
    def create_bandwidthprofile():
```

49. Implement the resource creation for Customer, Bandwidth Profile, and Site resources in the remote script.
    a. Each resource is created with the **bpo.resources.create()** method, similarly, as you created the Port resource from the Site remote script – you need to get the resource type's **productId**, you need to create a **label** for the resource, and you need to create a **properties** dictionary that contains the resource properties.
    b. Implement the **create_customer()** method.

```
    def create_customer(self, resource):
        properties = resource['properties']

        product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Customer')[0]
        customer = self.bpo.resources.create(self.params['resourceId'], {
            'productId': product['id'],
            'label': f"{resource['label']}.Customer",
            'properties': {'customerName': properties["customerName"],
                           'customerAccountNumber': properties["customerAccountNumber"],
                           'contactNumber': properties["contactNumber"],
                           'isActive': properties["isActive"]}
        })
        return customer
```

    c. Implement the **create_site()** method.

```
    def create_site(self, resource):
        properties = resource['properties']

        product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Site')[0]
        site = self.bpo.resources.create(self.params['resourceId'], {
            'productId': product['id'],
            'label': f"{resource['label']}.Site",
            'properties': {'siteId': properties["siteId"],
                           'portName': properties["portName"],
                           'deviceName': properties["deviceName"],
                           'portSpeed': properties["portSpeed"]}
        })
        return site
```

    d. Implement the **create_bandwidthprofile()** method.

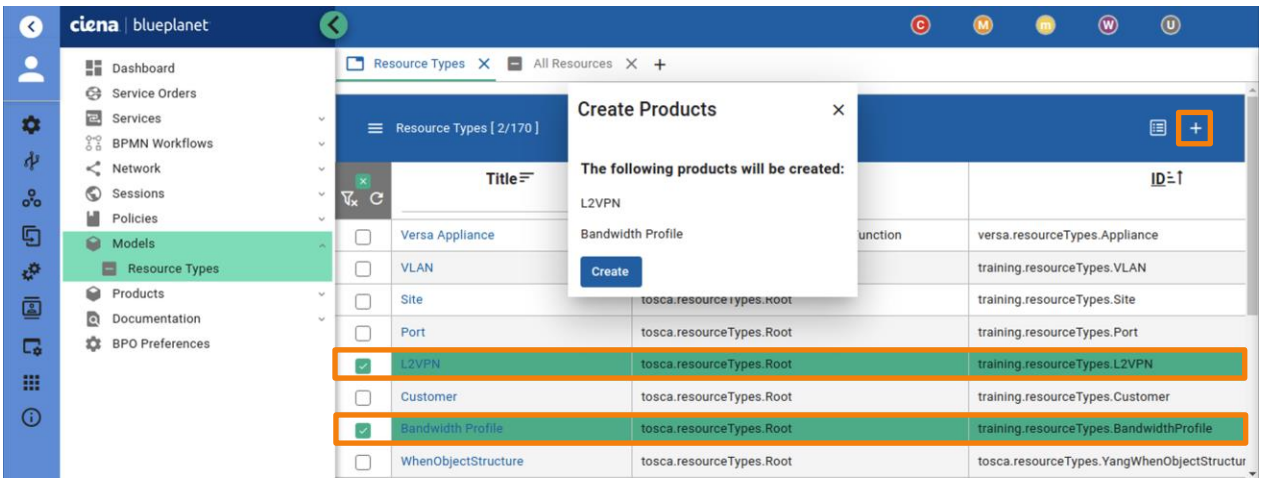
```
    def create_bandwidthprofile(self, resource):
        properties = resource['properties']

        product =
self.bpo.market.get_products_by_resource_type('training.resourceTypes.BandwidthProfile')[0]
        bandwidth_profile = self.bpo.resources.create(self.params['resourceId'], {
            'productId': product['id'],
            'label': f"{resource['label']}.BandwidthProfile",
            'properties': {'cir': properties["cir"],
                           'eir': properties["eir"],
                           'cbs': properties["cbs"],
                           'ebs': properties["ebs"]}
        })
        return bandwidth_profile
```

    e. Call all three methods from the **run()** method in the **L2VPN Activate** class.

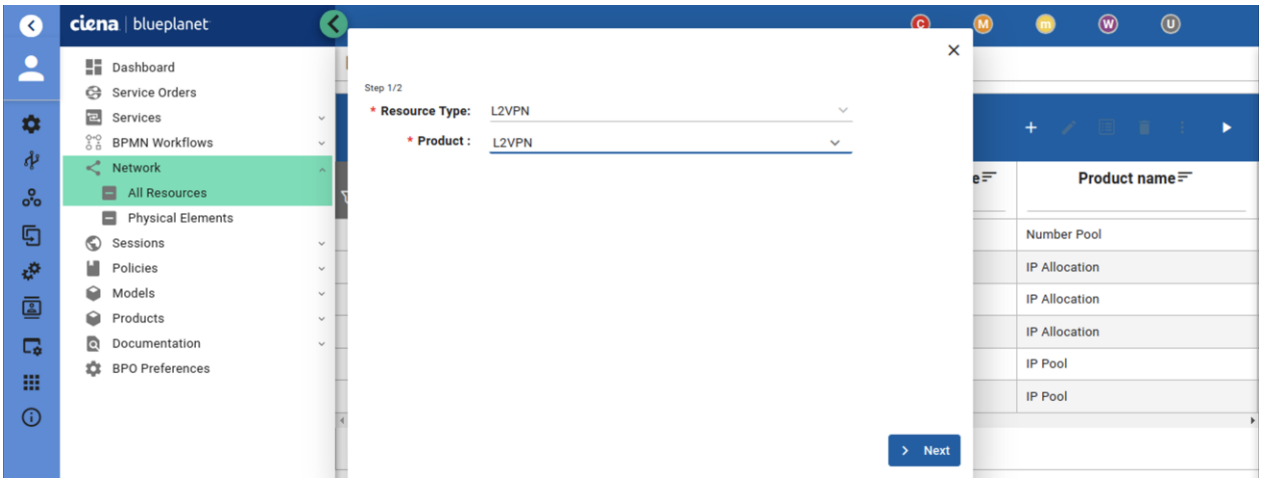
```
class Activate(Plan):
    """
    Create L2VPN resource into market
    Create Customer resource into market
    Create Site resource into market
    Create Bandwidth Profile resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")
```

```
        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        customer = self.create_customer(resource)
        log.info(f"Create Customer resource: {customer}")

        site = self.create_site(resource)
        log.info(f"Create Site resource: {site}")

        bw_profile = self.create_bandwidthprofile(resource)
        log.info(f"Create Bandwidth Profile resource: {bw_profile}")

        log.info("Activate: DONE")
        return {}
```

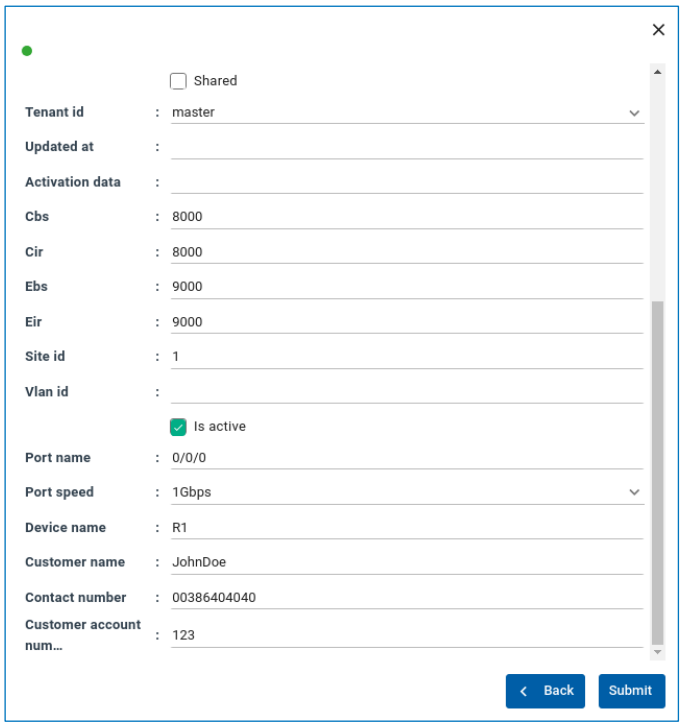
50. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
51. Create a **Bandwidth Profile** and **L2VPN** Products based on the updated resource types by going to **Models > Resource Types**, choosing the Bandwidth Profile and L2VPN resource types, clicking the **+** symbol in the upper-right corner, and clicking **Create**.



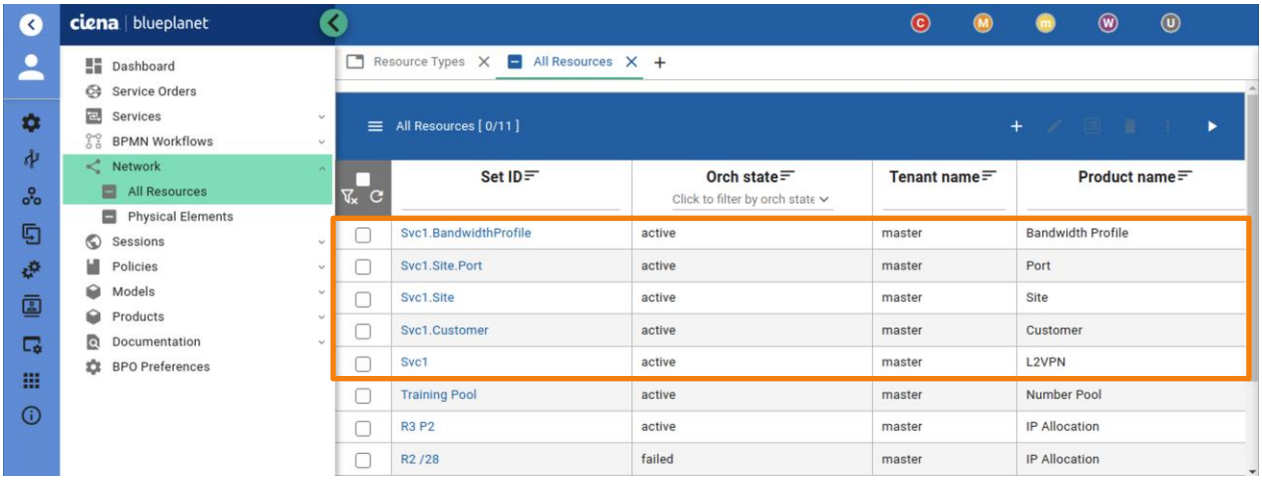
52. Try out the new Activate operation for the L2VPN resource. Make sure you onboard all the changes to the BPO server and create a new L2VPN resource through the BPO UI. You need to input the following parameters:
    - **Label** (for example - **Svc1**)



- You can skip the **Vlan id** for now, since you have not implemented the VLAN assignment yet in Python.



53. A new L2VPN resource will be created, along with the other four types of sub-resources – Port, Site, Customer, and BandwidthProfile. Make sure all the resources are in an **active** state. If that is not the case, inspect the resource error messages by clicking on them, and troubleshooting the issues.



NOTE: If you see additional resources with Resource Type ID starting with "ipam.resourceTypes", do not delete them, you will use them later in the labs.

54. You also need to implement the code to delete all the sub-resources when you delete the L2VPN resource. Create the following methods in the L2VPN Terminate class:

- **delete_customer()**
- **delete_site()**
- **delete_bandwidthprofile()**

```
class Terminate(Plan):
    """
    Delete L2VPN resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        log.info("Terminate: DONE")
        return {}

    def delete_customer():

    def delete_site():

    def delete_bandwidthprofile():
```

55. Implement the sub-resource deletion code. Similarly to when you have deleted the Port sub-resource from the Site Terminate remote script, you can use the **bpo.resources.delete_dependencies()** method. The finished code should look something like this:

```
    def delete_customer(self, resource_id):
        dependencies = self.bpo.resources.get_dependencies(resource_id)
        self.bpo.resources.delete_dependencies(
                            self.params['resourceId'],
                            'training.resourceTypes.Customer',
                            dependencies)

    def delete_site(self, resource_id):
        dependencies = self.bpo.resources.get_dependencies(resource_id)
        self.bpo.resources.delete_dependencies(
                            self.params['resourceId'],
                            'training.resourceTypes.Site',
                            dependencies)

    def delete_bandwidthprofile(self, resource_id):
        dependencies = self.bpo.resources.get_dependencies(resource_id)
        self.bpo.resources.delete_dependencies(
                            self.params['resourceId'],
                            'training.resourceTypes.BandwidthProfile',
                            dependencies)
```

56. Call these methods from the **L2VPN Terminate run()** method.

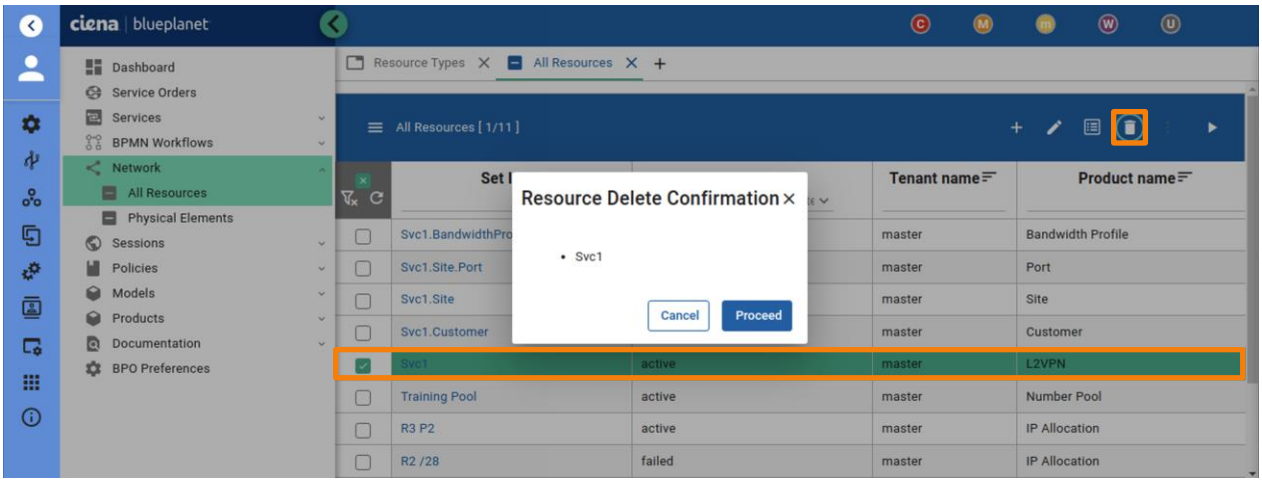
```
    class Terminate(Plan):
    """
    Delete L2VPN resource from market
    Delete Customer resource from market
    Delete Site resource from market
    Delete Bandwidthprofile resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        self.delete_customer(resource_id)
        log.info(f"Deleting Customer dependencies for L2VPN {resource['label']}")
```

```
        self.delete_site(resource_id)
        log.info(f"Deleting Site dependencies for L2VPN {resource['label']}")

        self.delete_bandwidthprofile(resource_id)
        log.info(f"Deleting BandwidthProfile dependencies for L2VPN {resource['label']}")

        log.info("Terminate: DONE")
        return {}
```

57. Add some code at the end of the **run()** method that makes sure the L2VPN resource you are deleting has no more dependencies.

```
class Terminate(Plan):
    """
    Delete L2VPN resource from market
    Delete Customer resource from market
    Delete Site resource from market
    Delete Bandwidthprofile resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        self.delete_customer(resource_id)
        log.info(f"Deleting Customer dependencies for L2VPN {resource['label']}")

        self.delete_site(resource_id)
        log.info(f"Deleting Site dependencies for L2VPN {resource['label']}")

        self.delete_bandwidthprofile(resource_id)
        log.info(f"Deleting BandwidthProfile dependencies for L2VPN {resource['label']}")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"L2VPN has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

58. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
59. To test out your changes, delete the L2VPN resource *only*. If there are no issues with your code, this will cause all the sub-resources to be deleted as well.



Task 4: Implement VLAN Allocation Using PlanSDK

In this task, you will learn how to utilize PlanSDK to create and manage built-in resources and to implement VLAN allocation for the L2VPN resource.

1. Your task is to implement VLAN allocation in the same way you did in the previous lab exercise – the user can either specify the Vlan ID as an input parameter for the L2VPN resource or leave the Vlan ID field empty and get the VLAN allocated dynamically using the NumberPool and PooledNumber resources. In VS Code, open the **resource_type_l2vpn.tosca** file and make sure that the vlanId property in L2VPN is correctly defined there and delete definition for VLAN it should looks like this:

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)"  ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {
      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
        updatable = true
      }

      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }
...
```

2. For VLAN allocation, you need to implement two methods in the L2VPN Activate class:
   - **create_vlan_pool()** method, which will create a NumberPool resource for VLAN allocation, but only if it does not exist yet – you do not want to have a unique NumberPool for each resource, but a global one.
   - **allocate_vlan()** method, which will allocate VLAN for the L2VPN service – if a vlanId parameter is provided, the method should allocate this static value, or allocate the next available VLAN from the VLAN pool.
3. First, assign some global variables at the top of the **l2vpn.py** file, which you need throughout this task:
   - POOL_NAME = **"Training Pool"**
   - POOL_RESOURCE_TYPE = **"tosca.resourceTypes.NumberPool"**
   - POOL_NUMBER_RESOURCE_TYPE = **"tosca.resourceTypes.PooledNumber"**

```
from plansdk.apis.plan import Plan
import logging

log = logging.getLogger(__name__)
```

```
POOL_NAME = "Training Pool"
POOL_RESOURCE_TYPE = "tosca.resourceTypes.NumberPool"
POOL_NUMBER_RESOURCE_TYPE = "tosca.resourceTypes.PooledNumber"

class Activate(Plan):
...
```

NOTE: This is done to reduce code duplication and to have these static strings defined in a single place.

4. Next, implement the **create_vlan_pool()** method.
   a. The method should first try to find an existing pool. You can use the **bpo.resources.get_one_by_filters()** method, which tries to find a resource with specified filters, or returns an error.

```
    def create_vlan_pool(self):
        # Try to find the "tosca.resourceTypes.NumberPool" resource
        pool = self.bpo.resources.get_one_by_filters(
            resource_type = POOL_RESOURCE_TYPE,
            q_params = { "label": POOL_NAME }
        )
```

   b. Since the method returns an error if the pool does not exist, you need to wrap it inside a try-except block.

```
    def create_vlan_pool(self):
        # Try to find the "tosca.resourceTypes.NumberPool" resource
        # If none are found, create it
        try:
            pool = self.bpo.resources.get_one_by_filters(
                resource_type = POOL_RESOURCE_TYPE,
                q_params = { "label": POOL_NAME }
            )
            log.info(f"Found VLAN pool {POOL_NAME}")
        except:
            log.info(f"VLAN pool {POOL_NAME} doesn't exist. Creating...")
```

   c. Next, create a **Number Pool** resource. For parameters, you need the following data:
      - The parent_resource_id should be set to **None** since the pool will be a global resource
      - The data object should be composed of the following three parameters:
        - NumberPool productId, which you can read through the **bpo.market.get_products_by_resource_type** method
        - label, which should be **Training Pool**.
        - properties, which should be lowest (**1**), and highest (**4095**) and where both values need to be passed as integers (and not strings).
   d. The finalized **create_vlan_pool()** method should look like this:

```
    def create_vlan_pool(self):
        # Try to find the "tosca.resourceTypes.NumberPool" resource
        # If none are found, create it
        try:
            pool = self.bpo.resources.get_one_by_filters(
                resource_type = POOL_RESOURCE_TYPE,
                q_params = { "label": POOL_NAME }
            )
            log.info(f"Found VLAN pool {POOL_NAME}")
        except:
            log.info(f"VLAN pool {POOL_NAME} doesn't exist. Creating...")
            vlan_pool_id = self.bpo.market.get_products_by_resource_type(POOL_RESOURCE_TYPE)[0]['id']

            # Training VLAN pool created with no parent - it is global
            pool = self.bpo.resources.create(None, {
                'productId': vlan_pool_id,
```

```
                'label': POOL_NAME,
                'properties': { "lowest":  1,
                                "highest": 4095}
            })
        return pool
```

5. Now, implement the **allocate_vlan** method. This method should:
    - Verify that a **PooledNumber** product exists or raise an error otherwise.
    - Try to read the **vlanId** from input properties.
    - Pass this property to the **PooledNumber** resource as the **requestedValue** parameter …
    - … or allocate the **VLAN** dynamically if vlanId is not set, skipping the **requestedValue**.

    a. Create a try-catch block and try reading a product with the **PooledNumber** resource type. Otherwise, terminate allocation with a new **Exception** and a meaningful error message.

```
    def allocate_vlan(self):
        pool_number_product_id = None

        # First, read the ID from product based on "tosca.resourceTypes.PooledNumber"
        try:
            pool_number_product_id = self.bpo.products.get_by_domain_and_type("built-in",
POOL_NUMBER_RESOURCE_TYPE)[0]['id']
            log.info(f"Found VLAN pool number product with ID {pool_number_product_id}")
        except:
            raise Exception(f"VLAN pool number product with ID {pool_number_product_id} doesn't exist")
```

    b. Read the **vlanId** from input properties. Read the L2VPN **label** as well since you will need it when creating the allocated VLAN label. Don't forget to add the resource as a method argument.

```
    def allocate_vlan(self, resource):
        pool_number_product_id = None

        # First, read the ID from product based on "tosca.resourceTypes.PooledNumber"
        try:
            pool_number_product_id = self.bpo.products.get_by_domain_and_type("built-in",
POOL_NUMBER_RESOURCE_TYPE)[0]['id']
            log.info(f"Found VLAN pool number product with ID {pool_number_product_id}")
        except:
            raise Exception(f"VLAN pool number product with ID {pool_number_product_id} doesn't exist")

        vlan_id = resource['properties'].get('vlanId')
        label = resource['label']
```

    c. Create a **PooledNumber** resource, based on the *vlanId* parameter – if it exists, pass it as the **requestedValue** parameter when creating the resource, else skip it.

```
    def allocate_vlan(self, resource):
        pool_number_product_id = None

        # First, read the ID from product based on "tosca.resourceTypes.PooledNumber"
        try:
            pool_number_product_id = self.bpo.products.get_by_domain_and_type("built-in",
POOL_NUMBER_RESOURCE_TYPE)[0]['id']
            log.info(f"Found VLAN pool number product with ID {pool_number_product_id}")
        except:
            raise Exception(f"VLAN pool number product with ID {pool_number_product_id} doesn't exist")

        vlan_id = resource['properties'].get('vlanId')
        label = resource['label']

        # If vlanId has been entered use that when creating the VLAN resource,
        # else use the allocated number from PooledNumber
        pool_number_product_id = self.bpo.products.get_by_domain_and_type("built-in",
POOL_NUMBER_RESOURCE_TYPE)[0]['id']
        if vlan_id:
            allocated_number = self.bpo.resources.create(resource['id'], {
```

```
                'productId': pool_number_product_id,
                'label': f"{label}.VLAN",
                'properties': { "requestedValue": vlan_id }
            })
        else:
            allocated_number = self.bpo.resources.create(resource['id'], {
                'productId': pool_number_product_id,
                'label': f"{label}.VLAN"
            })
        log.info(f"Successfully allocated VLAN pool number - {allocated_number}")
```

d. Call the two new methods from the **run()** method in the Activate class.

```
...
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        customer = self.create_customer(resource)
        log.info(f"Create Customer resource: {customer}")

        site = self.create_site(resource)
        log.info(f"Create Site resource: {site}")

        bw_profile = self.create_bandwidthprofile(resource)
        log.info(f"Create Bandwidth Profile resource: {bw_profile}")

        self.create_vlan_pool()
        self.allocate_vlan(resource)

        log.info("Activate: DONE")
        return {}
...
```

6. Next, take care of VLAN deallocation when you delete your resource. Delete the **PooledNumber** resource with the **bpo.resources.delete()** method within the **run()** method of the *Terminate* class.

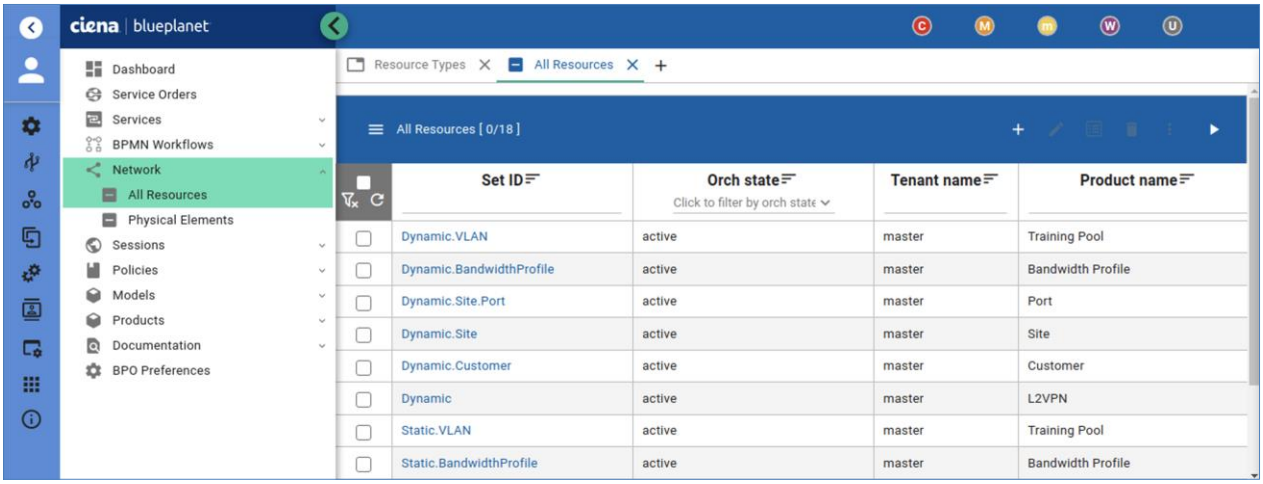
```
class Terminate(Plan):
    """
    Delete L2VPN resource from market
    Delete Customer resource from market
    Delete Site resource from market
    Delete Bandwidthprofile resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        self.delete_customer(resource_id)
        log.info(f"Deleting Customer dependencies for L2VPN {resource['label']}")

        self.delete_site(resource_id)
        log.info(f"Deleting Site dependencies for L2VPN {resource['label']}")

        self.delete_bandwidthprofile(resource_id)
        log.info(f"Deleting BandwidthProfile dependencies for L2VPN {resource['label']}")

        label = resource['label']
```

```
        try:
            vlan_num_res = self.bpo.resources.get_one_by_filters(
                    resource_type=POOL_NUMBER_RESOURCE_TYPE,
                    q_params={"label": f"{label}.VLAN"}
            )
            self.bpo.resources.delete(vlan_num_res['id'])
            log.info(f"VLAN for {label} released")
        except:
            log.info(f"VLAN for {label} not found")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

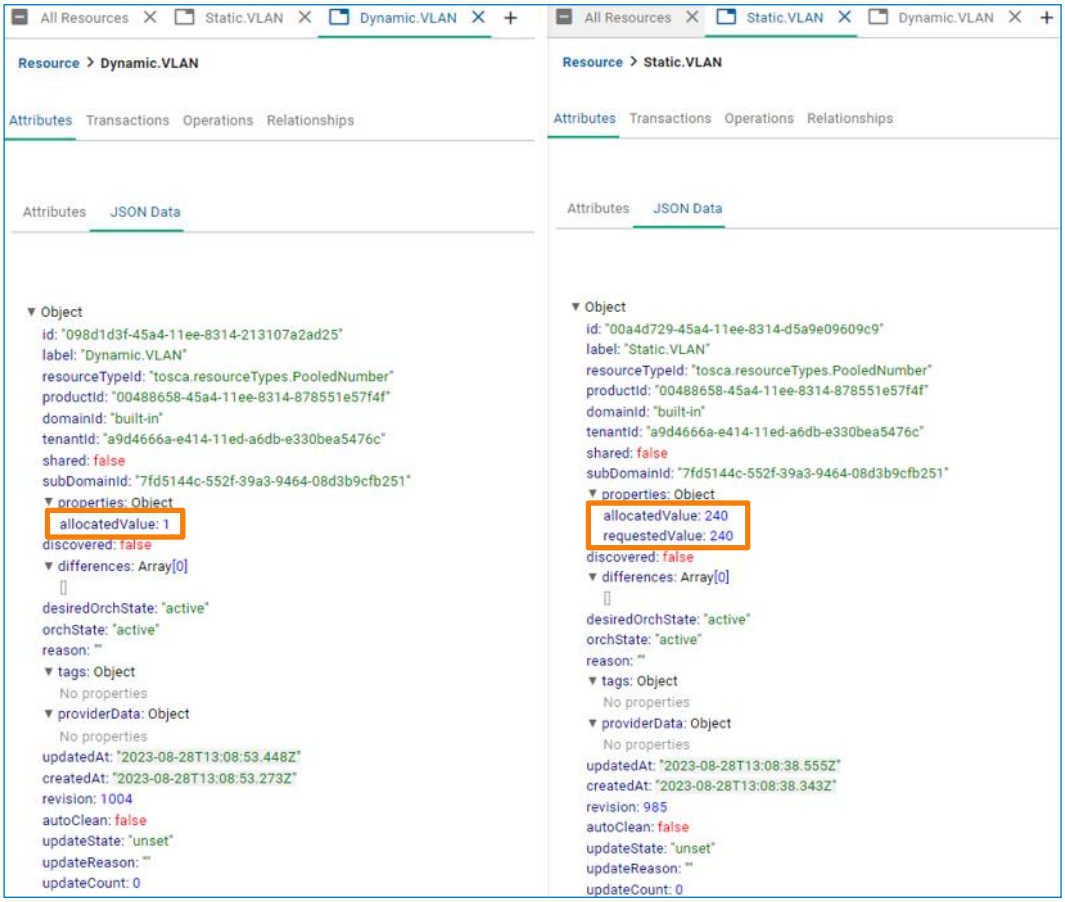
7. Resources sometimes take a few seconds to get deleted. This can cause unintended consequences when deleting their parent resources, where dependencies still exist. It is always a good idea to include some code that checks and waits for a resource to be deleted. Open the **PlanSDK** documentation and study the **bpo.resources.await_termination()** method. Use this method to make sure that the VLAN resource is deleted before proceeding.

```
class Terminate(Plan):
    """
    Delete L2VPN resource from market
    Delete Customer resource from market
    Delete Site resource from market
    Delete Bandwidthprofile resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        self.delete_customer(resource_id)
        log.info(f"Deleting Customer dependencies for L2VPN {resource['label']}")

        self.delete_site(resource_id)
        log.info(f"Deleting Site dependencies for L2VPN {resource['label']}")

        self.delete_bandwidthprofile(resource_id)
        log.info(f"Deleting BandwidthProfile dependencies for L2VPN {resource['label']}")

        label = resource['label']
        try:
            vlan_num_res = self.bpo.resources.get_one_by_filters(
                    resource_type=POOL_NUMBER_RESOURCE_TYPE,
                    q_params={"label": f"{label}.VLAN"}
            )
            self.bpo.resources.delete(vlan_num_res['id'])
            self.bpo.resources.await_termination(vlan_num_res['id'], f"{label}.VLAN", False)
            log.info(f"VLAN for {label} released")
        except:
            log.info(f"VLAN for {label} not found")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

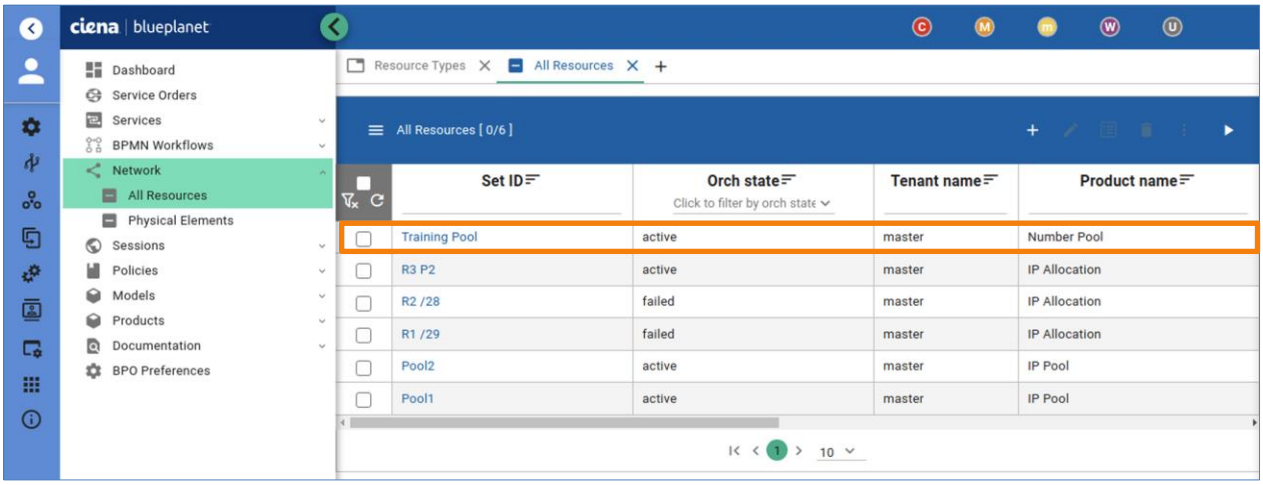
8. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
9. Try out the new changes. Create two **L2VPN** resources – one where you assign the **Vlan ID** parameter (label – **Static**), and the other when you leave the input field empty (label – **Dynamic**). You can also observe the **Training Pool Number Pool** get created.



10. Inspect and compare the **Static.VLAN** and **Dynamic.VLAN** resources and their properties. One should feature the same allocatedValue as you passed over the requestedValue property, and the other one should have a dynamically allocated allocatedValue.



11. Delete both **L2VPN** resources. All sub-resources should get deleted along them, as well as the allocated VLANs. Keep the Training Pool created.



NOTE: If you see additional resources with Resource Type ID starting with “ipam.resourceTypes”, do not delete them, you will use them later in the labs.

Task 5: Implement an Update Operation

In this final task for this lab, you implement an Update operation for your L2VPN resource using an imperative plan. This operation updates the vlanId parameter.

1. Open the **service_template_l2vpn.tosca** file. Add an **update** object to the service template plans.

```
...
    plans {

      activate {
        type = remote
        language = python
        path = training.l2vpn.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.l2vpn.Terminate
      }

      update {
        type = remote
        language = python
        path = training.l2vpn.Update
      }
    }
...
```

2. Open the **l2vpn.py** file. Add an **Update** class that contains the **run()** method, some basic logging like you did in the other callback methods, and return an empty dictionary.

```
...
class Update(Plan):
    """
    Updates the VLAN parameter
    """
    def run(self):
        log.info(f"Update: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Update: resourceId {resource_id}")
        log.info(f"Update: {resource}")

        log.info("Update: DONE")
        return {}
```

3. Think about what needs to happen for the VLAN resource to be updated:
    - First, you need to identify the current VLAN resource.
    - The relationship between the VLAN and the L2VPN resource needs to be deleted.
    - The VLAN resource then needs to be deleted as well.
    - A new VLAN resource must be created, either by requesting a specific value or assigning the next available value from the VLAN pool.

NOTE: In some cases, an update of a resource can simply be performed with the **bpo.resources.put()** or **bpo.resources.patch()** methods. But, since you are working with the PooledNumber built-in resource, which does not support patching the **requestedValue** parameter, you must manually take care of deletion and creation.

4. Add the code that identifies the current VLAN resource.

```
...
class Update(Plan):
    """
    Updates the VLAN parameter
    """
    def run(self):
        log.info(f"Update: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Update: resourceId {resource_id}")
        log.info(f"Update: {resource}")

        label = resource['label']
        vlan_num_res = self.bpo.resources.get_one_by_filters(
                resource_type=POOL_NUMBER_RESOURCE_TYPE,
                q_params={"label": f"{label}.VLAN"}
        )

        log.info("Update: DONE")
        return {}
```

5. Study the **bpo.relationships.delete_relationships()** method from the PlanSDK library. This method deletes all the relationships that are attached to a given target. In your case, you delete all the relationships towards the VLAN resource, since you cannot delete resources with active relationships.

```
delete_relationships(target_id)
```

6. Use the **bpo.relationships.delete_relationship()** method to delete the relationship between the VLAN and the L2VPN resource.

```
...
class Update(Plan):
    """
    Updates the VLAN parameter
    """
    def run(self):
        log.info(f"Update: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Update: resourceId {resource_id}")
        log.info(f"Update: {resource}")

        label = resource['label']
        vlan_num_res = self.bpo.resources.get_one_by_filters(
                resource_type=POOL_NUMBER_RESOURCE_TYPE,
                q_params={"label": f"{label}.VLAN"}
        )

        self.bpo.relationships.delete_relationships(vlan_num_res['id'])

        log.info("Update: DONE")
        return {}
```

7. Now you can add the code that deletes the VLAN resource.

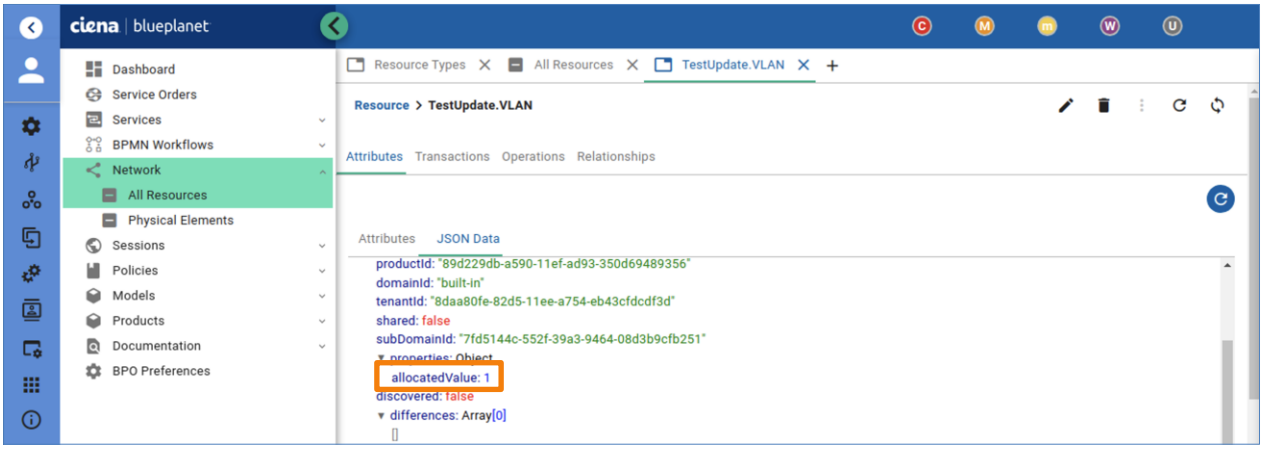
```
...
class Update(Plan):
    """
    Updates the VLAN parameter
    """
    def run(self):
        log.info(f"Update: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Update: resourceId {resource_id}")
        log.info(f"Update: {resource}")

        label = resource['label']
        vlan_num_res = self.bpo.resources.get_one_by_filters(
                resource_type=POOL_NUMBER_RESOURCE_TYPE,
                q_params={"label": f"{label}.VLAN"}
        )

        self.bpo.relationships.delete_relationships(vlan_num_res['id'])
        self.bpo.resources.delete(vlan_num_res['id'])
        log.info(f"VLAN for {label} released")

        log.info("Update: DONE")
        return {}
```

8. Implement the code that allocates a new VLAN. Since you have already implemented such functionality in the *Activate* class, you can simply re-use the **allocate_vlan()** method in the Update class as well.

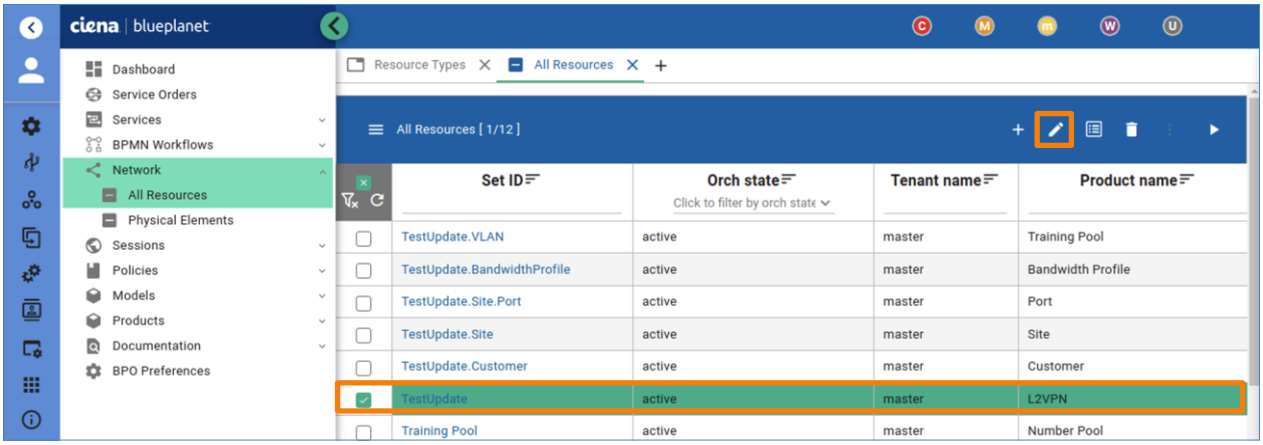
```
...
class Update(Plan):
    """
    Updates the VLAN parameter
    """
    def run(self):
        log.info(f"Update: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Update: resourceId {resource_id}")
        log.info(f"Update: {resource}")

        label = resource['label']
        vlan_num_res = self.bpo.resources.get_one_by_filters(
                resource_type=POOL_NUMBER_RESOURCE_TYPE,
                q_params={"label": f"{label}.VLAN"}
        )

        self.bpo.relationships.delete_relationships(vlan_num_res['id'])
        self.bpo.resources.delete(vlan_num_res['id'])
        log.info(f"VLAN for {label} released")

        Activate.allocate_vlan(self, resource)

        log.info("Update: DONE")
        return {}
```

9. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

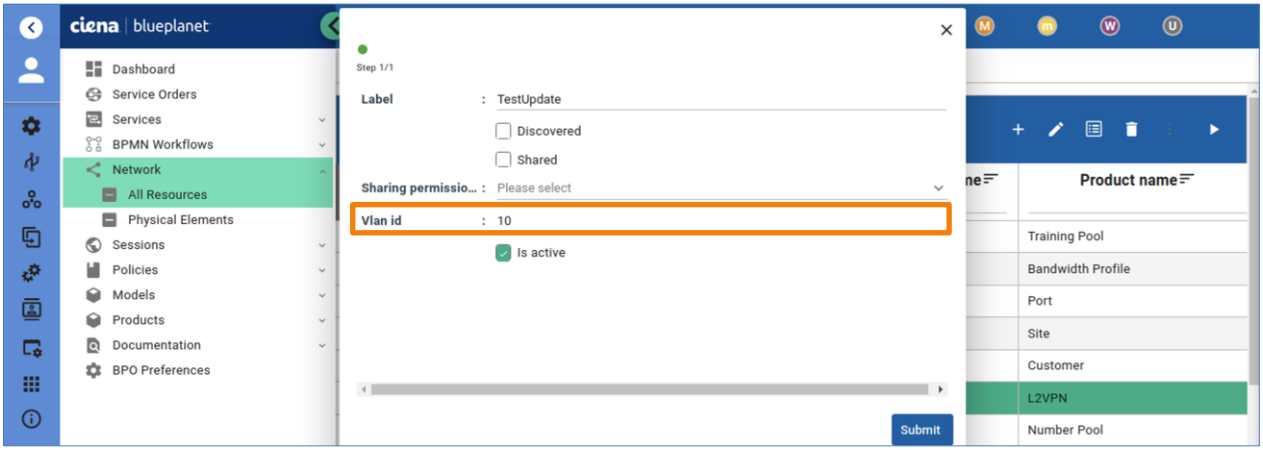
10. Test the new Update operation. Open the BPO UI and navigate to the **Network > All Resources** view. Create a new L2VPN resource called **TestUpdate** with an empty **Vlan ID** field.
11. Inspect the **TestUpdate.VLAN** resource. The allocatedValue parameter should be *1* since no other VLANs are currently assigned from that pool.



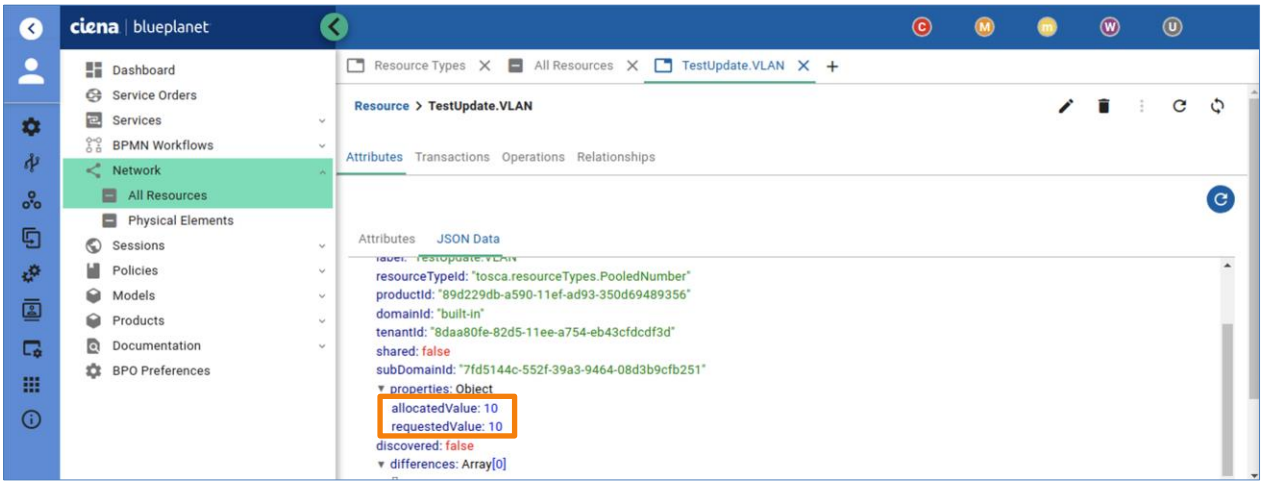
12. Edit the **TestUpdate** service by clicking on the **Pen** icon in the upper-right corner of the screen.



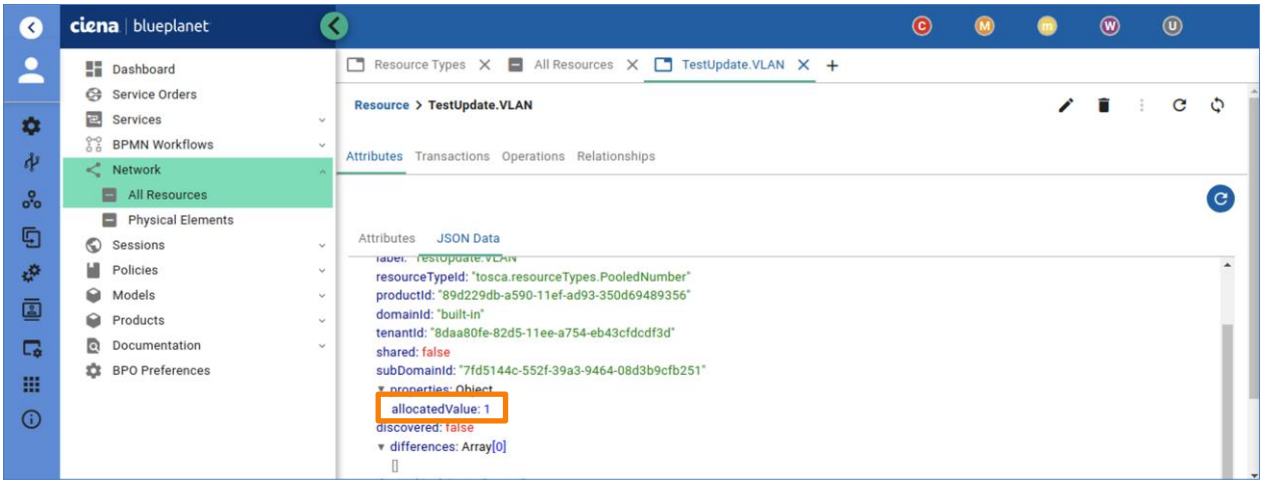
13. A new window pops up, allowing you to change some resource parameters. Notice that you cannot change all the parameters, only some generic ones and the ones that have an **"updatable = true"** parameter set in the resource type definition. Change the VLAN ID to an arbitrary number and click **Submit**.



14. Once the Update operation finishes, inspect the **TestUpdate.VLAN** resource again. The allocatedValue parameter should now equal *10*.



15. Update the **TestUpdate** resource again, this time leaving the VLAN ID field blank. The allocatedValue parameter should once again be *1*.



16. Delete all created resources when finished.

NOTE: If you see additional resources with Resource Type ID starting with "ipam.resourceTypes", do not delete them, you will use them later in the labs.

End of Lab

Lab 4: Create Custom Validations, Operations, and Interfaces

Objectives

- Modify the L2VPN resource to support existing resources
- Add resource validations for your resources
- Implement custom operations for your resources

Documentation

PlanSDK documentation - *https://developer.blueplanet.com/docs/plansdk/index.html*

Custom operations - *https://developer.blueplanet.com/docs/bpocore-docs/references/custom_operations.html*

Resource validation - *https://developer.blueplanet.com/docs/bpocore-docs/references/resource_validations.html*

Interface definitions - *https://developer.blueplanet.com/docs/bpocore-docs/references/type_layer/resource_type.html#rt_interfaces*

Task 1: Modify L2VPN Resource to Support Existing Resources

In this first task, you modify your L2VPN resource so that it supports existing, already-created resources. In practice, many of the resources are created beforehand – in other words, customer resources might be created when your company acquires a new customer. Port resources might get created every time a new device is onboarded. Bandwidth Profiles might get created beforehand as a static configuration, and so on. This is why you modify your L2VPN resource so that it no longer creates all its sub-resources, but rather only creates relationships with already existing sub-resources.

1. This lab continues the work done in the previous labs. Make sure that you have successfully completed the previous lab. If you did not complete the previous lab, make sure to onboard the resource definitions, service templates, and code from that lab to bring your BPO server into the initial state required for this lab. You can find them in the **~/Desktop/learning/bpo-sdd/solutions** folder.
2. Your first steps are to modify the L2VPN resource in a way that the input parameters no longer represent all the parameters needed to create the sub-resources, but rather only represent the names of the already existing sub-resources with which a relationship should be formed.
3. In VS Code, open the **resource_type_l2vpn.tosca** file in the **training/model-definitions/types/tosca/training** folder. Delete all the properties that are not needed to reference existing resources and leave only the ones needed to form relationships with sub-resources or for the L2VPN resource itself:
    - vlanId
    - customerLabel
    - siteLabel
    - bandwidthProfile

4. This is how the L2VPN resource file should look like after the changes:

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)"  ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {
      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
        updatable = true
      }

      customerLabel {
        title = "Customer Label"
        description = "Label of the customer"
        type = string
      }

      siteLabel {
        title = "Site Label"
        description = "Label of the Site"
        type = string
      }

      bandwidthProfile {
        title = "Bandwidth Profile"
        description = "Label of the bandwidth profile"
        type = string
      }
    }
  }
}
```

5. Next, you need to modify your Python code for the L2VPN resource. Open the **l2vpn.py** file in the **training/model-definitions/training/l2vpn.py** file. You need to do the following steps:
    - Replace the **create_customer()** method with an **assign_customer()** method. This method should add a relationship between the L2VPN and the Customer resource.
    - Replace the **create_site()** method with **assign_site()** method. This method should add a relationship between the L2VPN and the Site resource.
    - Replace the **create_bandwithprofile()** method with **assign_bandwidthprofile()** method. This method should add a relationship between the L2VPN and the Bandwidth Profile resource.
6. Take care of the **assign_customer()** method first. Open the **PlanSDK** documentation and study the **bpo.resources.add_relationship()** method.

```
add_relationship(source_id, target_id, requirement='composed', capability='composable')
```

7. As you can see, you need both resource ID's. You can read the L2VPN resource ID through the **self.params** object, but for the Customer resource, you need to access it through the SDK. Open the

PlanSDK documentation and study the **bpo.resources.get_one_by_filters()** method. You can see that you can get resources by their resource type, and either by query filter (**q_params**) or partial match (**p_params**) parameters.

```
get_one_by_filters(resource_type, q_params=None, p_params=None, domain_id=None)
```

8. Implement the **assign_customer()** method. Get the customer resource by its **training.resourceTypes.Customer** type and label, using the query filter. The following output is how the finished method should look like.

```
    def assign_customer(self, properties):
        resource_id = self.params['resourceId']
        customer = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Customer",
                q_params={"label": properties['customerLabel']}
        )
        self.bpo.relationships.add_relationship(resource_id, customer['id'])
```

9. Implement the **assign_site()** method next in a similar manner. This is how the finished method should look like:

```
    def assign_site(self, properties):
        resource_id = self.params['resourceId']
        site = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Site",
                q_params={"label": properties['siteLabel']}
        )
        self.bpo.relationships.add_relationship(resource_id, site['id'])
```

10. Finally, implement the **assign_bandwidthprofile()** method. This is how the finished method should look like:

```
    def assign_bandwidthprofile(self, properties):
        resource_id = self.params['resourceId']
        site = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.BandwidthProfile",
                q_params={"label": properties['bandwidthProfile']}
        )
        self.bpo.relationships.add_relationship(resource_id, site['id'])
```

11. Once finished with the methods that form relationships, replace the calls for resource creation in the **run()** method with the new methods. Do not forget to modify the log messages as well, since the new methods do not return any objects. This is how the finished **run()** method in the **Activate** class from the **l2vpn-py** file should be:

```
class Activate(Plan):
    """
    Create L2VPN resource into market
    Create relationship with Customer, Site, and Bandwidth Profile resource
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        properties = resource['properties']
        self.assign_customer(properties)
        log.info(f"Assign Customer resource: {properties['customerLabel']}")

        self.assign_site(properties)
        log.info(f"Assign Site resource: {properties['siteLabel']}")

        self.assign_bandwidthprofile(properties)
        log.info(f"Assign Bandwidth Profile resource: {properties['bandwidthProfile']}")
```

```
        self.create_vlan_pool()
        self.allocate_vlan(resource)

        log.info("Activate: DONE")
        return {}
```

12. Next, modify the **Terminate** class in the same file. Since you are working with already existing resources on BPO, you do not want to delete them when you terminate the L2VPN resource as you did in the previous tasks. Instead, you only want to delete the relationships towards the Customer, Site, and Bandwidth Profile resources. Open the **PlanSDK** documentation and study the **delete_source_relationships()** method.

```
delete_source_relationships(source_id)
```

NOTE: You can also find a **delete_relationship()** method, but that one is used to delete relationships attached to a given target, instead of source.

13. Implement the **delete_l2vpn_relationships()** method. This is how the method should look like.

```
    def delete_l2vpn_relationships(self, resource_id):
        self.bpo.relationships.delete_source_relationships(resource_id)
```

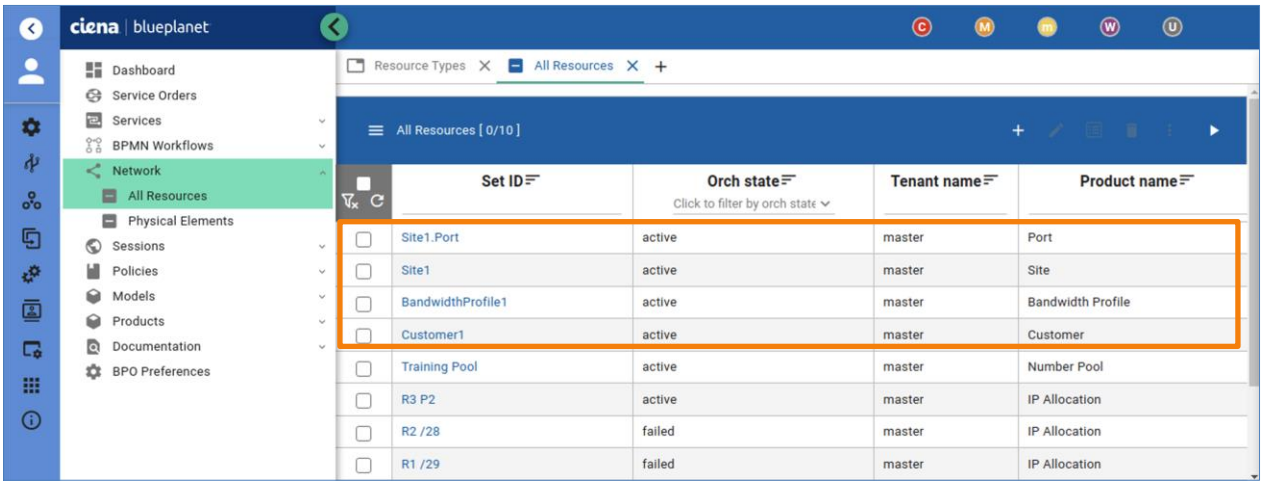
14. Call this method from the **run()** method in the **Terminate** class. Make sure to call it after the VLAN is unassigned. This is how the **run()** method should look in the end.

```
class Terminate(Plan):
    """
    Delete L2VPN resource from market
    Delete Customer, Site, and Bandwidthprofile relationships
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        label = resource['label']
        try:
            vlan_num_res = self.bpo.resources.get_one_by_filters(
                    resource_type=POOL_NUMBER_RESOURCE_TYPE,
                    q_params={"label": f"{label}.VLAN"}
            )
            self.bpo.resources.delete(vlan_num_res['id'])
            self.bpo.resources.await_termination(vlan_num_res['id'], f"{label}.VLAN", False)
            log.info(f"VLAN for {label} released")
        except:
            log.info(f"VLAN for {label} not found")

        self.delete_l2vpn_relationships(resource_id)
        log.info("Deleting Customer, Site and BandwidthProfile relationships")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

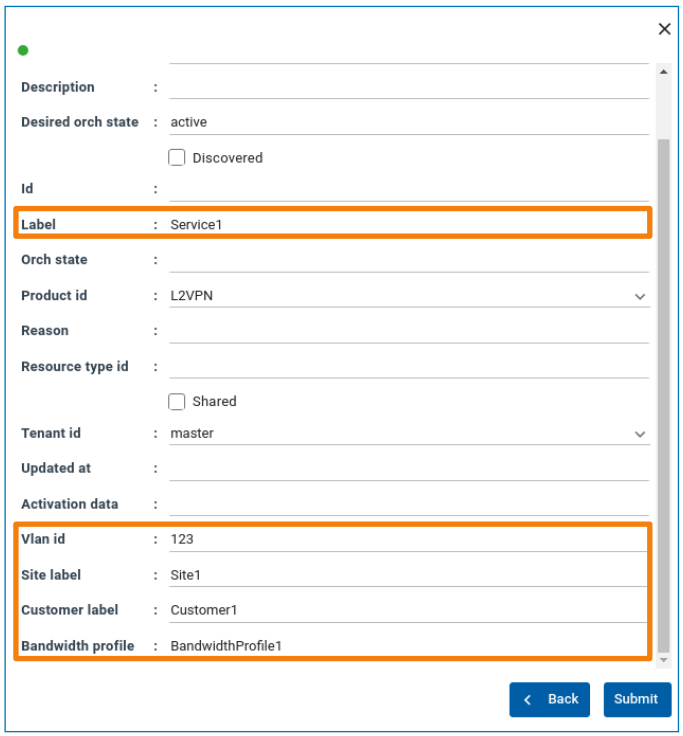
15. Onboard your changes to the BPO server through the **Visual Studio Code Blue Planet extension**.

NOTE: Sometimes backward incompatible changes are made to model definitions, such as in this case. You will get a warning that the changes conflict with existing products. If this happens, just delete the Products in question from the BPO UI and open a pull request again.

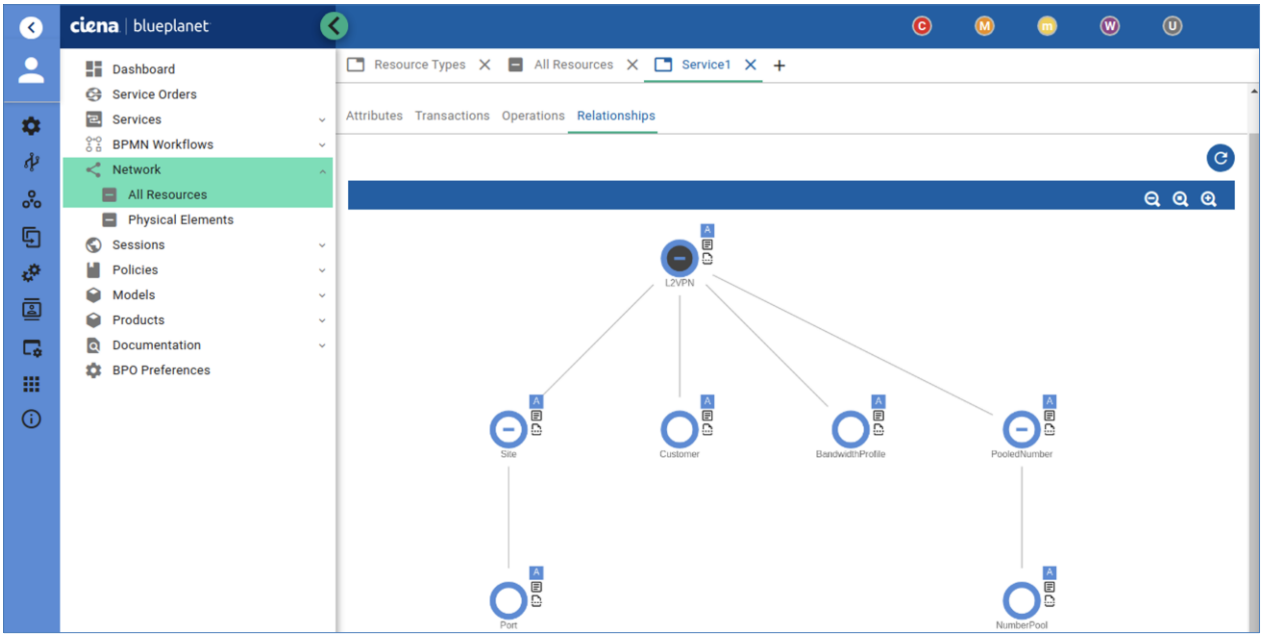
16. Create the **Customer**, **Bandwidth Profile** and **Site** resources first. To do this, navigate to the **Network > All Resources** tab, click the **+** symbol in the upper-right corner and choose the desired resource types. Mind the resource labels since you will use these same labels when creating the L2VPN resource.



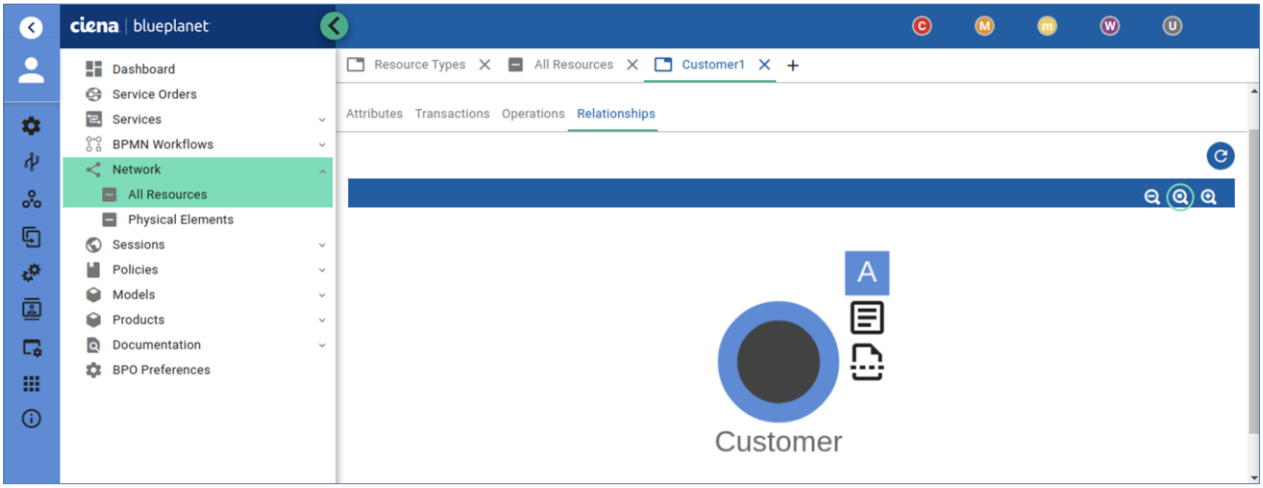
17. Create the **L2VPN** resource now. Use the appropriate resource labels from the existing resources. You can either assign a custom VLAN or leave it be. The resource should be created with no errors. If there are some, inspect the error message and fix the issue.



18. Inspect the newly created L2VPN service by clicking on it and opening the **Relationships** tag. Expand the **Site** and **PooledNumber** relationships by clicking on the **+** button inside the blue circle, and the relationships should look like the following figure.



19. Next, delete the L2VPN **Service1** resource. Make sure to only delete this resource and not the Customer, Site, and Bandwidth Profile resources. The L2VPN and the VLAN resources should get deleted, while other, previously created resources should be left active.
20. Inspect the **Customer** resource and look at the relationships. It should have no active relationships, the same as other resources.



Task 2: Add Resource Validation for Your Resources

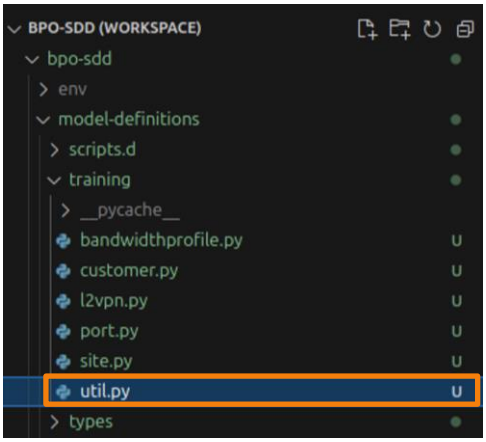
In this task, you add custom validation to your resources. Some validation is already built-in and performed whenever you create, delete, or modify a resource (for example – if some input requires an integer, the validation will prevent you from entering a string). Additionally, you can add your custom validation to reduce the number of errors that can happen due to misconfiguration. Keeping this in consideration, you create validation operations for all your resources.

1. You need to add custom validation for the following resources and operations:
    - Site:
        - No Site already exists with this siteId in the Activate phase.
        - No Site is using the same Port and Device in the Activate phase.
    - Port:
        - The same port on the device does not exist yet in the Activate phase.
    - Customer:
        - No Customer already exists with the same label or account number in the Activate phase.
    - Bandwidth Profile:
        - No Bandwidth Profile already exists with the same label in the Activate phase.
    - L2VPN:
        - No L2VPN resource already exists with the same label in the Activate phase.
        - Customer with the chosen label exists in the Activate phase.
        - Site with the chosen label exists in the Activate phase.
        - Bandwidth Profile with the chosen label exists in the Activate phase.
2. First, create a function that handles validation reporting. Study the resource validation documentation from the *Documentation* section of this lab. You will see that a validation script must return a Validation Report in the following form.

```
{
  state {
    title = State of the report
    description = Indicator of whether the validation overall is successful or failed.
    type = string
    enum = [ passed, failed ]
  }
  results {
    title = Detailed results of the validation
    description = Detailed results, each indicating whether a specific validation test passes or not. It
can be empty, but if specified, the logical conclusion of all the results should be coherent with the
overall state of the report.
    type = array
    items {
      type = object
      properties = {
        status {
          type = integer
          title = HTTP status code of the validation
        }
        pointer {
          type = string
          optional = true
          title = JsonPath pointer to place in the body that produces this result
        }
        detail {
          type = string
          optional = true
          title = Detail of the validation
        }
```

```
        validator {
          type = string
          optional = true
          title = Name of the validator that produces the result
        }
      }
    }
  }
}
```

3. This validation report could be created at the end of every validation method. However, this would cause a large amount of code duplication, which is best avoided. For this reason, you create a function that handles this report and use it in in all the validation methods. First, create a utility file called **util.py** in the **bpo-sdd/method-definitions/training** folder.



4. Add a **failure()** function inside **util.py**. You could also add a **success**() function, but since you are only interested in the status message of failed validations, you simply pass an empty object as the return of any successful validation.

```
def failure():
    """
    Validation report for failure
    """
```

5. Add a simple return statement to the failure method that returns a Python dictionary. According to the documentation, return the **state** parameter as **failed** and the **results** parameter as an array of objects. This array should contain one additional object, where you set the **status** parameter as the HTTP return code of the validation, and the **detail** parameter as a relevant message for why the validation failed. Pass this message and the status code as the function arguments and set the default value for the HTTP return code to **400** to indicate failure.

```
def failure(message, status_code=400):
    """
    Validation report for failure
    :param message:
    :param status_code:
    :return:
    """
    return {
        "state": "failed",
        "results": [
            {
                "status": status_code,
                "detail": message
            }
        ]
    }
```

6. Add a similar method for successful validation. Set the **state** parameter to **passed**, and the **status_code** to **200**.

```
def success(message, status_code=200):
    """
    Validation report for success
    :param message:
    :param status_code:
    :return:
    """
    return {
        "state": "passed",
        "results": [
            {
                "status": status_code,
                "detail": message
            }
        ]
    }
```

7. Similarly to the default lifecycle operations, the scripts that implement custom validate operations also need to be specified in the service template files. Open the **service_template_customer.tosca** file from the **bpo-sdd/model-definitions/types/tosca/training** folder. Add a new **validators** segment under **serviceTemplates**, which contains an **activate** object. Set the type to **remote**, language to **python**, and path to **training.customer.ValidateActivate** parameters. This is how the file should look like when finished.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer service template."
authors     = [ "Developer (developer@bptrn.com)"  ]

serviceTemplates = {
  Customer {
    title = "Customer"
    description = "A service template for Customer."
    implements = training.resourceTypes.Customer

    plans {
      activate {
        type = remote
        language = python
        path = training.customer.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.customer.Terminate
      }
    }

    validators {
      activate {
        type = remote
        language = python
        path = training.customer.ValidateActivate
      }
    }
  }
}
```

8. Add the remote script definition to the **Port** service template as well.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Port service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Port service template."
authors     = [ "Developer (developer@bptrn.com)" ]

serviceTemplates = {
  Port {
    title = "Port"
    description = "A service template for Port Workflow and operations."
    implements = training.resourceTypes.Port

    plans {
      activate {
        type = remote
        language = python
        path = training.port.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.port.Terminate
      }
    }

    validators {
      activate {
        type = remote
        language = python
        path = training.port.ValidateActivate
      }
    }
  }
}
```

9. Add the remote script definition to the **Site** service template as well.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Site service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Site service template."
authors     = [ "Developer (developer@bptrn.com)" ]

serviceTemplates = {
  Site {
    title = "Site"
    description = "A service template for Site Workflow and operations."
    implements = training.resourceTypes.Site

    plans {
      activate {
        type = remote
        language = python
        path = training.site.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.site.Terminate
      }
    }
```

```
    validators {
      activate {
        type = remote
        language = python
        path = training.site.ValidateActivate
      }
    }
  }
}
```

10. Add the remote script definition to the **Bandwidth Profile** service template as well.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Bandwidth Profile service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the BandwidthProfile service template."
authors     = [ "Developer (developer@bptrn.com)" ]

serviceTemplates = {
  BandwidthProfile {
    title = "BandwidthProfile"
    description = "A service template for BandwidthProfile"
    implements = training.resourceTypes.BandwidthProfile

    plans {
      activate {
        type = remote
        language = python
        path = training.bandwidthprofile.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.bandwidthprofile.Terminate
      }
    }

    validators {
      activate {
        type = remote
        language = python
        path = training.bandwidthprofile.ValidateActivate
      }
    }
  }
}
```

11. And finally, add the remote script definition to the **L2VPN** service template as well.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN service template."
authors     = [ "Developer (developer@bptrn.com)"  ]

serviceTemplates {

  L2VPN {
    title = "L2VPN"
    description = "A service template for L2VPN."
    implements = training.resourceTypes.L2VPN

    plans {
      activate {
```

```
        type = remote
        language = python
        path = training.l2vpn.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.l2vpn.Terminate
      }
    }

    validators {
      activate {
        type = remote
        language = python
        path = training.l2vpn.ValidateActivate
      }
    }
  }
}
```

12. Open the **port.py** file from the **bpo-sdd/model-definitions/training** folder. Add a new class called **ValidateActivate**, so that it matches the remote script location you defined in the service template validator section before. Add a **run()** method to this class.

```
...
class ValidateActivate(Plan):
    """
    Validate that the same port on the same device does not exist yet
    """
    def run(self):
```

13. Add import your **failure** and **success** functions from the **util.py** file.

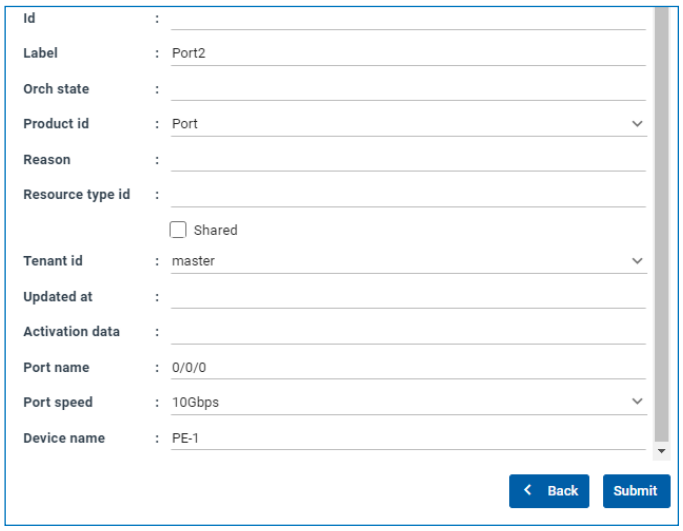
```
from plansdk.apis.plan import Plan
from .util import failure, success
import logging
...
```

14. The parameters passed to the validation callback differ a bit in structure from the ones that are passed to the activate callback. This is because the resource is not yet created in the first case. To inspect the difference, add a logging statement to the **ValidateActivate run()** method and return a successful validation report.

```
...

class ValidateActivate(Plan):
    """
    Validate that the same port on the same device does not exist yet
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        return success("Validation successful")
```

15. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

16. Create a new **Port** resource in the BPO UI.



17. Inspect the **scriptplan** logs, either in Kibana in BPO UI or in the **scriptplan** container to compare parameters passed to the validation callback and activation callbacks.

    a. The Validation parameters look like the following output (with relevant data highlighted).


```
{
   "marketwatchUrl":"http://127.0.0.1:41019",
   "marketwatchFeatures":[
      "resource1",
      "resourceOperation1",
      "minimum1"
   ],
   "bpUserId":"89f2242a-4100-43ab-aac1-2f5ca916aaac",
   "bpTenantId":"a9d4666a-e414-11ed-a6db-e330bea5476c",
   "bpRoleId":"0a421592-e415-11ed-b387-8b0414ffca16,0a407322-e415-11ed-b387-4b41a6e4b6ca,0a426e20-e415-
11ed-b387-db05add626bf,0a40c188-e415-11ed-b387-eb03ebe9c11d,0a3f8ba6-e415-11ed-b387-d74f0cde79ba,0a3ec072-
e415-11ed-b387-9f7a7d619a52,0a3fcf6c-e415-11ed-b387-ebddde9b13b8,4df2d9fc-e229-4e6e-b1fa-8de28fcf633b",
   "bpSubdomainId":"",
   "operation":"660be9ff-d2ec-462d-8be9-ffd2ec262d62.validate.activate",
   "logFile":"/bp2/log/plan-log/plan-script-660be9ff-d2ec-462d-8be9-ffd2ec262d62.validate.activate-2023-09-
06T07:15:18.974Z",
   "requestId":"3c1c42d7-5ac5-4cfc-92a2-ef51d8667ae7",
   "traceId":"ae896f89-4bbd-449a-883c-4d29f16d3f60",
   "inputs":{
      "full":true,
      "resource":{
         "productId":"d920fce7-4bdc-11ee-8314-eb8a23c9b0ba",
         "label":"Port2",
         "properties":{
            "portName":"0/0/0",
            "portSpeed":"10Gbps",
            "deviceName":"PE-1"
         },
         "discovered":false,
         "shared":false,
         "desiredOrchState":"active",
         "orchState":"requested",
         "reason":"",
         "tags":{

         },
         "autoClean":false,
         "providerData":{
```

```
        },
        "subDomainId":"7fd5144c-552f-39a3-9464-08d3b9cfb251",
        "activationData":{

        }
     }
  },
  "uri":"http://blueplanet"
}
```

b. While the Activate parameters should look like the following output (with resource ID already existing).

```
{
   "marketwatchUrl":"http://127.0.0.1:41019",
   "marketwatchFeatures":[
      "resource1",
      "resourceOperation1",
      "minimum1"
   ],
   "operation":"activate",
   "operationId":"26615b7f-4c85-11ee-8314-bd1625d673f6",
   "resourceId":"2285cefe-4c85-11ee-8314-9774f5899279",
   "requestId":"6cbab3df-e137-41b3-b526-07b66ede5bdd",
   "bpUserId":"89f2242a-4100-43ab-aac1-2f5ca916aaac",
   "bpTenantId":"a9d4666a-e414-11ed-a6db-e330bea5476c",
   "bpRoleId":"0a40c188-e415-11ed-b387-eb03ebe9c11d,0a3ec072-e415-11ed-b387-9f7a7d619a52,0a3fcf6c-e415-
11ed-b387-ebddde9b13b8,0a421592-e415-11ed-b387-8b0414ffca16,0a426e20-e415-11ed-b387-db05add626bf,0a407322-
e415-11ed-b387-4b41a6e4b6ca,4df2d9fc-e229-4e6e-b1fa-8de28fcf633b,0a3f8ba6-e415-11ed-b387-d74f0cde79ba",
   "logFile":"/bp2/log/plan-log/plan-script-2285cefe-4c85-11ee-8314-9774f5899279-26615b7f-4c85-11ee-8314-
bd1625d673f6-activate-2023-09-06T07:15:25.465Z",
   "bpSubdomainId":"",
   "uri":"http://blueplanet",
   "traceId":"ae896f89-4bbd-449a-883c-4d29f16d3f60"
}
```

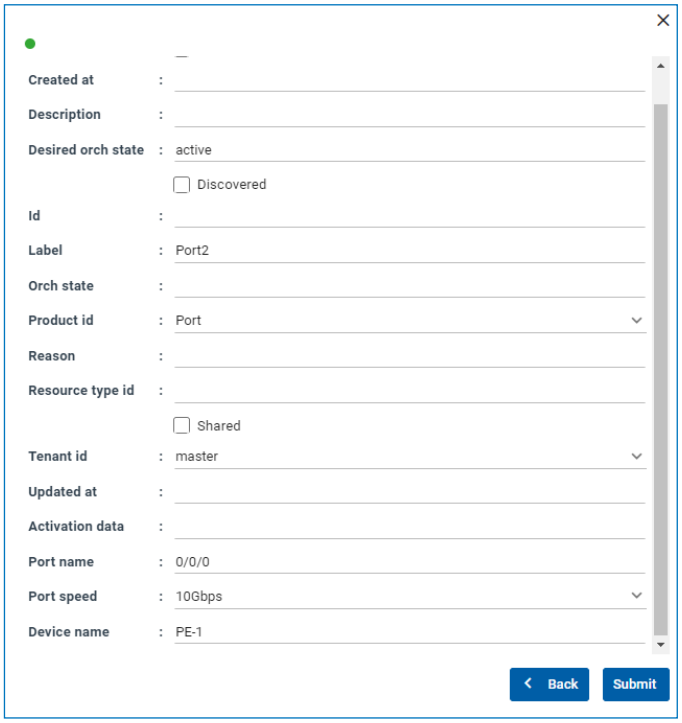
18. Delete the port resource you have created.
19. Now that you understand where to take the values for resource validation from, implement the Port **ValidateActivate run()** method so that it makes sure that the same port on the same device does not exist yet. In case it does, use the **failure** function you imported from the *util.py* file to properly format the validation failure response.

```
...

class ValidateActivate(Plan):
    """
    Validate that the same port on the same device does not exist yet
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')

        ports = self.bpo.resources.get_by_type("training.resourceTypes.Port")
        for port in ports:
            if (port['properties']['deviceName'] == properties['deviceName']) and
(port['properties']['portName'] == properties['portName']):
                return failure(f"Port with name ({properties['portName']}) already exists on device
{properties['deviceName']}")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

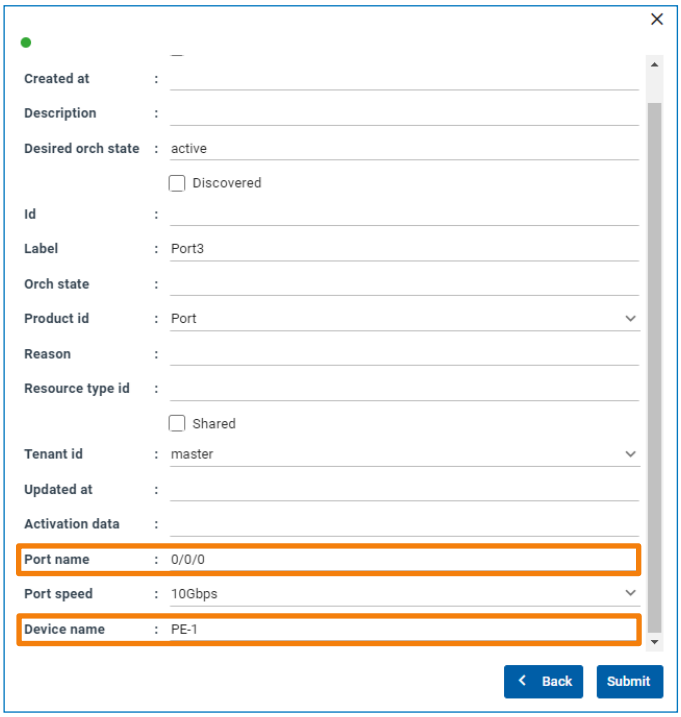
20. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

21. Try out your new validation operation.

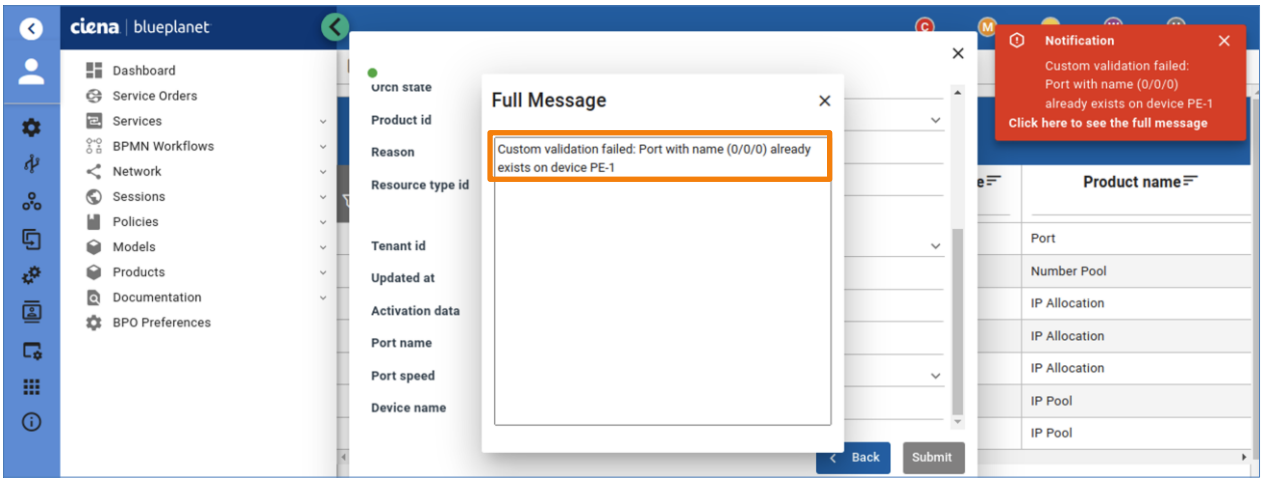
    c. Create a new Port resource again.



    d. Now create another port. Make sure that the **Port Name** and **Device Name** input parameters are the same as the ones you used for the Port from the previous step.



e. A small red box with a warning comes up in the upper-right corner of the screen. Click on the **Click here to see the full message** button to inspect the validation failure message, which should look like the following figure.



22. Delete the Port resource created in the previous step.
23. Next, open the **site.py** file and import the **failure** and **success** function.

```
from plansdk.apis.plan import Plan
from .util import failure, success
import logging
...
```

24. Add a **ValidateActivate** class with a **run()** method.

```
...
class ValidateActivate(Plan):
    """
    Verify that another Site does not exists with the same siteId or using the same Port
    Verify that another Site does not exists with the same portName and deviceName combination
    """
    def run(self):
```

25. Implement the **run()** method – you need to verify that no Site exists with the same *siteId*.

```
...
class ValidateActivate(Plan):
    """
    Verify that another Site does not exists with the same siteId or using the same Port
    Verify that another Site does not exists with the same portName and deviceName combination
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')

        sites = self.bpo.resources.get_by_type("training.resourceTypes.Site")
        for site in sites:
            if site['properties']['siteId'] == properties['siteId']:
                return failure(f"Site with siteId ({properties['siteId']}) already exists")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

26. Additionally, you need to verify that a site resource does not exist already with the same *portName* and *deviceName* combination.

```
...
class ValidateActivate(Plan):
    """
    Verify that another Site does not exists with the same siteId or using the same Port
    Verify that another Site does not exists with the same portName and deviceName combination
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')

        sites = self.bpo.resources.get_by_type("training.resourceTypes.Site")
        for site in sites:
            if site['properties']['siteId'] == properties['siteId']:
                return failure(f"Site with siteId ({properties['siteId']}) already exists")
            elif (site['properties']['deviceName'] == properties['deviceName']) and
(site['properties']['portName'] == properties['portName']):
                return failure("Site with the same portName and deviceName combination already exists")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

27. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
28. Test these changes by creating two new Site resources and trigger the validation failure by using the same site ID and the same device/port combination.
29. Take care of the Customer **ValidateActivate** class next. Edit the **customer.py** file and add it there.

```
from plansdk.apis.plan import Plan
from .util import failure, success
import logging

log = logging.getLogger(__name__)

...

class ValidateActivate(Plan):
    """
    Verify that another customer does not exists with the same customerName or customerAccountNumber
    """
    def run(self):
```

30. Implement the **run()** method. Make sure that the customer you are creating has a unique *customerName* and *customerAccountNumber*.

```
from plansdk.apis.plan import Plan
from .util import failure, success
import logging

log = logging.getLogger(__name__)

...

class ValidateActivate(Plan):
    """
    Verify that another customer does not exists with the same customerName or customerAccountNumber
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')
```

```
        customers = self.bpo.resources.get_by_type("training.resourceTypes.Customer")
        for customer in customers:
            if customer['properties']['customerName'] == properties['customerName']:
                return failure(f"Customer with customerName ({properties['customerName']}) already exists")
            elif customer['properties']['customerAccountNumber'] == properties['customerAccountNumber']:
                return failure(f"Customer with customerAccountNumber
({properties['customerAccountNumber']}) already exists")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

31. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
32. You can again test your validation implementation by creating a customer with a non-unique name or account number.
33. Edit the **bandwidthprofile.py** file now. The validation for this resource should make sure that it has a unique label. The following output shows what the finished **ActivateValidate** class should look like.

```
from plansdk.apis.plan import Plan
from .util import failure, success
import logging

log = logging.getLogger(__name__)

...

class ValidateActivate(Plan):
    """
    Verify that another BandwidthProfile does not exists with the same label
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')

        profiles = self.bpo.resources.get_by_type("training.resourceTypes.BandwidthProfile")
        for profile in profiles:
            if profile['label'] == resource.get('label'):
                return failure(f"BandwidthProfile with label ({resource.get('label')}) already exists")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

34. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
35. You can again test your validation implementation by creating a bandwidth profile with a non-unique label.
36. The final validations that you need to add are for the **l2vpn.py** file:
    - Validate that no L2VPN resource already exists with the same label in the Activate phase.
    - Validate that a Customer with the chosen label exists in the Activate phase.
    - Validate that a Site with the chosen label exists in the Activate phase.
    - Validate that a Bandwidth Profile with the chosen label exists in the Activate phase.

    a. Open the **l2vpn.py** file and add a **ValidateActivate** class with the **run()** method. Import the **failure** function from the *util.py* file as well.

```
from plansdk.apis.plan import Plan
from .util import failure, success
import logging

log = logging.getLogger(__name__)

...

POOL_NAME = "training_vlan_pool"
```

```
POOL_RESOURCE_TYPE = "tosca.resourceTypes.NumberPool"
POOL_NUMBER_RESOURCE_TYPE = "tosca.resourceTypes.PooledNumber"

class ValidateActivate(Plan):
    """
    Verify that BandwidthProfile resource exists
    Verify that Customer resource exists
    Verify that Site resource exist
    Verify that a L2VPN resource does not exist with the same label
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs['resource']
        properties = resource['properties']

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

b. Add the code that verifies if a customer exists.

```
class ValidateActivate(Plan):
    """
    Verify that BandwidthProfile resource exists
    Verify that Customer resource exists
    Verify that Site resource exist
    Verify that a L2VPN resource does not exist with the same label
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs['resource']
        properties = resource['properties']

        # Verify that Customer resource exists
        try:
            customer = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Customer",
                q_params={"label": properties['customerLabel']}
            )
        except:
            return failure(f"Customer with label ({properties['customerLabel']}) does not exist")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

c. Add the code that verifies if a bandwidth profile exists.

```
class ValidateActivate(Plan):
    """
    Verify that BandwidthProfile resource exists
    Verify that Customer resource exists
    Verify that Site resource exist
    Verify that a L2VPN resource does not exist with the same label
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs['resource']
        properties = resource['properties']

        # Verify that Customer resource exists
        try:
            customer = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Customer",
                q_params={"label": properties['customerLabel']}
            )
        except:
```

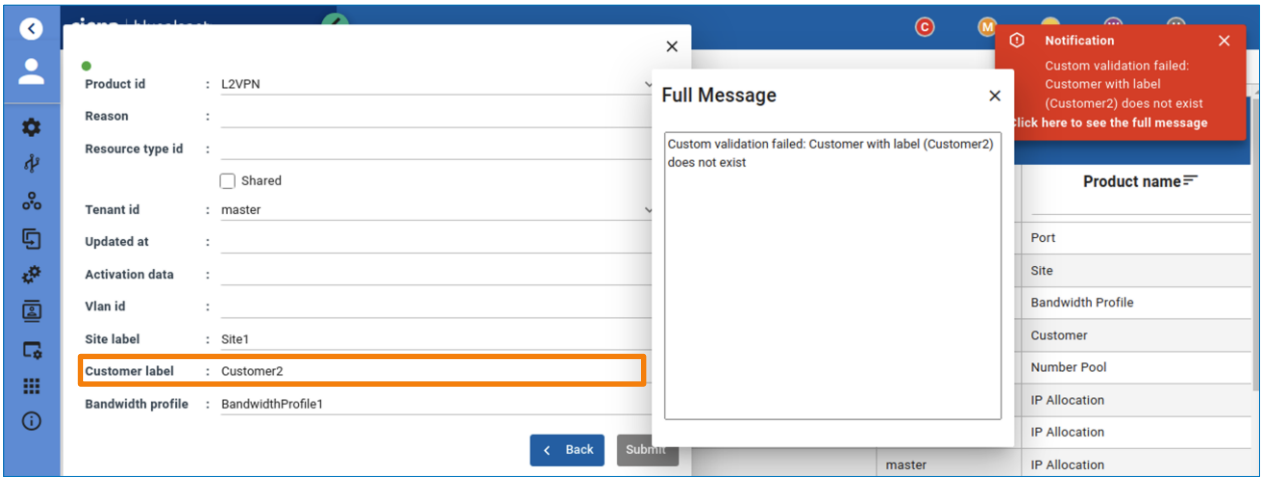
```
            return failure(f"Customer with label ({properties['customerLabel']}) does not exist")

        # Verify that BandwidthProfile resource exists
        try:
            bp = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.BandwidthProfile",
                q_params={"label": properties['bandwidthProfile']}
            )
        except:
            return failure(f"BandwidthProfile with label ({properties['bandwidthProfile']}) does not
exist")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

d. Add the code that verifies if a site exists.

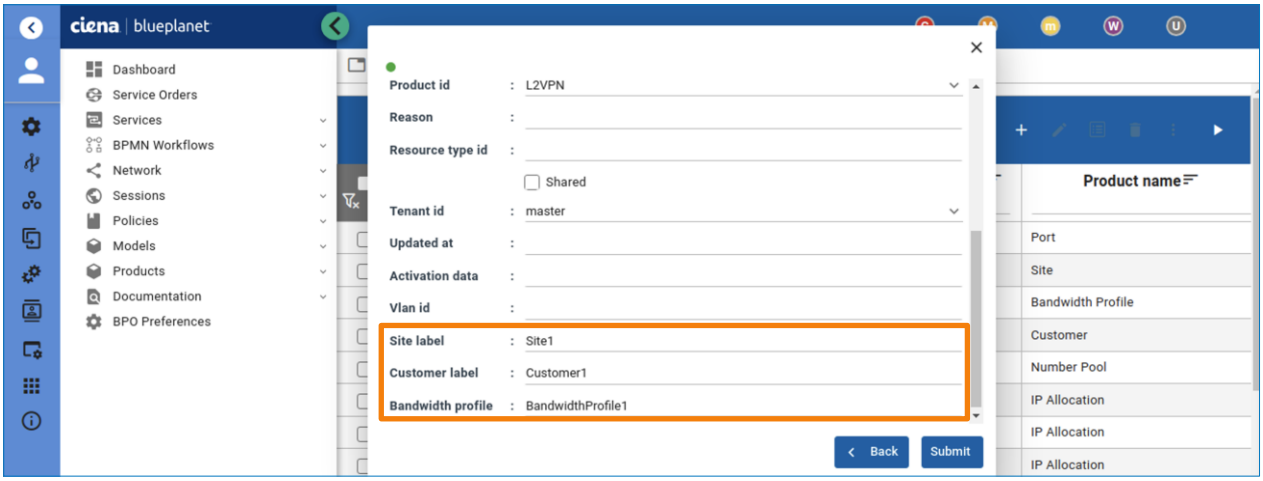
```
class ValidateActivate(Plan):
    """
    Verify that BandwidthProfile resource exists
    Verify that Customer resource exists
    Verify that Site resource exist
    Verify that a L2VPN resource does not exist with the same label
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs['resource']
        properties = resource['properties']

        # Verify that Customer resource exists
        try:
            customer = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Customer",
                q_params={"label": properties['customerLabel']}
            )
        except:
            return failure(f"Customer with label ({properties['customerLabel']}) does not exist")

        # Verify that BandwidthProfile resource exists
        try:
            bp = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.BandwidthProfile",
                q_params={"label": properties['bandwidthProfile']}
            )
        except:
            return failure(f"BandwidthProfile with label ({properties['bandwidthProfile']}) does not
exist")

        # Verify that Site resource exists
        try:
            site = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Site",
                q_params={"label": properties['siteLabel']}
            )
        except:
            return failure(f"Site with label ({properties['siteLabel']}) does not exist")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

37. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

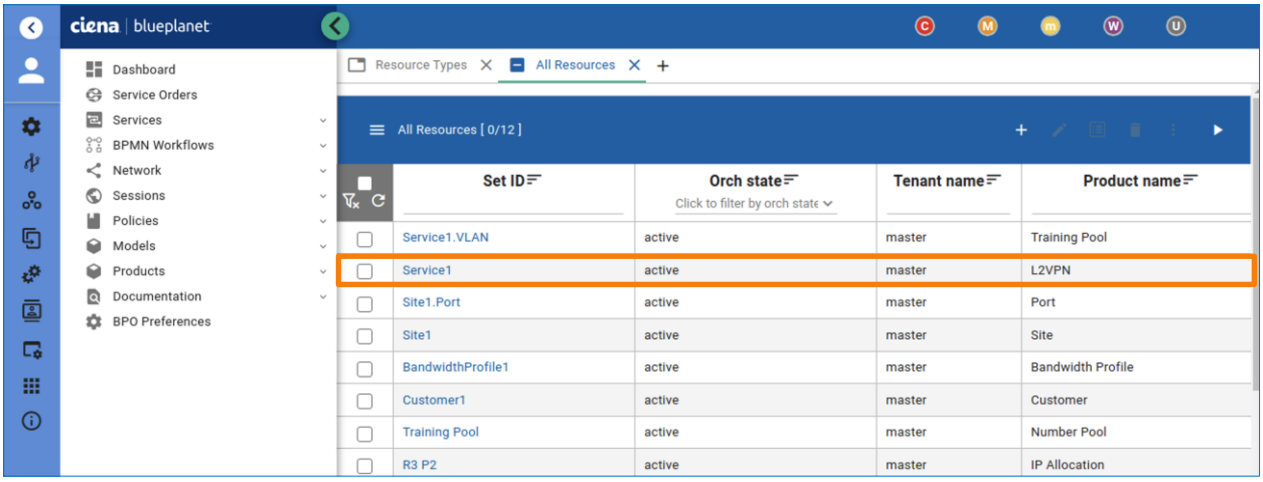
38. Test the L2VPN validation callbacks by creating a **L2VPN** resource (**label: Service1**) while referencing non-existing sub-resources. This should trigger a validation failure.



39. Finally, create a valid **L2VPN** resource (**label: Service1**) again, by referencing the existing **Site**, **Customer**, and **Bandwidth Profile** sub-resources.



40. Make sure that the L2VPN resource ends up in an active state and no validation errors occur.



Task 3: Create Custom Operations and Interfaces for Your Resources

In this task, you create a custom interface and a custom operation for your L2VPN resource. This custom operation and interface should allow you to change the Port attached to the L2VPN Site sub-resource, and its properties.

1. In VS Code, open the **service_template_l2vpn.tosca** file from the **bpo-sdd/model-definitions/types/tosca/training** folder. Add a new **plan** object there called **changePort**. The type should be **remote**, the language used **python**, and the remote script path should be **training.l2vpn.ChangePort**. The following output is how your file should look like after you add it.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN service template."
authors     = [ "Developer (developer@bptrn.com)"  ]

serviceTemplates {

  L2VPN {
    title = "L2VPN"
    description = "A service template for L2VPN."
    implements = training.resourceTypes.L2VPN

    plans {
      activate {
        type = remote
        language = python
        path = training.l2vpn.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.l2vpn.Terminate
      }

      update {
        type = remote
        language = python
        path = training.l2vpn.Update
      }

      changePort {
        type = remote
        language = python
        path = training.l2vpn.ChangePort
      }
    }

    validators {
      activate {
        type = remote
        language = python
        path = training.l2vpn.ValidateActivate
      }
    }
  }
}
```

2. While the *activate*, *terminate*, and *update* lifecycle operations already have their interfaces implicitly defined, you need to define interfaces for any custom operations you add. Interfaces are objects which define the inputs and outputs of an operation, along with some other parameters. Study the *Interface Definitions* documentation in the *Documentation* section of this lab.
3. Implement an interface for the *changePort* remote plan. Open the **resource_type_l2vpn.tosca** file and add an **interfaces** object under the L2VPN resource type.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)"  ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {
      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
        updatable = true
      }

      customerLabel {
        title = "Customer Name"
        description = "Label of the customer"
        type = string
      }

      siteLabel {
        title = "Site Label"
        description = "Label of the Site"
        type = string
      }

      bandwidthProfile {
        title = "Bandwidth Profile"
        description = "Name of the bandwidth profile"
        type = string
      }
    }

    interfaces {
    }
  }
}
```

4. Add a **changePort** interface to this interfaces object. Set the **title** and **description** parameters, as well as add **inputs** and **outputs** objects.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)"  ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {
      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
        updatable = true
      }

      customerLabel {
        title = "Customer Name"
        description = "Label of the customer"
        type = string
      }

      siteLabel {
        title = "Site Label"
        description = "Label of the Site"
        type = string
      }

      bandwidthProfile {
        title = "Bandwidth Profile"
        description = "Name of the bandwidth profile"
        type = string
      }
    }

    interfaces {
      changePort {
        title = "Change port"
        description = "Change port on a specified site for l2vpn resource"
        inputs {
        }

        outputs {
        }
      }
    }
  }
}
```

5. Specify the inputs for the Change Port custom operation. Outputs are not required for this operation. The inputs should match the input parameters used by the Port and Site resources and should be the following:
    - **portName** (string)

- **deviceName** (string)
- **portSpeed** (enum - 100Mbps, 1Gbps, 10Gbps)
- **siteLabel** (string)

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)"  ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {
      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
        updatable = true
      }

      customerLabel {
        title = "Customer Name"
        description = "Label of the customer"
        type = string
      }

      siteLabel {
        title = "Site Label"
        description = "Label of the Site"
        type = string
      }

      bandwidthProfile {
        title = "Bandwidth Profile"
        description = "Name of the bandwidth profile"
        type = string
      }
    }

    interfaces {
      changePort {
        title = "Change port"
        description = "Change port on a specified site for l2vpn resource"
        inputs {
          portName {
            title = "Port Name"
            description = "New port for the site"
            type = string
          }

          deviceName {
            title = "Device Name"
            description = "New device for the site"
            type = string
          }
```

```
          portSpeed {
            title = "Port Speed"
            description = "Speed of the port"
            type = string
            enum = [100Mbps, 1Gbps, 10Gbps]
          }

          siteLabel {
            title = "Site Label"
            description = "Site in which to change the port"
            type = string
          }
        }

        outputs {
        }
      }
    }
  }
}
```

6. Next you need to think about how you are going to handle the Port resource updates. You can either update the Port resource with **put** or **patch** methods or delete and then re-create the resource. In the instructions, the Port resource will be updated, which means that these properties need to have the **updateable = true** parameter set in the Port resource type. Open the **resource_type_port.tosca** file and add this parameter to all three properties.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Port resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Port resource type."
authors     = ["Developer (developer@bptrn.com)" ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  Port {
    derivedFrom = Root
    title = "Port"
    description = "The Port resource is used to define port information"

    properties {
      deviceName {
        title = "Device Name"
        description = "Name of the device"
        type = string
        updatable = true
      }

      portName {
        title = "Port Number"
        description = "Representing the port interface number"
        type = string
        updatable = true
      }

      portSpeed {
        title = "Port Speed"
        description = "Speed of the port"
        type = string
        enum = [100Mbps, 1Gbps, 10Gbps]
        updatable = true
```

```
      }
    }
  }
}
```

7. Since the port information is stored in the **Site** resource too, you need to make sure that the Site properties are updatable as well.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Site resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Site resource type."
authors     = [ "Developer (developer@bptrn.com)"  ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  Site {
    derivedFrom = Root
    title = "Site"
    description = "The Site resource is used to define the site information of the L2VPN service"

    properties {
      portName {
        title = "Port Name"
        description = "Label of the first Port that is to be used for service creation"
        type = string
        updatable = true
      }

      siteId {
        title = "Site Identifier"
        description = "Description of the Site that the port is from"
        type = string
        updatable = true
      }

      deviceName {
        title = "Device Name"
        description = "Name of the device"
        type = string
        updatable = true
      }

      portSpeed {
        title = "Port Speed"
        description = "Speed of the port"
        type = string
        enum = [100Mbps, 1Gbps, 10Gbps]
        updatable = true
      }
    }
  }
}
```

8. With interface, plans, and resource type definition taken care of, you now need to add the code that implements this Change Port operation. Open the **l2vpn.py** file from the **bpo-sdd/model-definitions/training** folder and add a **ChangePort** class, which contains a **run()** method. Add some basic logging as you did in the previous tasks.

```
...
class ChangePort(Plan):
    """
    Changes the Port and Device attached to selected Site in L2VPN resource
```

```
    """
    def run(self):
        log.info(f"ChangePort: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"ChangePort: resourceId {resource_id}")
        log.info(f"ChangePort: {resource}")

        log.info("ChangePort: DONE")
        return {}
```

9. Make a plan on how you are going to approach the Port resource update. In general, you need to make sure the following three things happen:
    - You need to verify that the Site to which the Port is attached exists.
    - You need to find which Port resource is attached to this Site.
    - You need to patch this Port resource.
    - You need to patch the Site resource as well.
10. First, verify that the Site resource exists.

```
...
class ChangePort(Plan):
    """
    Changes the Port and Device attached to selected Site in L2VPN resource
    """
    def run(self):
        log.info(f"ChangePort: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"ChangePort: resourceId {resource_id}")
        log.info(f"ChangePort: {resource}")

        operation = self.bpo.resources.get_operation(resource_id, self.params['operationId'])
        inputs = operation['inputs']

        log.info(f"ChangePort: inputs {inputs}")
        site_label = inputs['siteLabel']

        # Verify that the Site exists
        try:
            site_res = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": f"{site_label}"}
            )
            log.info(f"Site resource for {site_label} found.")
        except:
            raise Exception(f"Site resource for {site_label} not found.")

        log.info("ChangePort: DONE")
        return {}
```

NOTE: There are several ways you can handle expected exceptions in custom operations. The standard way is to either write to stderr and exit the script with a non-zero code, or simply raise an Exception. If a script fails due to an uncaught exception, the Python interpreter writes a traceback to stderr and exits with the status code 1.

11. Next, find the Port resource that is attached to this Site. The easiest way to do it is to find all the dependencies for this Site and filter out the ones with the Port resource type. Open the **PlanSDK** documentation and study the **bpo.resources.get_dependency_by_type()** method.

```
get_dependencies_by_type(resource_id, resource_type, recursive=False)
```

12. Use this method to find the Port resource. Raise an exception if the port is not found.

```
...
class ChangePort(Plan):
    """
    Changes the Port and Device attached to selected Site in L2VPN resource
    """
    def run(self):
        log.info(f"ChangePort: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"ChangePort: resourceId {resource_id}")
        log.info(f"ChangePort: {resource}")

        operation = self.bpo.resources.get_operation(resource_id, self.params['operationId'])
        inputs = operation['inputs']

        log.info(f"ChangePort: inputs {inputs}")
        site_label = inputs['siteLabel']

        # Verify that the Site exists
        try:
            site_res = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": f"{site_label}"}
            )
            log.info(f"Site resource for {site_label} found.")
        except:
            raise Exception(f"Site resource for {site_label} not found.")

        # Find the Port resource attached to this Site
        try:
            port_res = self.bpo.resources.get_dependency_by_type(site_res['id'],
"training.resourceTypes.Port")
            log.info(f"Port resource for {site_label} found - {port_res}")
        except:
            raise Exception(f"Port resource for {site_label} not found!")
        log.info(f"ChangePort: DONE")
        return {}
```

13. The last thing to do is to update the Port resource. Study the **bpo.resources.patch()** method in the *PlanSDK* documentation.

```
get_dependencies_by_type(resource_id, resource_type, recursive=False)
```

14. Use this method to update the Port resource.

```
...
class ChangePort(Plan):
    """
    Changes the Port and Device attached to selected Site in L2VPN resource
    """
    def run(self):
        log.info(f"ChangePort: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"ChangePort: resourceId {resource_id}")
        log.info(f"ChangePort: {resource}")

        operation = self.bpo.resources.get_operation(resource_id, self.params['operationId'])
        inputs = operation['inputs']

        log.info(f"ChangePort: inputs {inputs}")
```

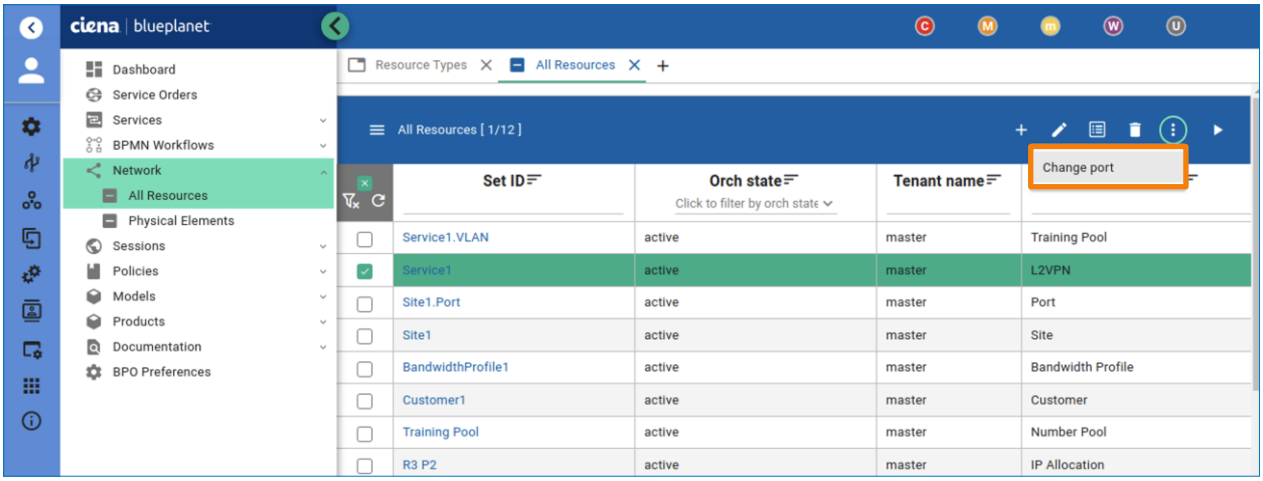
```
        site_label = inputs['siteLabel']

        # Verify that the Site exists
        try:
            site_res = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": f"{site_label}"}
            )
            log.info(f"Site resource for {site_label} found.")
        except:
            raise Exception(f"Site resource for {site_label} not found.")

        # Find the Port resource attached to this Site
        try:
            port_res = self.bpo.resources.get_dependency_by_type(site_res['id'],
"training.resourceTypes.Port")
            log.info(f"Port resource for {site_label} found - {port_res}")
        except:
            raise Exception(f"Port resource for {site_label} not found!")

        # Patch the Port resource
        self.bpo.resources.patch(port_res['id'], {
            "properties": {
                "portName": inputs['portName'],
                "deviceName": inputs['deviceName'],
                "portSpeed": inputs['portSpeed']
            }
        })
        log.info(f"Port resource for {site_label} patched.")

        log.info("ChangePort: DONE")
        return {}
```

15. Finally, patch the Site resource as well.

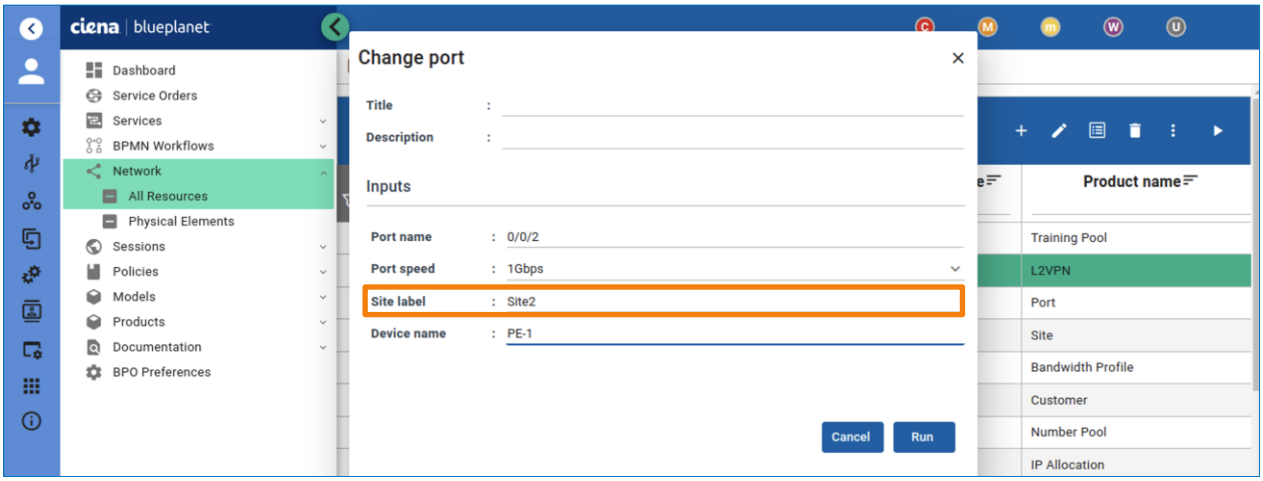
```
...
class ChangePort(Plan):
    """
    Changes the Port and Device attached to selected Site in L2VPN resource
    """
    def run(self):
        log.info(f"ChangePort: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"ChangePort: resourceId {resource_id}")
        log.info(f"ChangePort: {resource}")

        operation = self.bpo.resources.get_operation(resource_id, self.params['operationId'])
        inputs = operation['inputs']

        log.info(f"ChangePort: inputs {inputs}")
        site_label = inputs['siteLabel']

        # Verify that the Site exists
        try:
            site_res = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": f"{site_label}"}
            )
            log.info(f"Site resource for {site_label} found.")
        except:
            raise Exception(f"Site resource for {site_label} not found.")

        # Find the Port resource attached to this Site
        try:
```

```
            port_res = self.bpo.resources.get_dependency_by_type(site_res['id'],
"training.resourceTypes.Port")
            log.info(f"Port resource for {site_label} found - {port_res}")
        except:
            raise Exception(f"Port resource for {site_label} not found!")

        # Patch the Port resource
        self.bpo.resources.patch(port_res['id'], {
            "properties": {
                "portName": inputs['portName'],
                "deviceName": inputs['deviceName'],
                "portSpeed": inputs['portSpeed']
            }
        })
        log.info(f"Port resource for {site_label} patched.")

        # Patch the Site resource
        self.bpo.resources.patch(site_res['id'], {
            "properties": {
                "portName": inputs['portName'],
                "deviceName": inputs['deviceName'],
                "portSpeed": inputs['portSpeed']
            }
        })
        log.info(f"Site resource for {site_label} patched.")

        log.info("ChangePort: DONE")
        return {}
```

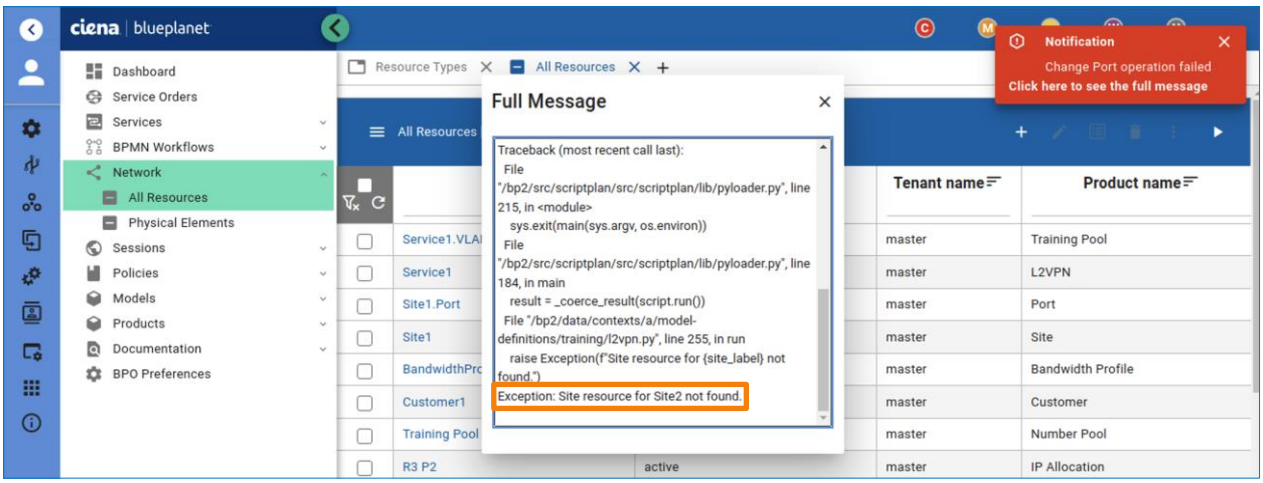
16. **Onboard** the changes to the BPO serve using the Blue Planet VS Code extension.
17. Open the BPO UI and navigate to the **Network > All Resources** tab. Choose your active **L2VPN** resource and click on the three dots (…) in the upper-right corner of the screen, denoting Operations. Click on the **Change Port** operation.



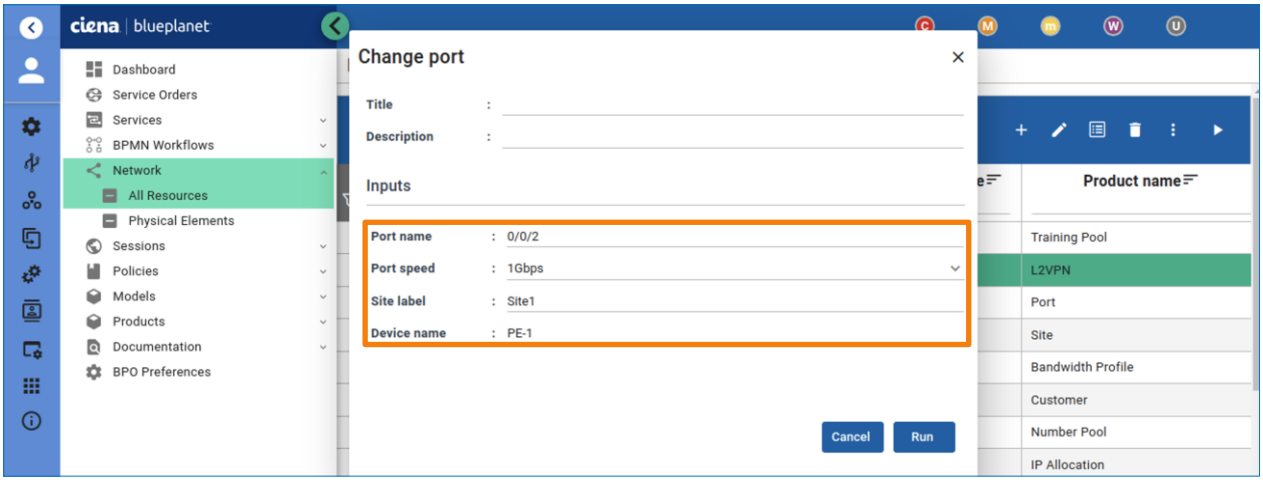
18. Input the required inputs – Port name, Port speed, Site label, and Device name. Initially, try to run this operation with a non-existing Site label and click **Run**.



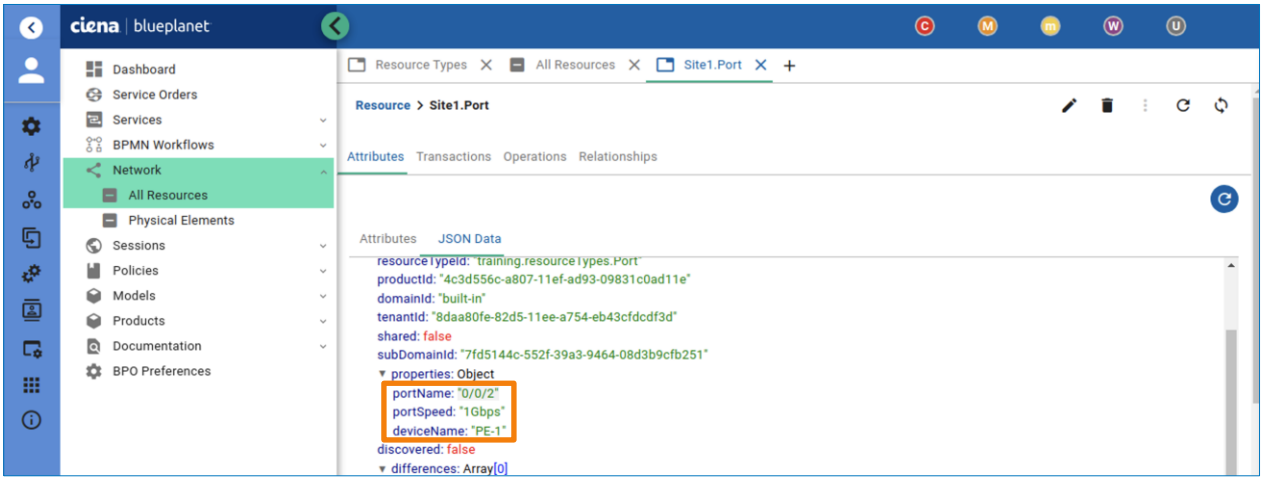
19. A notification pops up, telling you that the Change Port operation failed. Click on it to inspect the details. You see that your code triggered an exception since the Site you entered does not exist.



20. Next, execute the **Change Port** operation in the same way, but reference an existing Site this time.



21. The operation should be completed successfully now. If not, examine the error message and troubleshoot the issue.
22. Inspect the **Port** resource attached to the Site you have entered. You see that the Port resource has now been updated successfully.

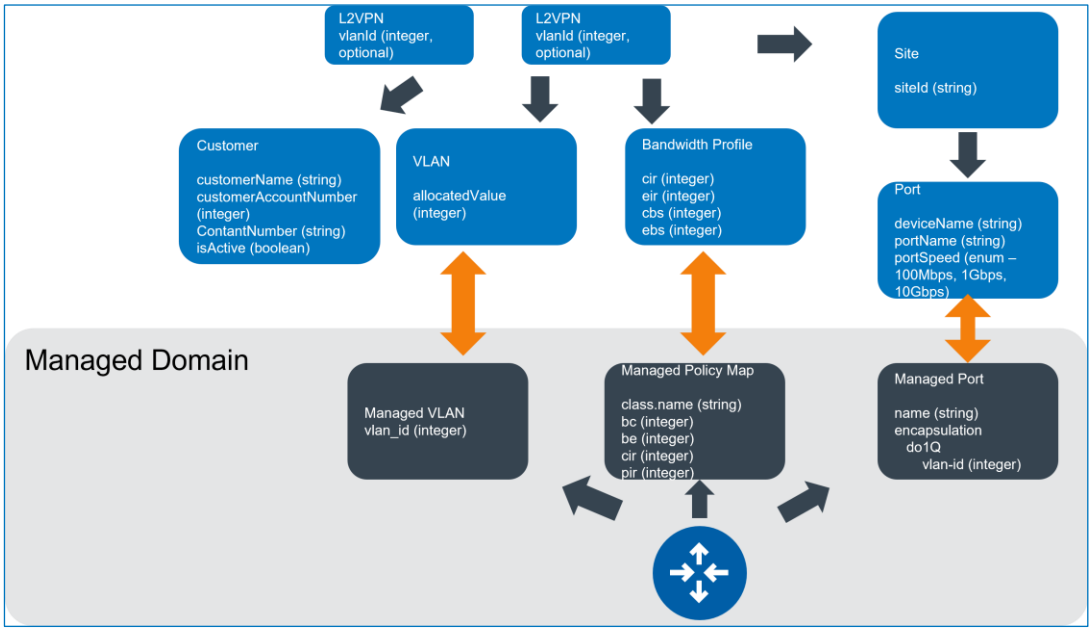


End of Lab

Lab 5: Integrate Services and Managed Domains

Objectives

- Add a managed domain and devices to BPO
- Integrate the L2VPN service and managed domain



Documentation

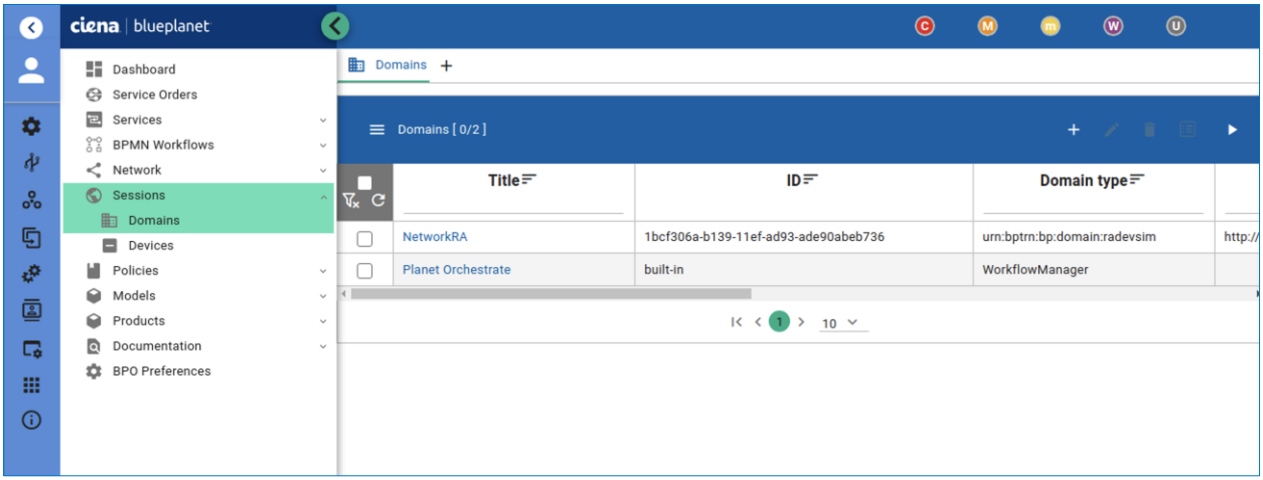
PlanSDK documentation: *https://developer.blueplanet.com/docs/plansdk/index.html*

Task 1: Add a Managed Domain and Managed Devices to BPO

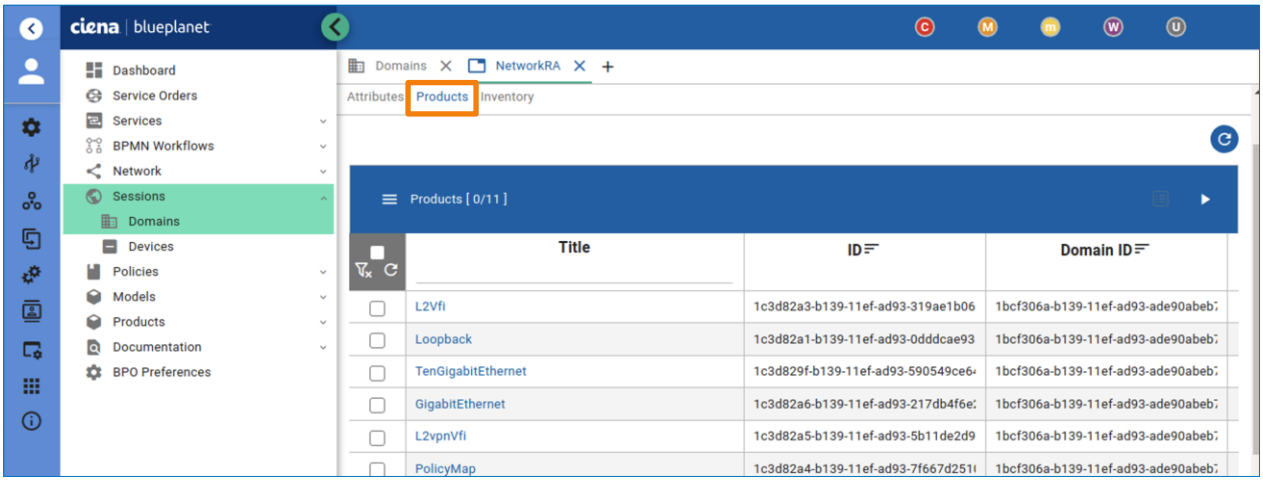
In this task, you review a managed domain in BPO. A managed domain is a set of devices or other resources, which is orchestrated by BPO over a Resource Adapter. In your case, the managed domain is a set of simulated network devices.

1. This lab continues the work done in the previous labs. Make sure that you have successfully completed the previous lab. If you did not complete the previous lab, make sure to onboard the resource definitions, service templates, and code from that lab to bring your BPO server into the initial state required for this lab. You can find them in the **~/Desktop/learning/bpo-sdd/solutions** folder.

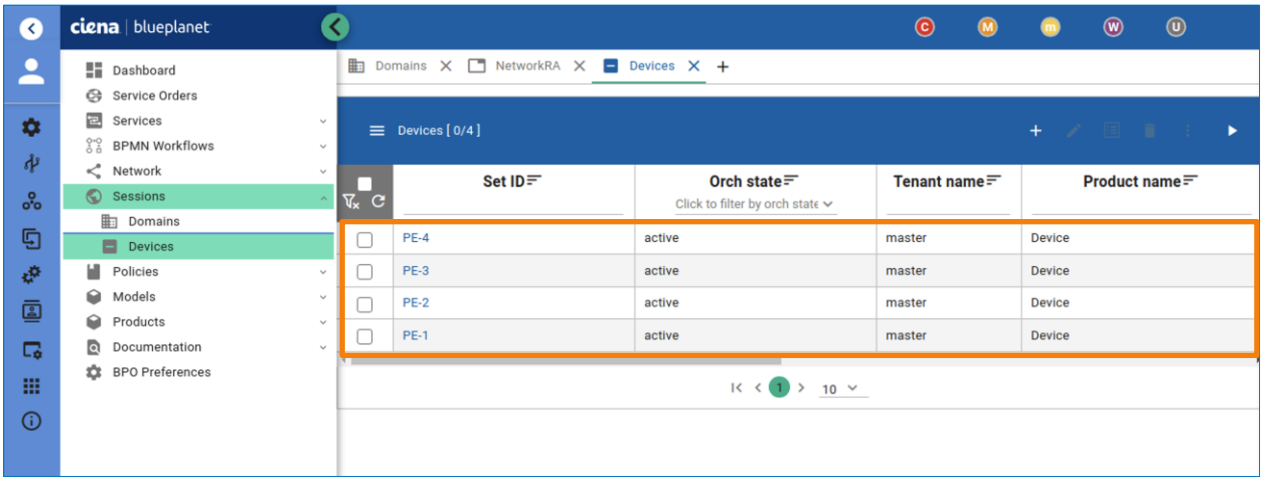
2. In BPO UI navigate to Sessions > Domains. This is how the Network RA domain should look:



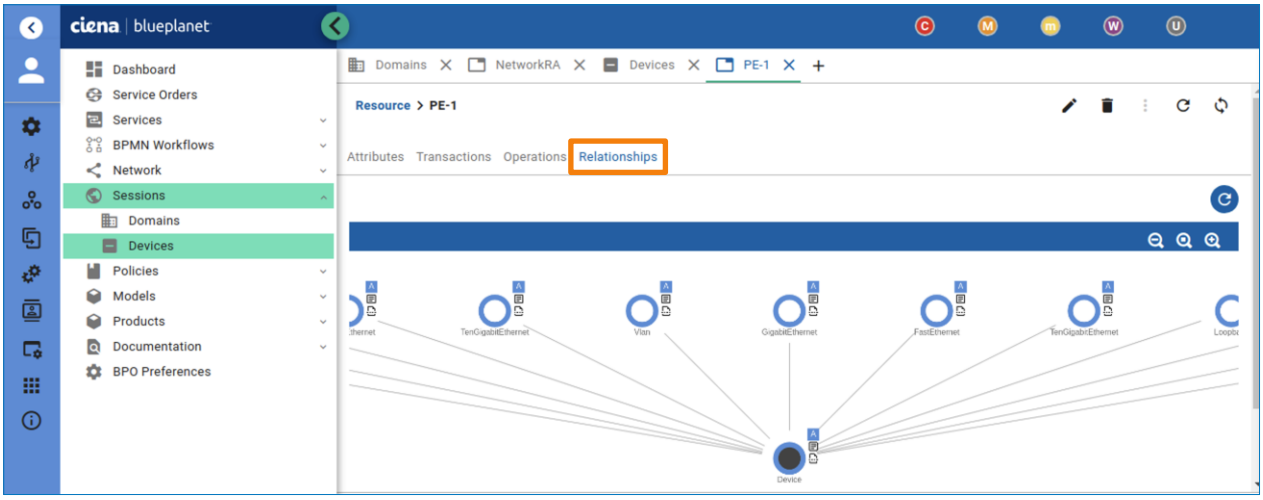
3. Click the **Network RA** domain to inspect it. Click the **Products** tab. Here, you will see which products are implemented for this managed domain.



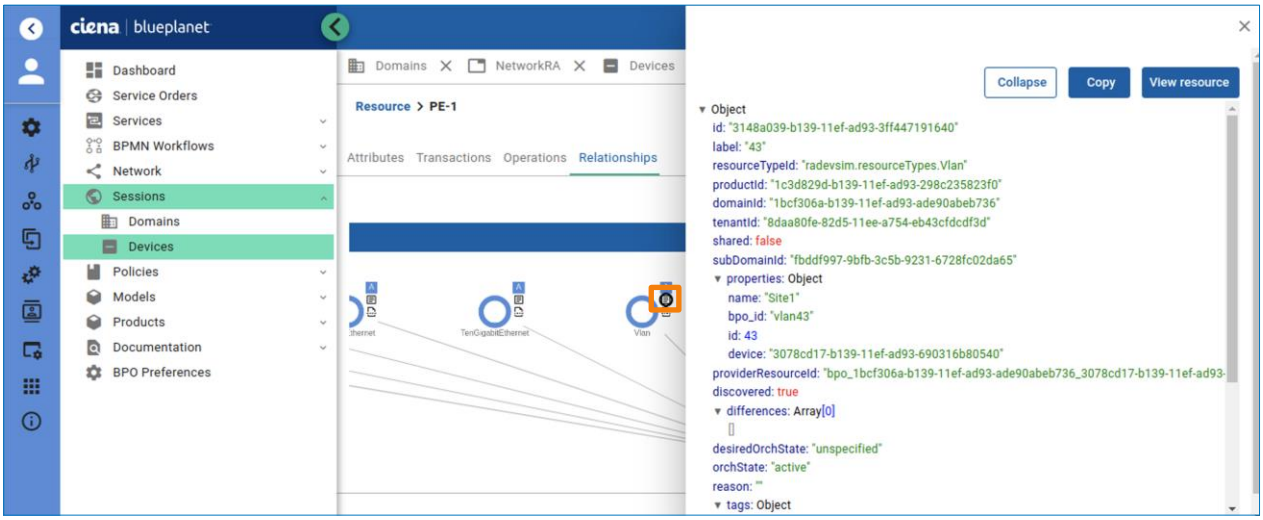
4. Navigate to **Sessions > Devices**. You will see the managed devices here.



5. Click one of the devices and open the **Relationships** tab. You see that adding a device also automatically added a few managed device resources, such as interfaces and VLAN. Modifying these resources and their properties will directly affect the managed devices.

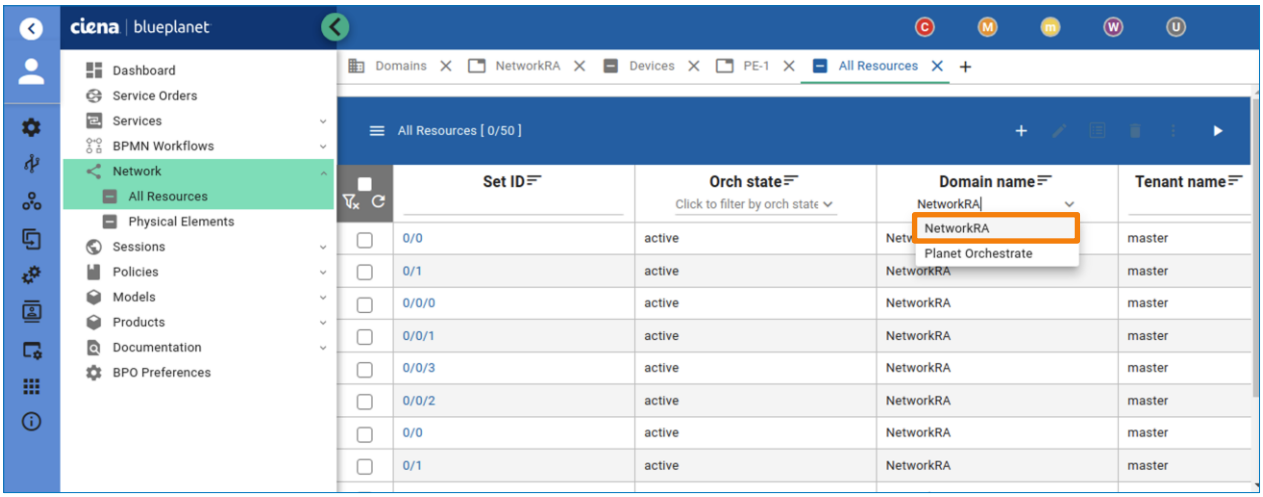


6. Click the second button by the VLAN resource. This will open a sidebar which will display the resource properties.



7. Inspect the other related resources in the same way so you know what you are working with.

   Open the **Network > All Resources** tab. All the managed resources are found here. You can add a filter to quickly see only these resources using the **Domain name** filter.



Task 2: Integrate the L2VPN Service and Managed Domain

In this next task, you integrate the existing L2VPN service and its resources with the Network RA managed domain. Make sure that a managed logical port is created, that a managed policy map is created based on the Bandwidth Profile, and that a managed VLAN resource is created when you create an L2VPN resource. To understand the objective more clearly, refer to the diagram at the beginning of this lab.

1. First, make sure that a logical port (subinterface) is created and that a relationship is formed between this managed logical port and the port resource. Since there are multiple different resource types that represent interfaces (GigabitEthernet, TenGigabitEthernet…), you need to include some logic that will figure out which logical port type needs to be created.
   a. In VS Code, open the **bpo-sdd/model-definitions/training/port.py** file. Add a new utility class called **PortUtils**. This class contains methods that are reused by different lifecycle operations, so you do not need to duplicate this code.

```
...
class PortUtils():
...
```

   b. Add a **get_managed_port_resource** method inside, which returns a managed Port resource based on **port name**, **device name**, and **port speed** parameters.

```
class PortUtils():
    def get_managed_port_resource(self, port_name, device_name, port_speed):

        return managed_port_resource
```

   c. Import the **ExactTypeId** class from **plansdk.apis.bpo** at the top of the file. This class is used to match an exact resource type for query purposes.

```
from plansdk.apis.plan import Plan
from plansdk.apis.bpo import ExactTypeId
from .util import failure, success
import logging
...
```

   d. Using if-else statements, set a **managed_port_resource_type** variable based on the **ExactTypeId** class. To find out the exact resource types, inspect the managed interfaces on the BPO UI. You see that you need to cover the following types:
      - radevsim.resourceTypes.GigabitEthernet
      - radevsim.resourceTypes.TenGigabitEthernet
      - radevsim.resourceTypes.HundredMegabitEthernet

```
class PortUtils():
    def get_managed_port_resource(self, port_name, device_name, port_speed):
        if port_speed == '1Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.GigabitEthernet")
        elif port_speed == '10Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.TenGigabitEthernet")
        elif port_speed == '100Mbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.HundredMegabitEthernet")
        else:
            return failure(f"{port_speed} is not a valid choice for Port Speed! Supported speeds are:
1Gbps, 10Gbps, 100Mbps")
        return managed_port_resource
```

   e. Since there are multiple ports with the same type, speed and name, you need to differentiate them based on the device where they are located. To do this, you need a similar **get_managed_device** method to get the managed device resource and its dependencies based on the device name. Add this method to the **PortUtils** class as well.

```
class PortUtils():
    def get_managed_port_resource(self, port_name, device_name, port_speed):
```

```
        if port_speed == '1Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.GigabitEthernet")
        elif port_speed == '10Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.TenGigabitEthernet")
        elif port_speed == '100Mbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.HundredMegabitEthernet")
        else:
            return failure(f"{port_speed} is not a valid choice for Port Speed! Supported speeds are:
1Gbps, 10Gbps, 100Mbps")
        return managed_port_resource

    def get_managed_device_resource(self, device_name):
        return managed_device_resource
```

f. Implement this method. Use the **bpo.resources.get_with_filters** method to get a list of all the managed devices (with **radevsim.resourceTypes.Device** resource type) and find out the device which has the desired name. Make sure to throw an exception if a device is not found! There are multiple ways to implement such filtering (nested for loops, lambda functions, and more).

```
class PortUtils():
    def get_managed_port_resource(self, port_name, device_name, port_speed):
        if port_speed == '1Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.GigabitEthernet")
        elif port_speed == '10Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.TenGigabitEthernet")
        elif port_speed == '100Mbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.HundredMegabitEthernet")
        else:
            return failure(f"{port_speed} is not a valid choice for Port Speed! Supported speeds are:
1Gbps, 10Gbps, 100Mbps")
             return managed_port_resource

    def get_managed_device_resource(self, device_name):
        managed_devices = self.bpo.resources.get_with_filters(
            ExactTypeId("radevsim.resourceTypes.Device")
        )
        try:
            managed_device_resource = [device for device in managed_devices if device['label'] ==
device_name][0]
        except:
            raise Exception(f"Managed device {device_name} does not exist!")
        return managed_device_resource
```

g. You can now use this **get_managed_device_resource()** method to get all dependent resources for the managed device using the **bpo.resources.get_dependents_with_filters()** method. Study this method in the PlanSDK documentation to understand how to use it and which parameters you need to add.

h. Continue implementing the **get_managed_port_resource()** method to get the managed port resource. Make sure you raise an exception if the managed port resource is not found!

```
class PortUtils():
    def get_managed_port_resource(self, port_name, device_name, port_speed):
        if port_speed == '1Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.GigabitEthernet")
        elif port_speed == '10Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.TenGigabitEthernet")
        elif port_speed == '100Mbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.HundredMegabitEthernet")
        else:
            return failure(f"{port_speed} is not a valid choice for Port Speed! Supported speeds are:
1Gbps, 10Gbps, 100Mbps")

        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)

        try:
```

```
            managed_device_all_ports =
self.bpo.resources.get_dependents_with_filters(managed_device_resource['id'], managed_port_resource_type)
            managed_port_resource = [port for port in managed_device_all_ports if port['label'] ==
port_name][0]
        except:
            raise Exception(f"Managed port {port_name} does not exist on device
{managed_device_resource['label']}!")
        return managed_port_resource

    def get_managed_device_resource(self, device_name):
        managed_devices = self.bpo.resources.get_with_filters(
            ExactTypeId("radevsim.resourceTypes.Device")
        )
        try:
            managed_device_resource = [device for device in managed_devices if device['label'] ==
device_name][0]
        except:
            raise Exception(f"Managed device {device_name} does not exist!")
        return managed_device_resource
```

2. Now that you have a way of finding out the managed port and managed device resources, you can use those methods in the **Port.ValidateActivate** class, to make sure that the device and port the user uses to create the Port resource actually exists.
   a. Add a check for the managed device resource to the **ValidateActivate run()** method. Make sure to return the **failure** response from the **util.py** file if it does not exist. Use your **get_managed_device** method from the **PortUtils** class.

```
class ValidateActivate(Plan):
    """
    Validate that the same port on the same device does not exist yet
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')

        ports = self.bpo.resources.get_by_type("training.resourceTypes.Port")
        for port in ports:
            if (port['properties']['deviceName'] == properties['deviceName']) and
(port['properties']['portName'] == properties['portName']):
                return failure(f"Port with number ({properties['portName']}) already exists on device
{properties['deviceName']}")

        # Verify that a managed device exists
        try:
            managed_device_resource = PortUtils.get_managed_device_resource(self, properties['deviceName'])
            log.info(f"Found managed device resource {managed_device_resource}")
        except:
            return failure(f"Managed device {properties['deviceName']} does not exist!")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

   b. Add a check for the managed port.

```
class ValidateActivate(Plan):
    """
    Validate that the same port on the same device does not exist yet
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')

        ports = self.bpo.resources.get_by_type("training.resourceTypes.Port")
```

```
        for port in ports:
            if (port['properties']['deviceName'] == properties['deviceName']) and
(port['properties']['portName'] == properties['portName']):
                return failure(f"Port with number ({properties['portName']}) already exists on device
{properties['deviceName']}")

        # Verify that a managed device exists
        try:
            managed_device_resource = PortUtils.get_managed_device_resource(self, properties['deviceName'])
            log.info(f"Found managed device resource {managed_device_resource}")
        except:
            return failure(f"Managed device {properties['deviceName']} does not exist!")

        # Verify that a managed port with chosen speed and label exist on managed device
        try:
            managed_port_resource = PortUtils.get_managed_port_resource(self, properties['portName'],
properties['deviceName'], properties['portSpeed'])
            log.info(f"Found managed port resource {managed_port_resource}")
        except:
            return failure(f"Unable to find managed port resource that matches device/port/speed
combination {properties['deviceName']} ({properties['portName']}) {properties['portSpeed']}")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

c. Update the class comment as well.

```
class ValidateActivate(Plan):
    """
    Validate that the same port on the same device does not exist yet
    Validate that a managed device exists
    Validate that a managed port exists
    """
    def run(self):
...
```

3. Edit the **Activate** class for the Port resource now. You need to read the appropriate managed port resource and create a relationship between the port resource as the source, and the managed port resource as the target. Use the **bpo.relationships.add.relationship()** method to do so.

```
class Activate(Plan):
    """
    Create Port resource into market
    Create relationship with managed port resource
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        # Create relationship with managed resource
        properties = resource['properties']
        managed_port_resource = PortUtils.get_managed_port_resource(self, properties['portName'],
properties['deviceName'], properties['portSpeed'])
        self.bpo.relationships.add_relationship(resource['id'], managed_port_resource['id'])

        log.info("Activate: DONE")
        return {}
```

4. As the last action in the **port.py** file, edit the **Terminate** class. During termination, make sure that the relationships where the Port is the source are deleted using **bpo.relationships.delete_source_relationship()** method.

```
class Terminate(Plan):
    """
    Delete Port resource from market
    Delete relationship with managed port resource
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        self.bpo.relationships.delete_source_relationships(resource_id

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Port has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

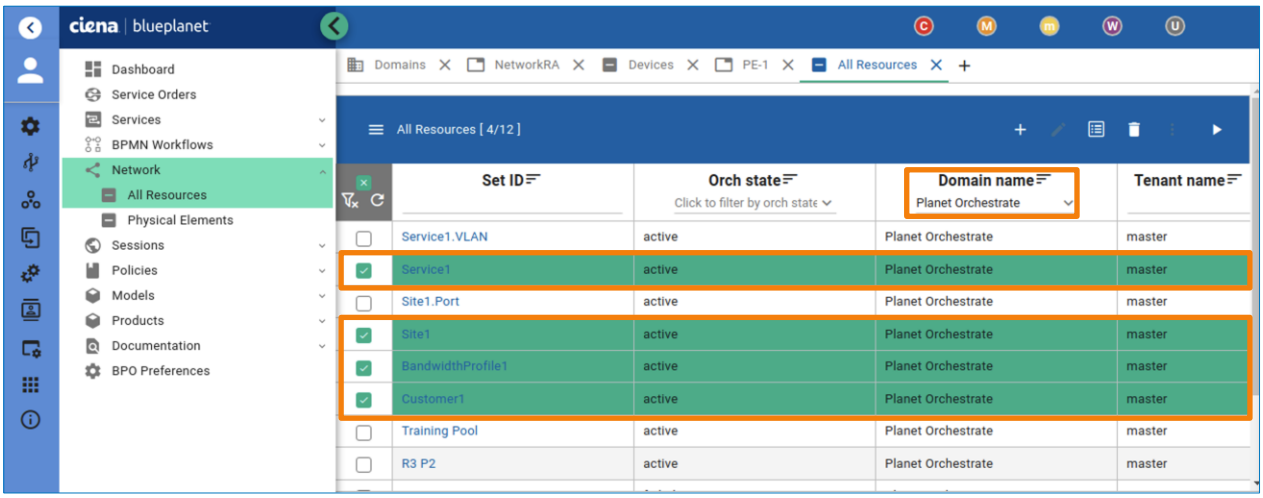
5. This is how the **port.py** file should look when you are finished:

```
from plansdk.apis.plan import Plan
from plansdk.apis.bpo import ExactTypeId
from .util import failure, success
import logging

log = logging.getLogger(__name__)

class PortUtils():
    def get_managed_port_resource(self, port_name, device_name, port_speed):
        if port_speed == '1Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.GigabitEthernet")
        elif port_speed == '10Gbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.TenGigabitEthernet")
        elif port_speed == '100Mbps':
            managed_port_resource_type = ExactTypeId("radevsim.resourceTypes.HundredMegabitEthernet")
        else:
            return failure(f"{port_speed} is not a valid choice for Port Speed! Supported speeds are:
1Gbps, 10Gbps, 100Mbps")

        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)

        try:
            managed_device_all_ports =
self.bpo.resources.get_dependents_with_filters(managed_device_resource['id'], managed_port_resource_type)
            managed_port_resource = [port for port in managed_device_all_ports if port['label'] ==
port_name][0]
        except:
            raise Exception(f"Managed port {port_name} does not exist on device
{managed_device_resource['label']}!")
        return managed_port_resource

    def get_managed_device_resource(self, device_name):
        managed_devices = self.bpo.resources.get_with_filters(
            ExactTypeId("radevsim.resourceTypes.Device")
        )
        try:
            managed_device_resource = [device for device in managed_devices if device['label'] ==
device_name][0]
        except:
            raise Exception(f"Managed device {device_name} does not exist!")
        return managed_device_resource
```

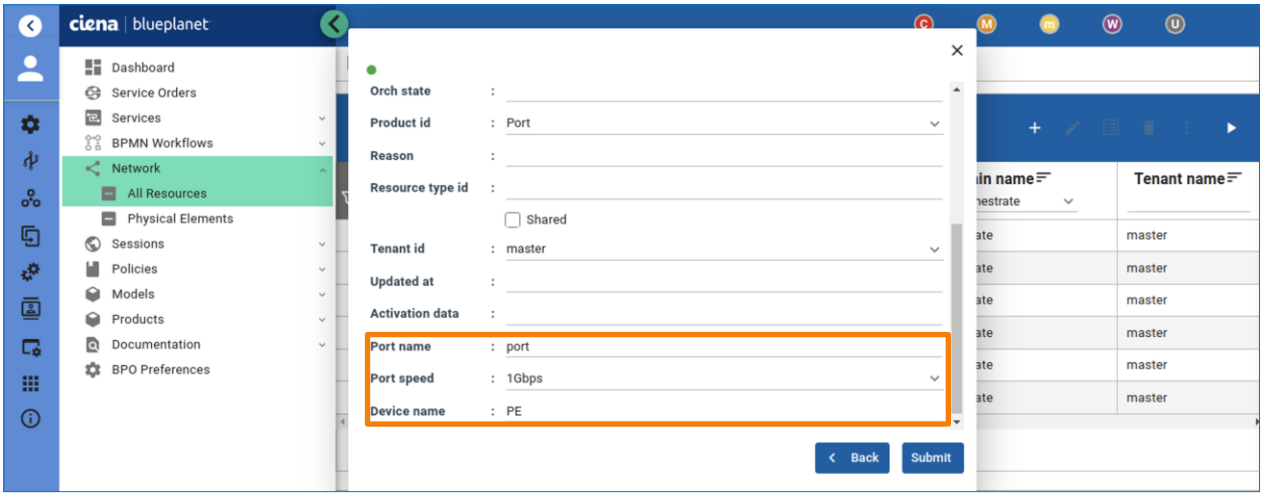
```
class ValidateActivate(Plan):
    """
    Validate that the same port on the same device does not exist yet
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')

        ports = self.bpo.resources.get_by_type("training.resourceTypes.Port")
        for port in ports:
            if (port['properties']['deviceName'] == properties['deviceName']) and
(port['properties']['portName'] == properties['portName']):
                return failure(f"Port with number ({properties['portName']}) already exists on device
{properties['deviceName']}")

        # Verify that a managed device exists
        try:
            managed_device_resource = PortUtils.get_managed_device_resource(self, properties['deviceName'])
            log.info(f"Found managed device resource {managed_device_resource}")
        except:
            return failure(f"Managed device {properties['deviceName']} does not exist!")

        # Verify that a managed port with chosen speed and label exist on managed device
        try:
            managed_port_resource = PortUtils.get_managed_port_resource(self, properties['portName'],
properties['deviceName'], properties['portSpeed'])
            log.info(f"Found managed port resource {managed_port_resource}")
        except:
            return failure(f"Unable to find managed port resource that matches device/port/speed
combination {properties['deviceName']} ({properties['portName']}) {properties['portSpeed']}")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")

class Activate(Plan):
    """
    Create Port resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        # Create relationship with managed resource
        properties = resource['properties']
        managed_port_resource = PortUtils.get_managed_port_resource(self, properties['portName'],
properties['deviceName'], properties['portSpeed'])
        self.bpo.relationships.add_relationship(resource['id'], managed_port_resource['id'])

        log.info("Activate: DONE")
        return {}

class Terminate(Plan):
    """
    Delete Port resource from market
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
```

```
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        self.bpo.relationships.delete_source_relationships(resource_id)

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Port has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

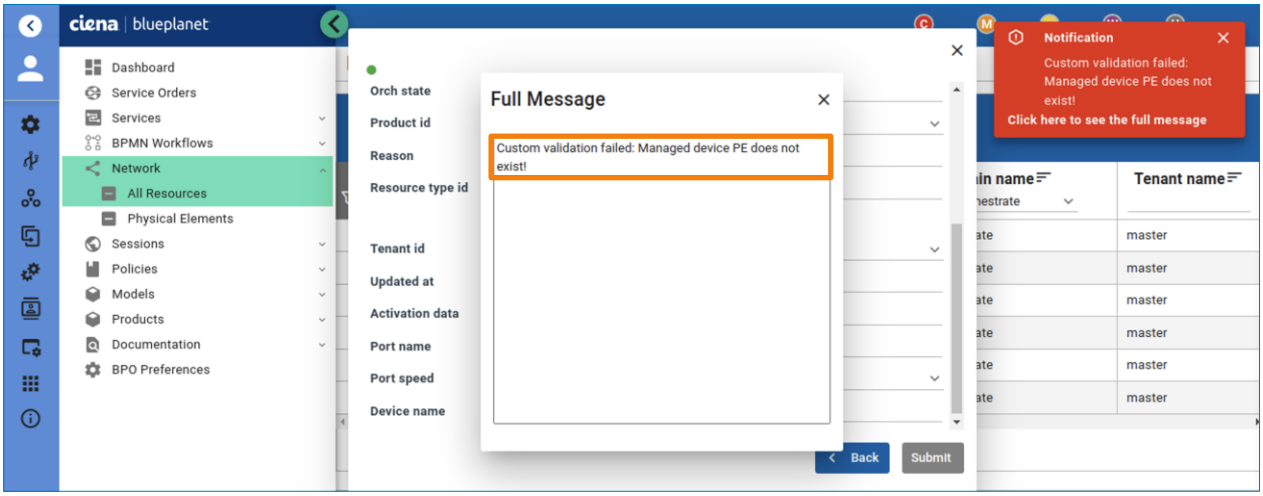
6. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
7. Open the BPO UI and navigate to the **Network > All Resources** view. Choose **Planet Orchestrate** as the Domain name. Delete any leftover resources from the previous labs that might exist. Do not delete the resources that have the Resource type ID starting with **ipam**.



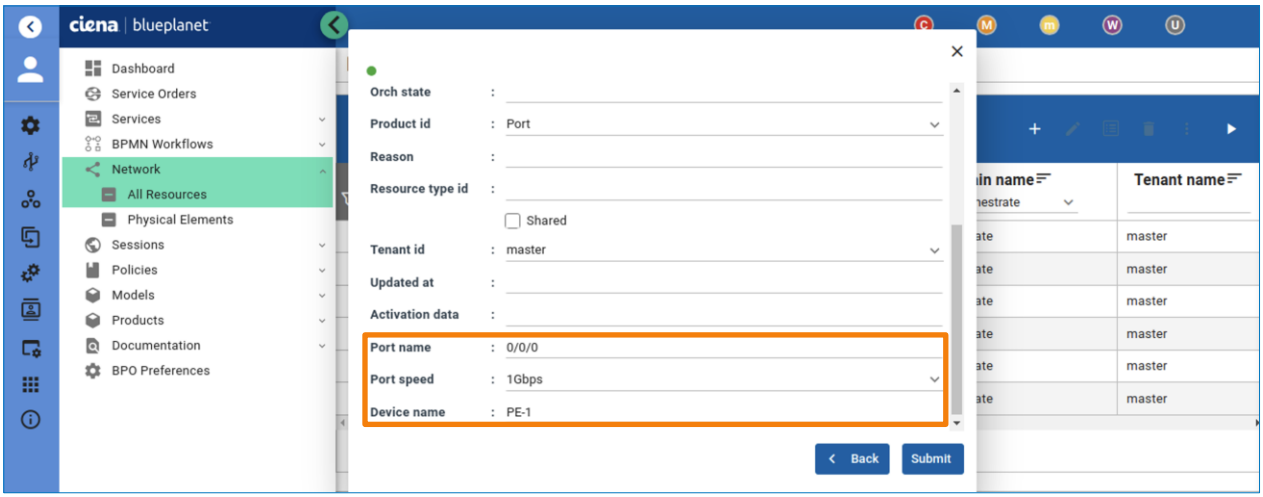
8. Create a new Port resource **TestPort**. Initially, use some parameters that do not match the ones that actually exist.



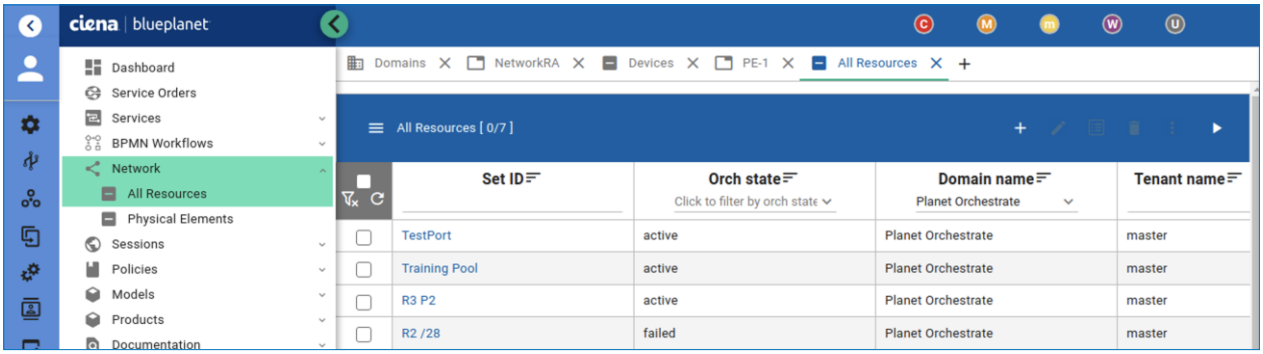
9. You get an error notification that a managed device/port does not exist.



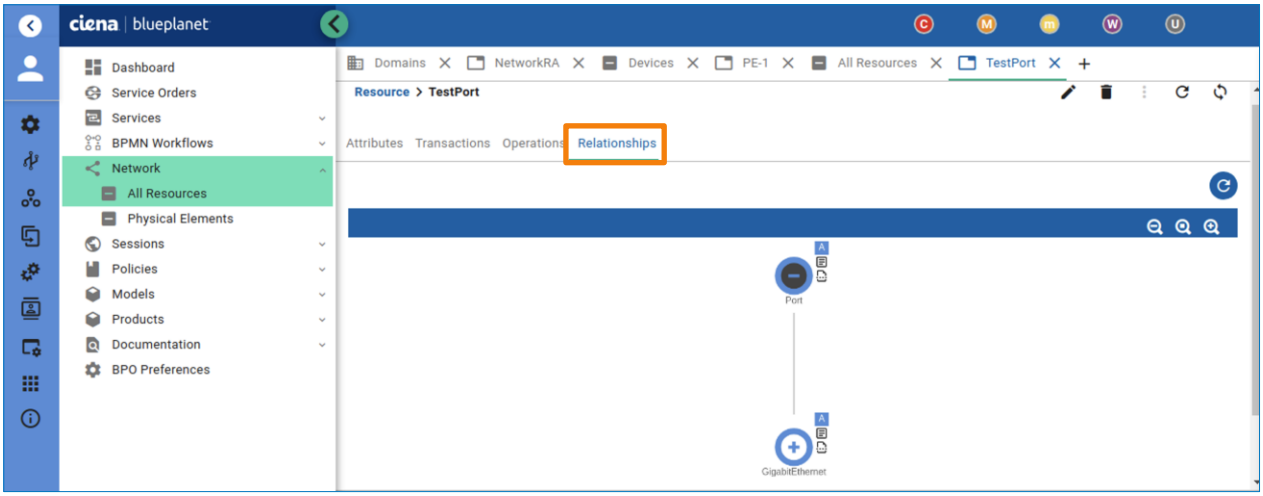
10. Now create another Port resource. This time, make sure you use parameters that match a port and device that actually exist, such as:
    - Label: **TestPort**
    - Port Name: **0/0/0**
    - Port Speed: **1Gbps**
    - Device name: **PE-1**



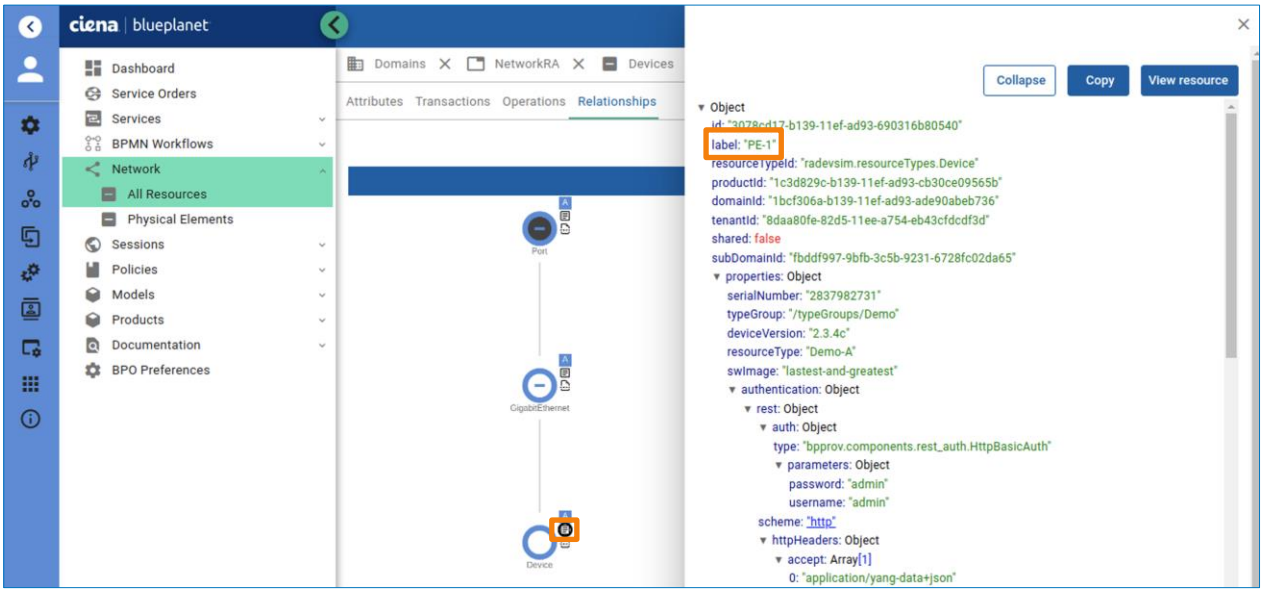
11. The Port resource should be successfully created and in an active state.



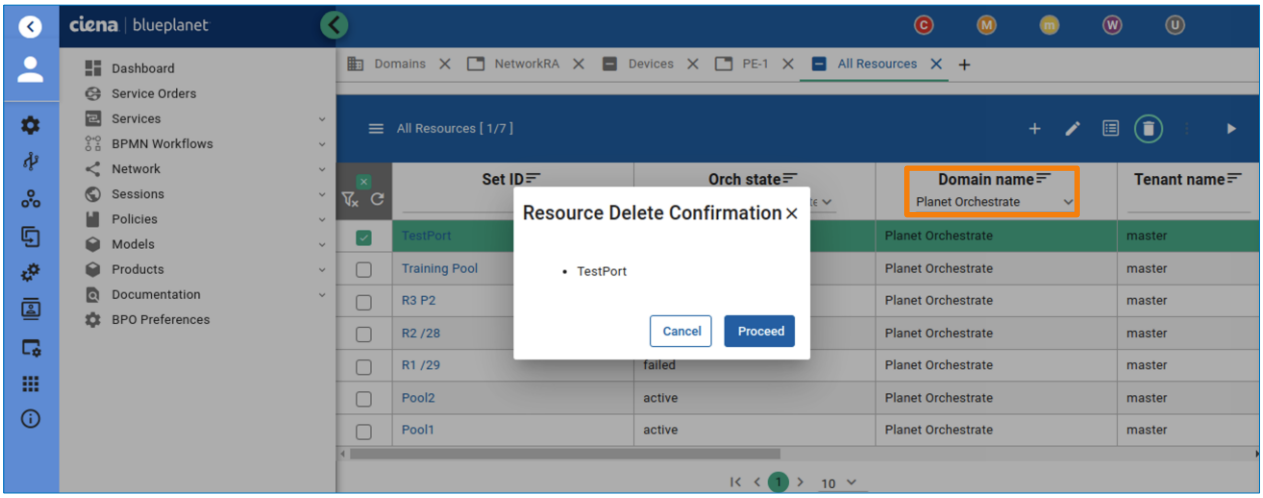
12. Examine the Port resource by clicking on it and inspecting the Relationships tab. If you have successfully implemented the code, a new relationship between the Port and the GigabitEthernet (managed port) resource is seen.



13. Click the **+** symbol in the managed port resource. You see that this resource has a further dependency on the managed device as well. Click the second button by the device resource to see which device this is and if it matches the one you entered when you created the Port resource.



14. Return to the **All Resources** tab and delete the created TestPort resource. The resource should be deleted without any issues.



15. Next, open the **site.py** file. Since you can create a Port resource at the same time that you create a Site resource (in its imperative plan), you need to add the same validation to the Site resource creation as you have added to the Port resource creation. Luckily, you do not need to implement this code again, but can just import the desired class and call its **run** method.
    a. Import the **ValidateActivate** class from the **port.py** file, give it a different name (since there already is a ValidateActivate class in this file), and call the **run** method from the validation plan for the Site resource.

```
from plansdk.apis.plan import Plan
from .util import failure, success
from .port import ValidateActivate as PortValidateActivate
import logging

log = logging.getLogger(__name__)

...

class ValidateActivate(Plan):
    """
    Verify that another Site does not exists with the same siteId or using the same Port
    Verify that another Site does not exists with the same portName and deviceName combination
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')

        sites = self.bpo.resources.get_by_type("training.resourceTypes.Site")
        for site in sites:
            if site['properties']['siteId'] == properties['siteId']:
                return failure(f"Site with siteId ({properties['siteId']}) already exists")
            elif (site['properties']['deviceName'] == properties['deviceName']) and
(site['properties']['portName'] == properties['portName']):
                return failure(f"Site with the same portName and deviceName combination already exists")

        PortValidateActivate.run(self)

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

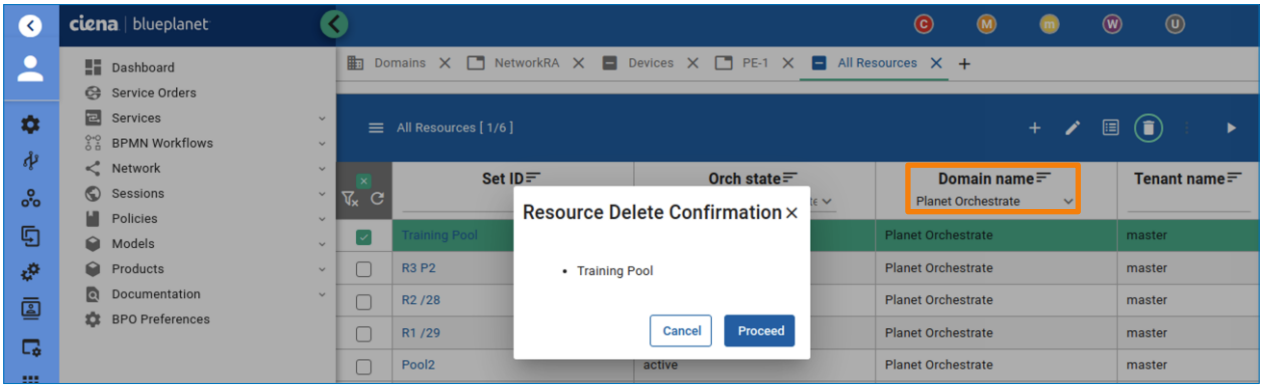
b. Add an if-clause which checks the **state** of this validation. Return the validation dictionary in case of failure, so a validation error triggers on the UI.

```
from plansdk.apis.plan import Plan
from .util import failure, success
from .port import ValidateActivate as PortValidateActivate
import logging

log = logging.getLogger(__name__)

class ValidateActivate(Plan):
    """
    Verify that another Site does not exists with the same siteId or using the same Port
    Verify that another Site does not exists with the same portName and deviceName combination
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs.get('resource')
        properties = resource.get('properties')

        sites = self.bpo.resources.get_by_type("training.resourceTypes.Site")
        for site in sites:
            if site['properties']['siteId'] == properties['siteId']:
                return failure(f"Site with siteId ({properties['siteId']}) already exists")
            elif (site['properties']['deviceName'] == properties['deviceName']) and
(site['properties']['portName'] == properties['portName']):
                return failure(f"Site with the same portName and deviceName combination already exists")

        port_validation = PortValidateActivate.run(self)
        if port_validation['state'] == "failed":
            return port_validation

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

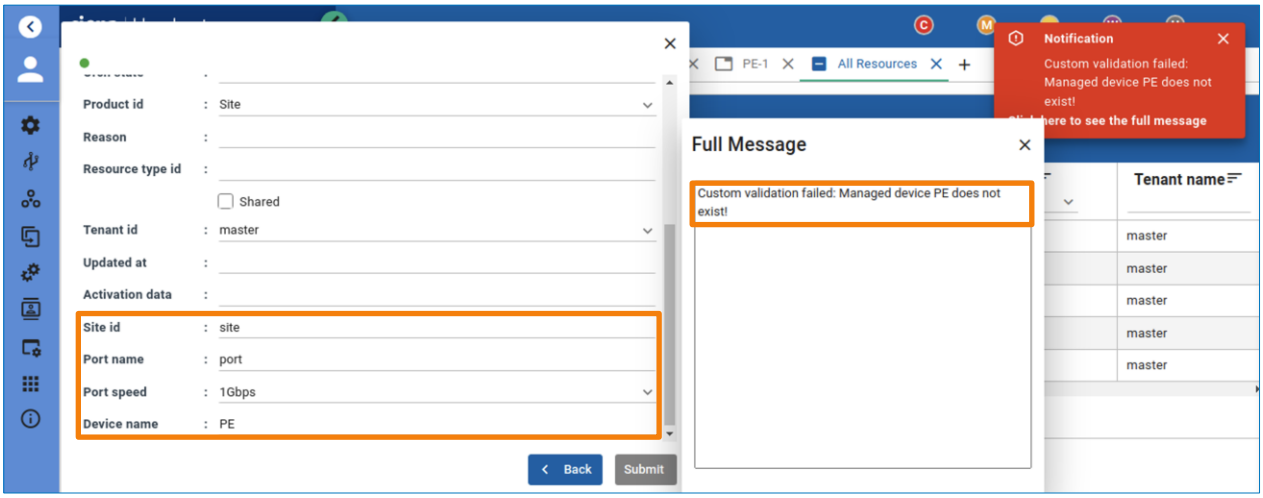
c. Update the class comments as well.

```
class ValidateActivate(Plan):
    """
    Verify that another Site does not exists with the same siteId or using the same Port
    Verify that another Site does not exists with the same portName and deviceName combination
    Verify that the managed device and port resources exist
    """
    def run(self):
...
```

16. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
17. Open the BPO UI and navigate to the **Network > All Resources** view again. choose **Planet Orchestrate** as the Domain name. Delete any leftover resources from the previous task that might exist. Do not delete the resources that have the Resource type ID starting with **ipam**.



18. Create a new **Site** resource. To test out this new validation, use some parameters that do not match the ones that actually exist.



19. Now you need to create a logical port resource.

    a. Open the **l2vpn.py** file and import the **PortUtils** from **port.py**, and **ExactTypeId** and **QQuery** from **plansdk.apis.bpo**.

```
from plansdk.apis.plan import Plan
from plansdk.apis.bpo import ExactTypeId, QQuery
from .port import PortUtils
from .util import failure, success
import logging
...
```

    b. Add a **create_managed_logical_port()** method to the **Activate class**. Read the port resource and use the recently created methods in the **PortUtils** class to read both managed device and port resources.

```
...
    def create_managed_logical_port(self, resource, allocated_vlan):

        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
            QQuery("label", f"{properties['siteLabel']}.Port")
        )

        # Read the L2VPN Port Managed Device and port resource
        device_name = port_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
        managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])
```

    c. Read the logical port product ID, which is the same as the product ID for the managed port.

```
...
    def create_managed_logical_port(self, resource, allocated_vlan):

        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
            QQuery("label", f"{properties['siteLabel']}.Port")
```

```
        )

        # Read the L2VPN Port Managed Device and port resource
        device_name = port_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
        managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])

        # Create managed logical port resource - <port>.<vlan>
        logical_port_product_id = managed_port_resource['productId']
```

d. Create the managed logical port resource. Use the managed port product ID as the **productId**, set the **label** to <interface.port> format and create a **properties** object. The properties object should contain the **name**, which should be the same as the label, the **device** should contain the managed device resource ID, and the **encapsulation** object should contain a **dot1Q** object, which further contains a **vlan-id** parameter which is set to the allocated VLAN value.

```
...
    def create_managed_logical_port(self, resource, allocated_vlan):

        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
            QQuery("label", f"{properties['siteLabel']}.Port")
        )

        # Read the L2VPN Port Managed Device and port resource
        device_name = port_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
        managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])

        # Create managed logical port resource - <port>.<vlan>
        logical_port_product_id = managed_port_resource['productId']
        managed_logical_port_resource = self.bpo.resources.create(
            managed_port_resource['id'], {
            'productId': logical_port_product_id,
            'label': f"{managed_port_resource['label']}.{allocated_vlan}",
            'properties': {
                "name":  f"{managed_port_resource['label']}.{allocated_vlan}",
                "device": managed_device_resource['id'],
                "encapsulation": {
                    "dot1Q": {
                        "vlan-id": allocated_vlan
                    }
                }
            }
        })
```

e. Create a relationship between this new managed logical port resource and the port resource at the end.

```
    def create_managed_logical_port(self, resource, allocated_vlan):

        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
            QQuery("label", f"{properties['siteLabel']}.Port")
        )

        # Read the L2VPN Port Managed Device and port resource
```

```
        device_name = port_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
        managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])

        # Create managed logical port resource - <port>.<vlan>
        logical_port_product_id = managed_port_resource['productId']
        managed_logical_port_resource = self.bpo.resources.create(
            managed_port_resource['id'], {
            'productId': logical_port_product_id,
            'label': f"{managed_port_resource['label']}.{allocated_vlan}",
            'properties': {
                "name":  f"{managed_port_resource['label']}.{allocated_vlan}",
                "device": managed_device_resource['id'],
                "encapsulation": {
                    "dot1Q": {
                        "vlan-id": allocated_vlan
                    }
                }
            }
        })
        self.bpo.relationships.add_relationship(port_resource['id'],
managed_logical_port_resource[0]['id'])
```

20. In addition to creating a new managed logical port resource, you also need to create a method to delete it.

    a. Add a **delete_managed_logical_port_resource()** to the **Terminate** class.

```
...
    def delete_managed_logical_port_resource(self, resource_id):
```

    b. Within a try-except loop try to read the Site resource and use that to read the port, managed port, and the managed logical port resources.

```
    def delete_managed_logical_port_resource(self, resource_id):
        try:
            site_resource = self.bpo.resources.get_dependency_with_filters(resource_id,
ExactTypeId("training.resourceTypes.Site"))
            port_resource = self.bpo.resources.get_dependency_with_filters(site_resource['id'],
ExactTypeId("training.resourceTypes.Port"))
            managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])
            managed_logical_port_resources =
self.bpo.resources.get_dependencies_by_type(port_resource['id'], managed_port_resource['resourceTypeId'])
            managed_logical_port_resource = [port for port in managed_logical_port_resources if "." in
port['label']][0]
        except Exception as e:
            raise Exception(f"Could not delete managed Logical Port: {e}")
```

    c. Delete all the relationships for the managed logical port resource and delete the resource as well. Use the **bpo.resources.await_termination** after you delete the resource to make sure that the deletion request is complete.

```
    def delete_managed_logical_port_resource(self, resource_id):
        try:
            site_resource = self.bpo.resources.get_dependency_with_filters(resource_id,
ExactTypeId("training.resourceTypes.Site"))
            port_resource = self.bpo.resources.get_dependency_with_filters(site_resource['id'],
ExactTypeId("training.resourceTypes.Port"))
            managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])
            managed_logical_port_resources =
self.bpo.resources.get_dependencies_by_type(port_resource['id'], managed_port_resource['resourceTypeId'])
```

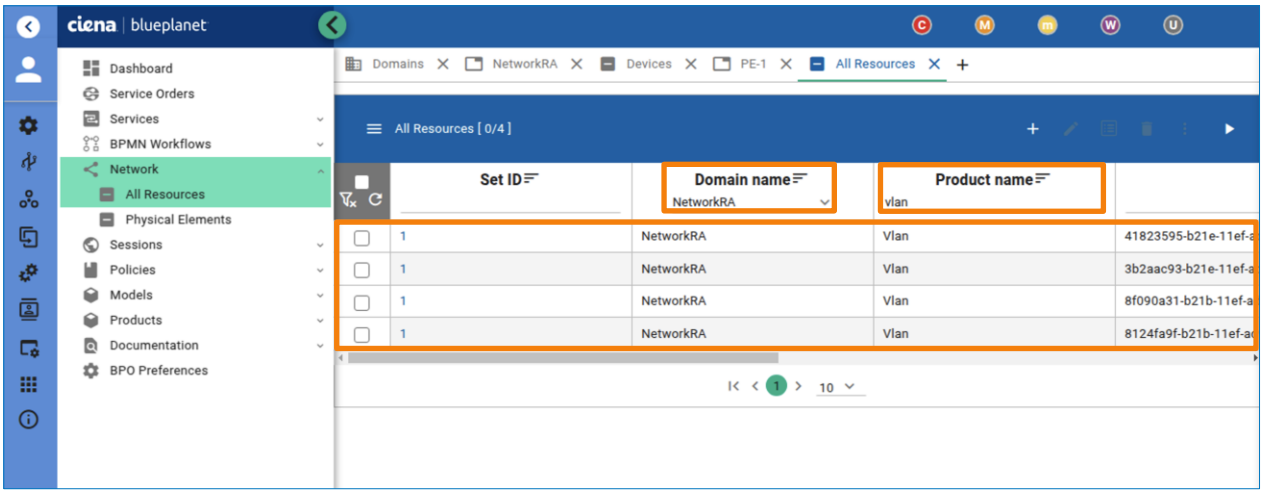
```
            managed_logical_port_resource = [port for port in managed_logical_port_resources if "." in
port['label']][0]
            self.bpo.relationships.delete_source_relationships(managed_logical_port_resource['id'])
            self.bpo.relationships.delete_relationships(managed_logical_port_resource['id'])
            self.bpo.resources.delete(managed_logical_port_resource['id'])
            self.bpo.resources.await_termination(managed_logical_port_resource['id'],
managed_logical_port_resource['label'], True)
            log.info(f"Managed Logical Port {managed_logical_port_resource['label']} deleted.")
        except Exception as e:
            raise Exception(f"Could not delete managed Logical Port: {e}")
```

21. Call the **delete_managed_logical_port_resource()** from the **Terminate run()** method.

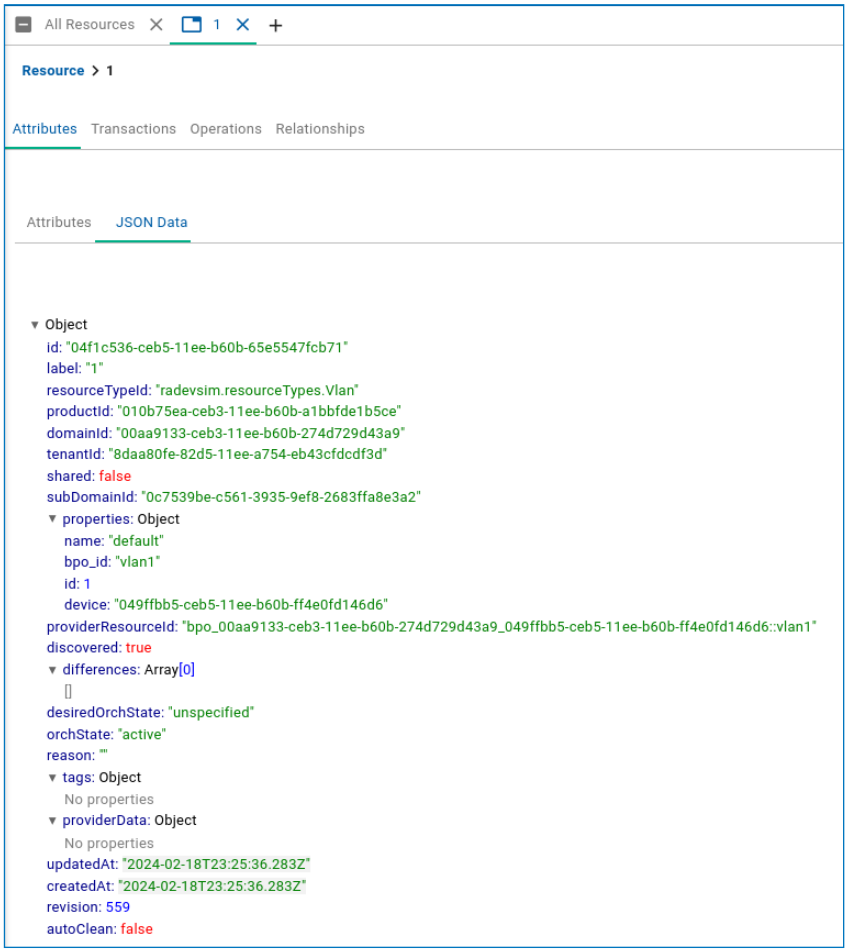
```
class Terminate(Plan):
    """
    Delete L2VPN resource from market
    Delete Customer, Site, and Bandwidthprofile relationships
    Delete managed logical port resource
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        label = resource['label']
        try:
            vlan_num_res = self.bpo.resources.get_one_by_filters(
                    resource_type=POOL_NUMBER_RESOURCE_TYPE,
                    q_params={"label": f"{label}.VLAN"}
            )
            self.bpo.resources.delete(vlan_num_res['id'])
            self.bpo.resources.await_termination(vlan_num_res['id'], f"{label}.VLAN", False)
            log.info(f"VLAN for {label} released")
        except:
            log.info(f"VLAN for {label} not found")

        self.delete_managed_logical_port_resource(resource_id)
        log.info("Deleting managed Logical Port resource")

        self.delete_l2vpn_relationships(resource_id)
        log.info("Deleting Customer, Site and BandwidthProfile relationships")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

22. Up until now, you have taken care of the logical port resource and the relationship between the Port and managed port resources. Now, you make sure that a managed VLAN resource is created on the device which is attached to these port resources. This must be taken care of in multiple actions:
    - Create a managed VLAN resource when a L2VPN resource is created.
    - Delete a managed VLAN resource when a L2VPN resource is deleted.
    - Update a managed VLAN resource when a L2VPN resource is updated.
    - Update a managed VLAN resource when the ChangePort operation is called.

    a. Inspect an already existing managed VLAN resource on the BPO UI. You will see that each managed device already has 1 vlan (label/Set ID).



b. Inspect one of them to understand the managed VLAN resource structure.



c. You can also see that the **VLAN with number 1** is reserved as the default VLAN. This means that it cannot be used for any L2VPN VLAN allocation. Fix this by updating the parameters used for VLAN pool creation in the **create_vlan_pool** method in the **Activate** class in the **l2vpn.py** file.

```
    def create_vlan_pool(self):
        # Try to find the "tosca.resourceTypes.NumberPool" resource
```

```
        # If none are found, create it
        try:
            pool = self.bpo.resources.get_one_by_filters(
                resource_type = POOL_RESOURCE_TYPE,
                q_params = { "label": POOL_NAME }
            )
            log.info(f"Found VLAN pool {POOL_NAME}")
        except:
            log.info(f"VLAN pool {POOL_NAME} doesn't exist. Creating...")
            vlan_pool_id = self.bpo.market.get_products_by_resource_type(POOL_RESOURCE_TYPE)[0]['id']

            # Training VLAN pool created with no parent - it is global
            pool = self.bpo.resources.create(None, {
                'productId': vlan_pool_id,
                'label': POOL_NAME,
                'properties': { "lowest":  2,
                                "highest": 4095}
            })
        return pool
```

d. This range must also be enforced on the resource type level. Open the **resource_type_l2vpn.tosca** file. Add **minimum** and a **maximum** definitions to the **vlanId** property there.

```
...
resourceTypes {
  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {

      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
        updatable = true
        minimum = 2
        maximum = 4095
      }
...
```

e. Return to the **l2vpn.**py file. In the **Activate** class, create a new **create_managed_vlan_resource()** method. Set **resource** and **allocated_vlan** as input parameters.

```
class Activate(Plan):
    """
    Create L2VPN resource into market
    Create relationship with Customer, Site, and Bandwidth Profile resource
    """
...

    def create_managed_vlan_resource(self, resource, allocated_vlan):
```

f. In the **create_managed_vlan_resource()** method, first read the Port resource using the **ExactTypeId** and **QQuery** classes. Study the **Query Options** chapter of the PlanSDK documentation to understand how they are used.

```
    def create_managed_vlan_resource(self, resource, allocated_vlan):
        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
```

```
            QQuery("label", f"{properties['siteLabel']}.Port")
        )
```

g. Based on the device used in this port resource, read the managed device resource using the **get_managed_device_resource()** method from the **PortUtils** class.

```
    def create_managed_vlan_resource(self, resource, allocated_vlan):
        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
            QQuery("label", f"{properties['siteLabel']}.Port")
        )

        # Read the L2VPN Port Managed Device resource
        device_name = port_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
```

h. You must now create the managed VLAN resource. Read the managed VLAN product ID with **bpo.market.get_product_by_resource_type()** method.

```
    def create_managed_vlan_resource(self, resource, allocated_vlan):
        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
            QQuery("label", f"{properties['siteLabel']}.Port")
        )

        # Read the L2VPN Port Managed Device resource
        device_name = port_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)

        # Create a Managed VLAN resource
        managed_vlan_product =
self.bpo.market.get_products_by_resource_type('radevsim.resourceTypes.Vlan')[0]['id']
```

i. In a try-except loop, create a managed VLAN resource using the **bpo.resources.create()** method. Use the following object structure as the **data** parameter:

- **productId** (managed VLAN product ID)
- **label** (managed VLAN label - <device>-<l2vpn resource>.VLAN)
- **properties**
  - **id** (allocated VLAN number)
  - **name** (managed VLAN label - <device>-<l2vpn resource>.VLAN)
  - **device** (managed device resource ID)

```
    def create_managed_vlan_resource(self, resource, allocated_vlan):
        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
            QQuery("label", f"{properties['siteLabel']}.Port")
        )

        # Read the L2VPN Port Managed Device resource
        device_name = port_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)

        # Create a Managed VLAN resource
        managed_vlan_product =
self.bpo.market.get_products_by_resource_type('radevsim.resourceTypes.Vlan')[0]['id']
        try:
```

```
            managed_vlan = self.bpo.resources.create(managed_device_resource["id"], {
                'productId': managed_vlan_product,
                'label': f"{device_name}.{resource['label']}.VLAN",
                'properties': {
                    "id":  allocated_vlan,
                    "name": f"{device_name}.{resource['label']}.VLAN",
                    "device": managed_device_resource["id"]
                }
            })
        except Exception as e:
            raise Exception(f"Failed to create VLAN with ID {allocated_vlan}. Make sure that it is
available on device: {e}")
```

j. Additionally, add a relationship between the L2VPN resource and the newly created managed VLAN resource as well.

```
    def create_managed_vlan_resource(self, resource, allocated_vlan):
        properties = resource['properties']

        # Read the L2VPN Port resource
        port_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Port"),
            QQuery("label", f"{properties['siteLabel']}.Port")
        )

        # Read the L2VPN Port Managed Device resource
        device_name = port_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)

        # Create a Managed VLAN resource
        managed_vlan_product =
self.bpo.market.get_products_by_resource_type('radevsim.resourceTypes.Vlan')[0]['id']
        try:
            managed_vlan = self.bpo.resources.create(managed_device_resource["id"], {
                'productId': managed_vlan_product,
                'label': f"{device_name}.{resource['label']}.VLAN",
                'properties': {
                    "id":  allocated_vlan,
                    "name": f"{device_name}.{resource['label']}.VLAN",
                    "device": managed_device_resource["id"]
                }
            })
        except Exception as e:
            raise Exception(f"Failed to create VLAN with ID {allocated_vlan}. Make sure that it is
available on device: {e}")
        self.bpo.relationships.add_relationship(resource['id'], managed_vlan[0]['id'])
```

k. To call this method from the **Activate run()** method, you need the number of the allocated VLAN as a parameter. Update the **allocate_vlan()** method so that it returns this number.

```
    def allocate_vlan(self, resource):
        pool_number_product_id = None

        # First, read the ID from product based on "tosca.resourceTypes.PooledNumber"
        try:
            pool_number_product_id = self.bpo.products.get_by_domain_and_type("built-in",
POOL_NUMBER_RESOURCE_TYPE)[0]['id']
            log.info(f"Found VLAN pool number product with ID {pool_number_product_id}")
        except:
            raise Exception(f"VLAN pool number product with ID {pool_number_product_id} doesn't exist")

        vlan_id = resource['properties'].get('vlanId')
        label = resource['label']

        # If vlanId has been entered use that when creating the VLAN resource, else use the allocated
number from PooledNumber
        pool_number_product_id = self.bpo.products.get_by_domain_and_type("built-in",
POOL_NUMBER_RESOURCE_TYPE)[0]['id']
```

```
        if vlan_id:
            allocated_number = self.bpo.resources.create(resource['id'], {
                'productId': pool_number_product_id,
                'label': f"{label}.VLAN",
                'properties': { "requestedValue": vlan_id }
            })
        else:
            allocated_number = self.bpo.resources.create(resource['id'], {
                'productId': pool_number_product_id,
                'label': f"{label}.VLAN"
            })
        log.info(f"Successfully allocated VLAN pool number -
{allocated_number[0]['properties']['allocatedValue']}")
        return allocated_number[0]['properties']['allocatedValue']
```

l. Call this **create_managed_vlan_resource** trom the **Activate run()** method. Call the **create_managed_logical_port_resource** method as well, now that you have the allocated VLAN number.

```
class Activate(Plan):
    """
    Create L2VPN resource into market
    Create relationship with Customer, Site, and Bandwidth Profile resource
    Allocate VLAN
    Create a managed VLAN resource
    Create a managed logical port resource
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        properties = resource['properties']
        self.assign_customer(properties)
        log.info(f"Assign Customer resource: {properties['customerLabel']}")

        self.assign_site(properties)
        log.info(f"Assign Site resource: {properties['siteLabel']}")

        self.assign_bandwidthprofile(properties)
        log.info(f"Assign Bandwidth Profile resource: {properties['bandwidthProfile']}")

        self.create_vlan_pool()
        allocated_vlan = self.allocate_vlan(resource)

        self.create_managed_vlan_resource(resource, allocated_vlan)
        self.create_managed_logical_port(resource, allocated_vlan)

        log.info("Activate: DONE")
        return {}
```

23. This takes care of managed VLAN creation. Now take care of deletion as well.
    a. Add a new **delete_managed_vlan()** method to the **Terminate** class and use the **resource_id** (ID of the L2VPN resource) as the input parameter.

```
    def delete_managed_vlan_resource(self, resource_id):
```

    b. Implement the method. In a try-except loop first get the managed VLAN resource as a L2VPN dependency. You can use the **bpo.resources.get_dependencies_with_filters()** method.

```
    def delete_managed_vlan_resource(self, resource_id):
        try:
            managed_vlan = self.bpo.resources.get_dependencies_with_filters(
```

```
                resource_id,
                ExactTypeId("radevsim.resourceTypes.Vlan")
            )[0]
        except Exception as e:
            raise Exception(f"Could not delete managed VLAN: {e}")
```

c. Then, delete both source and target relationships for the managed VLAN resource.

```
    def delete_managed_vlan_resource(self, resource_id):
        try:
            managed_vlan = self.bpo.resources.get_dependencies_with_filters(
                resource_id,
                ExactTypeId("radevsim.resourceTypes.Vlan")
            )[0]
            self.bpo.relationships.delete_source_relationships(managed_vlan['id'])
            self.bpo.relationships.delete_relationships(managed_vlan['id'])
        except Exception as e:
            raise Exception(f"Could not delete managed VLAN: {e}")
```

d. And finally, delete the managed VLAN resource and await its termination.

```
    def delete_managed_vlan_resource(self, resource_id):
        try:
            managed_vlan = self.bpo.resources.get_dependencies_with_filters(
                resource_id,
                ExactTypeId("radevsim.resourceTypes.Vlan")
            )[0]
            self.bpo.relationships.delete_source_relationships(
                managed_vlan['id'])
            self.bpo.relationships.delete_relationships(
                managed_vlan['id'])
            self.bpo.resources.delete(managed_vlan['id'])
            self.bpo.resources.await_termination(managed_vlan['id'], managed_vlan['label'], True)
            log.info(f"Managed VLAN {managed_vlan['label']} deleted.")
        except Exception as e:
            raise Exception(f"Could not delete managed VLAN: {e}")
```

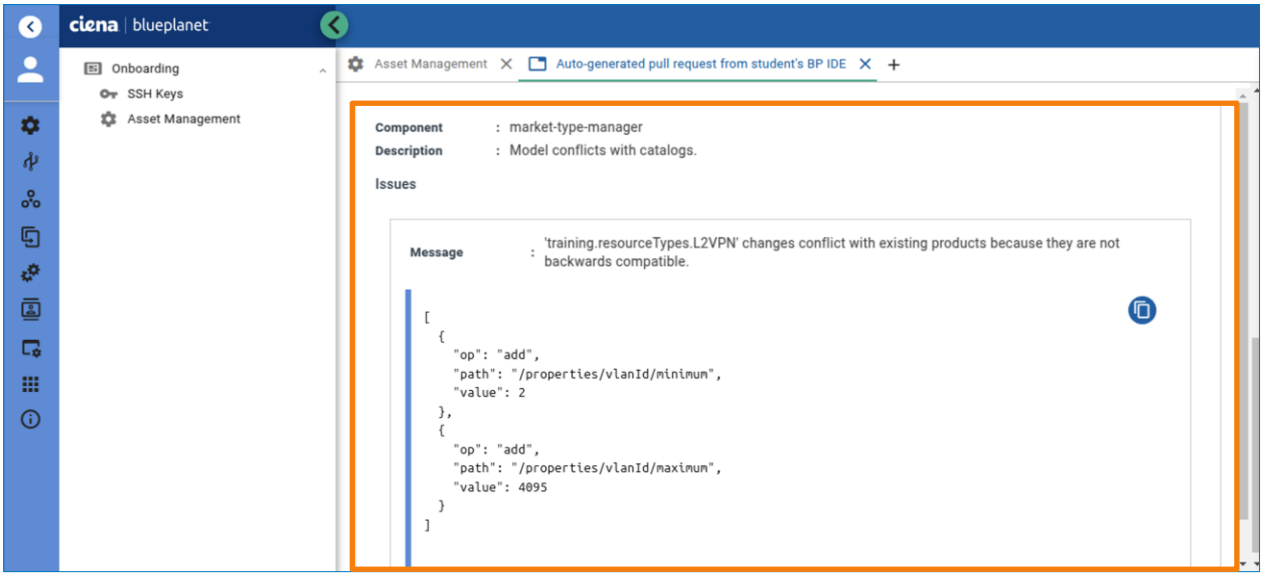
e. Call this method from **Terminate run()** method.

```
class Terminate(Plan):
    """
    Delete L2VPN resource from market
    Delete Customer, Site, and Bandwidthprofile relationships
    Delete managed VLAN resource
    Delete managed logical port resource
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        label = resource['label']
        try:
            vlan_num_res = self.bpo.resources.get_one_by_filters(
                    resource_type=POOL_NUMBER_RESOURCE_TYPE,
                    q_params={"label": f"{label}.VLAN"}
            )
            self.bpo.resources.delete(vlan_num_res['id'])
            self.bpo.resources.await_termination(vlan_num_res['id'], f"{label}.VLAN", False)
            log.info(f"VLAN for {label} released")
        except:
            log.info(f"VLAN for {label} not found")

        self.delete_managed_vlan_resource(resource_id)
        log.info("Deleting managed VLAN resource")
```

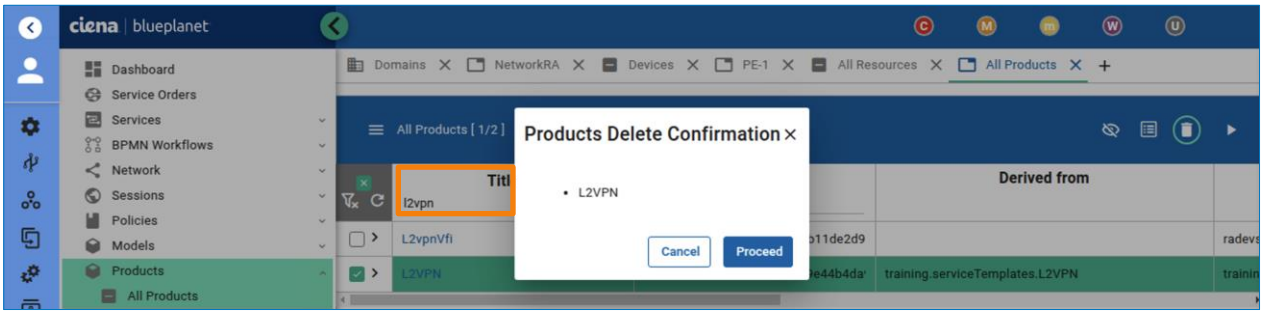
```
        self.delete_managed_logical_port_resource(resource_id)
        log.info("Deleting managed Logical Port resource")

        self.delete_l2vpn_relationships(resource_id)
        log.info("Deleting Customer, Site and BandwidthProfile relationships")

        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

24. **Onboard** the changes to the BPO server. This time, a following conflict should occur due to backwards incompatible changes. You will learn how to handle such changes in one of the upcoming labs.

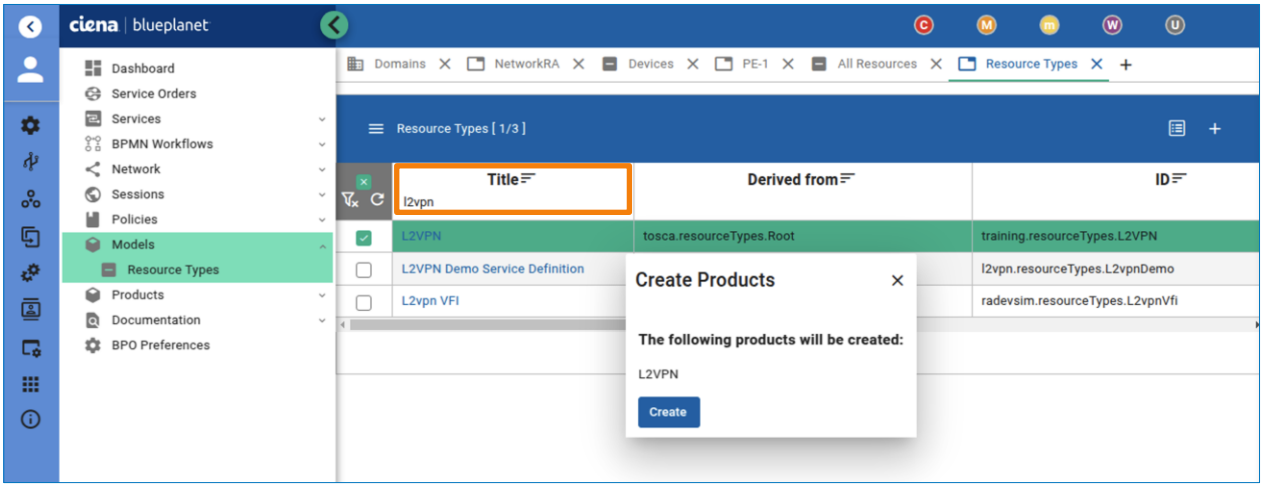
You can see more details in the rejected pull request in the Asset Manager.



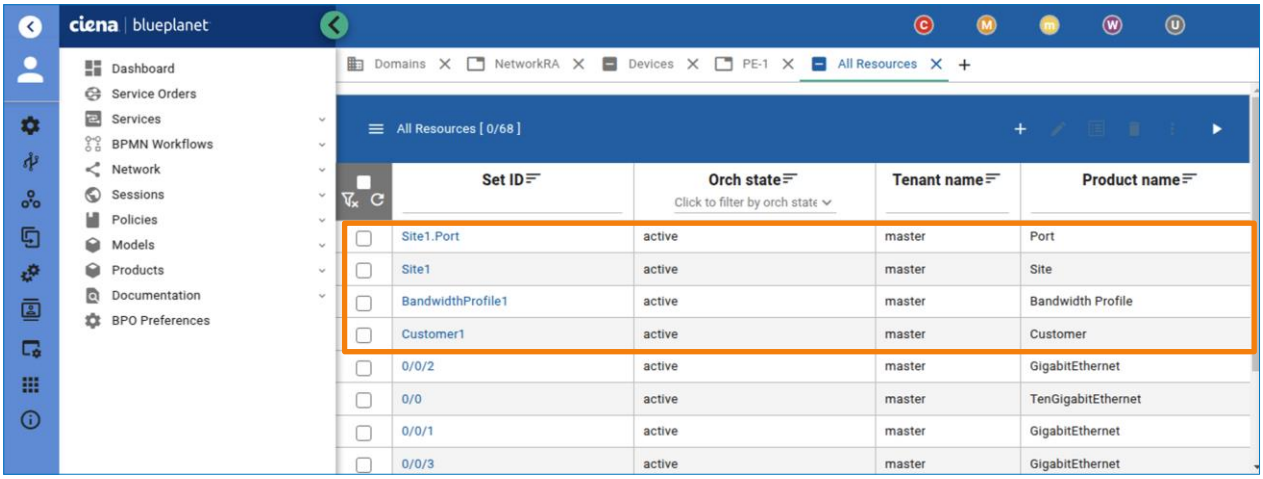
25. For now, simply delete the L2VPN product by going under **Products > All Products,** choosing the L2VPN Product and deleting it.



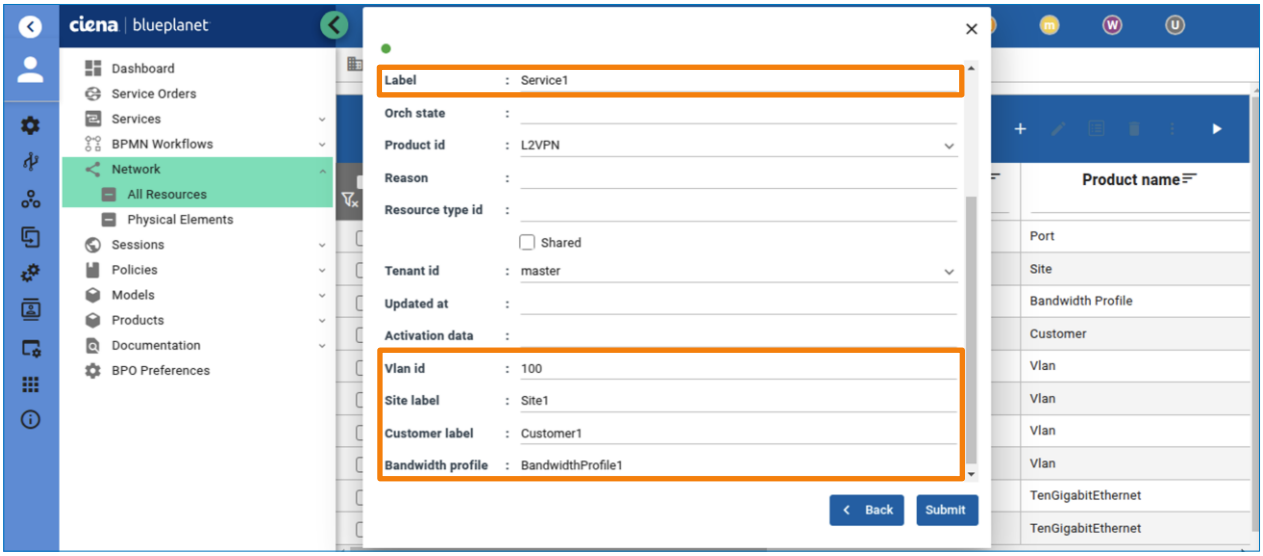
26. **Onboard** the changes to the BPO server again. This time, the changes should be onboarded with no issues. If there are any, troubleshoot and fix them.
27. Re-create the L2VPN product by going under **Models > Resource Types**, choosing the L2VPN resource type, and clicking the **+** symbol in the upper right corner of the screen to create the product.



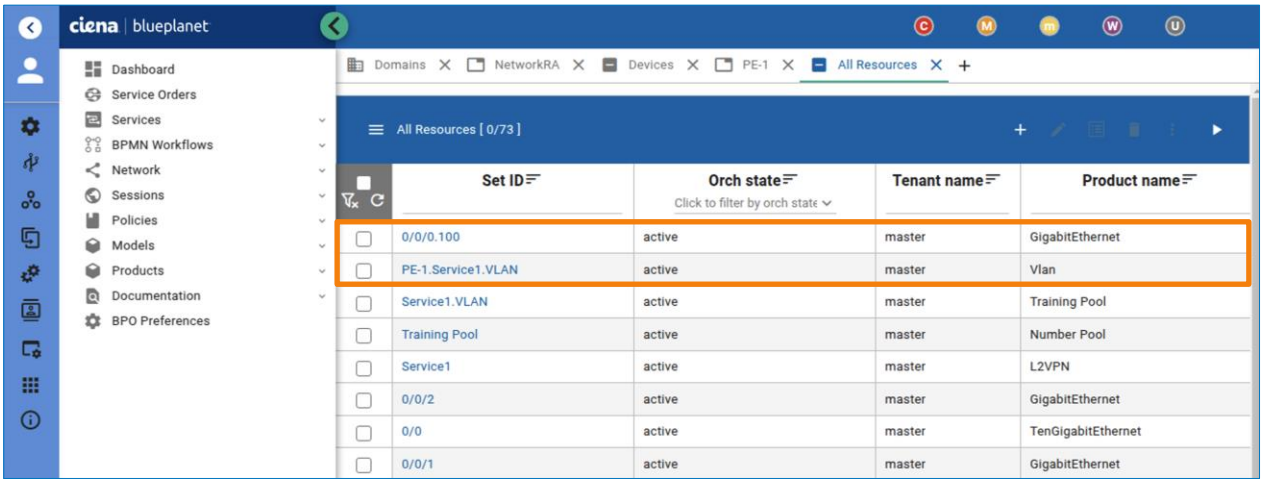
28. Open the Resources tab under **Network > All Resources** tab. Create new Site (Port name : 0/0/0, Port speed: 1Gbps, Device name : PE-1), Customer and Bandwidth Profile resources. Make sure you specify a port and device that actually exist as managed resources when creating the site and that all resources end up in an active state.



29. Create a new L2VPN resource based on these sub-resources.



You see that a new managed VLAN and managed logical port resources have been created in addition to the service VLAN resource. If you do not see it, make sure to clear the Domain Name filter.



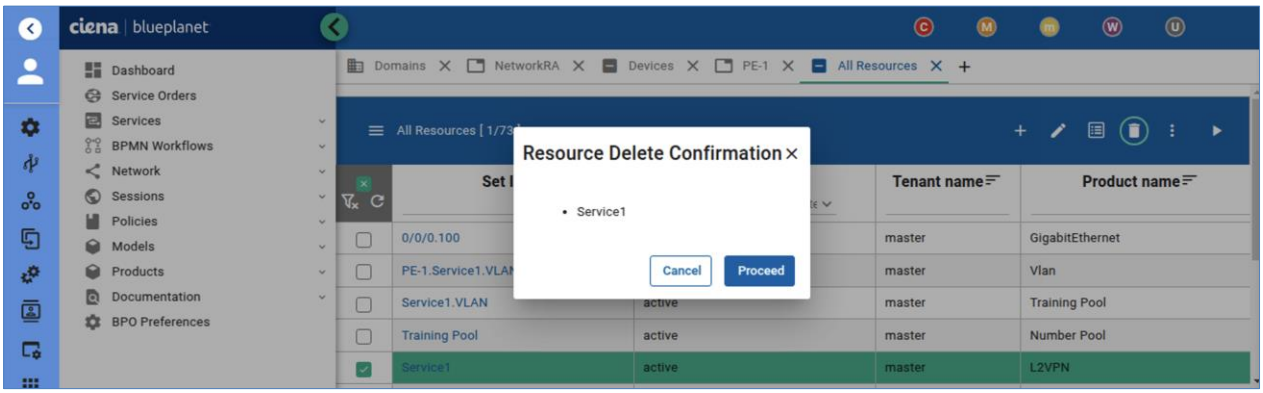
30. These managed resources are created on the devices as well. Open the terminal and connect to the PE-1 device (or whichever device you used in the site configuration). Observe the VLAN 100 if you have entered the same **vlanId** parameter, or whichever VLAN you set otherwise.

```
student@POD-XX:~$ ssh admin@PE-1
admin@pe-1's password: admin
admin@PE-1 [/]> show configuration vlan
dev-sim:vlan {
   vlan-list 1 {
      name default;
   }
   vlan-list 100 {
      name PE-1.Service1.VLAN;
   }
}
```

31. Display the interfaces configuration as well. The managed logical port should be created and have the encapsulation and vlan-id set.

```
admin@PE-1 [/]> show configuration interface GigabitEthernet
GigabitEthernet 0/0/0 {
}
GigabitEthernet 0/0/0.100 {
   encapsulation {
      dot1Q {
         vlan-id 100;
      }
   }
}

GigabitEthernet 0/0/1 {
}
GigabitEthernet 0/0/2 {
}
GigabitEthernet 0/0/3 {
}
```

32. Open the BPO UI again. Delete **only** the L2VPN resource. The managed VLAN and logical port should get deleted, along with VLAN resource.



33. Connect to the PE-1 device again and verify that the VLAN has disappeared from the device configuration. Only the default VLAN should be present.

```
admin@PE-1 [/]> show configuration vlan
dev-sim:vlan {
   vlan-list 1 {
      name default;
   }
}
admin@PE-1 [/]>
```

34. Verify that the logical port has been deleted on the device as well.

```
admin@PE-1 [/]> show configuration interface GigabitEthernet
GigabitEthernet 0/0/0 {
}
GigabitEthernet 0/0/1 {
}
GigabitEthernet 0/0/2 {
}
GigabitEthernet 0/0/3 {
}
```

35. Take care of the update lifecycle operation for the L2VPN resource now. A user has the ability to update the VLAN ID for a l2vpn resource, which means that the managed VLAN resource needs to be deleted and recreated. Luckily, you have already created the code that both deletes and creates this managed resource.

a. Open the **l2vpn.py** file again. In the **Update run()** method, add the **Terminate delete_managed_vlan_resource()** method. Make sure you add it before you delete the relationships, since the deletion method relies on those to find the managed resource.

```
class Update(Plan):
    """
    Updates the VLAN parameter
    """
    def run(self):
        log.info(f"Update: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Update: resourceId {resource_id}")
        log.info(f"Update: {resource}")

        label = resource['label']
        vlan_num_res = self.bpo.resources.get_one_by_filters(
                resource_type=POOL_NUMBER_RESOURCE_TYPE,
                q_params={"label": f"{label}.VLAN"}
        )

        Terminate.delete_managed_vlan_resource(self, resource_id)
        self.bpo.relationships.delete_relationships(vlan_num_res['id'])
        self.bpo.resources.delete(vlan_num_res['id'])
        log.info(f"VLAN for {label} released")

        Activate.allocate_vlan(self, resource)

        log.info("Update: DONE")
        return {}
```

b. Store the results of the **allocate_vlan()** method to a variable and use it to call the **Activate create_managed_vlan_resource()** method. This is how the **Update** class should look after this.

```
class Update(Plan):
    """
    Updates the VLAN parameter
    Updates the managed VLAN resource
    """
    def run(self):
        log.info(f"Update: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Update: resourceId {resource_id}")
        log.info(f"Update: {resource}")

        label = resource['label']
        vlan_num_res = self.bpo.resources.get_one_by_filters(
                resource_type=POOL_NUMBER_RESOURCE_TYPE,
                q_params={"label": f"{label}.VLAN"}
        )

        Terminate.delete_managed_vlan_resource(self, resource_id)
        self.bpo.relationships.delete_relationships(vlan_num_res['id'])
        self.bpo.resources.delete(vlan_num_res['id'])
        log.info(f"VLAN for {label} released")

        allocated_vlan = Activate.allocate_vlan(self, resource)
        Activate.create_managed_vlan_resource(self, resource, allocated_vlan)

        log.info("Update: DONE")
        return {}
```

c. Since the VLAN has changes, do must the encapsulation on the managed logical port resource. Use the **create_managed_logical_port_resource** and **delete_managed_logical_port_resource** to do so.

```
class Update(Plan):
    """
    Updates the VLAN parameter
    Updates the managed VLAN resource
    Updates the managed logical port resource
    """
    def run(self):
        log.info(f"Update: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Update: resourceId {resource_id}")
        log.info(f"Update: {resource}")

        label = resource['label']
        vlan_num_res = self.bpo.resources.get_one_by_filters(
                resource_type=POOL_NUMBER_RESOURCE_TYPE,
                q_params={"label": f"{label}.VLAN"}
        )

        Terminate.delete_managed_vlan_resource(self, resource_id)
        self.bpo.relationships.delete_relationships(vlan_num_res['id'])
        self.bpo.resources.delete(vlan_num_res['id'])
        log.info(f"VLAN for {label} released")

        allocated_vlan = Activate.allocate_vlan(self, resource)
        Activate.create_managed_vlan_resource(self, resource, allocated_vlan)

        Terminate.delete_managed_logical_port_resource(self, resource_id)
        Activate.create_managed_logical_port(self, resource, allocated_vlan)

        log.info("Update: DONE")
        return {}
```

36. The last thing you need to take care of when it comes to the managed VLAN resource, is the ChangePort custom operation. When a user changes the port, they have the ability to change the device on which the port is active as well, meaning that the managed VLAN resource will need to be deleted on the old device, and created on the new device. Same goes for the relationship between the managed and non-managed port resource.

    a. In the **run()** method from the **ChangePort** class, after patching the non-managed port resource, read the old managed port resource using the **PortUtils get_managed_port_resource** method. Delete the relationship between this old managed port resource and the current non-managed port resource.

```
class ChangePort(Plan):
    """
    Changes the Port and Device attached to selected Site in L2VPN resource
    Updates the managed VLAN resource
    """
    def run(self):
        log.info(f"ChangePort: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"ChangePort: resourceId {resource_id}")
        log.info(f"ChangePort: {resource}")

        operation = self.bpo.resources.get_operation(resource_id, self.params['operationId'])
        inputs = operation['inputs']
```

```
        log.info(f"ChangePort: inputs {inputs}")
        site_label = inputs['siteLabel']

        # Verify that the Site exists
        try:
            site_res = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": f"{site_label}"}
            )
            log.info(f"Site resource for {site_label} found.")
        except:
            raise Exception(f"Site resource for {site_label} not found.")

        # Find the Port resource attached to this Site
        try:
            port_res = self.bpo.resources.get_dependency_by_type(site_res['id'],
"training.resourceTypes.Port")
            log.info(f"Port resource for {site_label} found - {port_res}")
        except:
            raise Exception(f"Port resource for {site_label} not found!")

        # Patch the Port resource
        self.bpo.resources.patch(port_res['id'], {
            "properties": {
                "portName": inputs['portName'],
                "deviceName": inputs['deviceName'],
                "portSpeed": inputs['portSpeed']
            }
        })
        log.info(f"Port resource for {site_label} patched.")

        # Patch the Site resource
        self.bpo.resources.patch(site_res['id'], {
            "properties": {
                "portName": inputs['portName'],
                "deviceName": inputs['deviceName'],
                "portSpeed": inputs['portSpeed']
            }
        })
        log.info(f"Site resource for {site_label} patched.")

        # Update relationship between port and managed port resources
        old_managed_port_resource = PortUtils.get_managed_port_resource(self,
port_res['properties']['portName'], port_res['properties']['deviceName'],
port_res['properties']['portSpeed'])
        self.bpo.relationships.delete_by_source_and_target(port_res['id'], old_managed_port_resource['id'])

        log.info("ChangePort: DONE")
        return {}
```

b. Read the new managed port resource and create a relationship between the non-managed port resource and the new managed port resource.

```
...
        # Update relationship between port and managed port resources
        old_managed_port_resource = PortUtils.get_managed_port_resource(self,
port_res['properties']['portName'], port_res['properties']['deviceName'],
port_res['properties']['portSpeed'])
        self.bpo.relationships.delete_by_source_and_target(port_res['id'], old_managed_port_resource['id'])
        managed_port_resource = PortUtils.get_managed_port_resource(self, inputs['portName'],
inputs['deviceName'], inputs['portSpeed'])
        self.bpo.relationships.add_relationship(port_res['id'], managed_port_resource['id'])
```

c. Update the managed VLAN resource. Similar to the L2VPN Update operation, reuse the **delete_managed_port_resource**, and **create_managed_port_resource** methods. First, read the current VLAN ID parameter from either managed or non-managed VLAN resource, then delete

and re-create the managed VLAN resource. This is how the **ChangePort** class should look when finished:

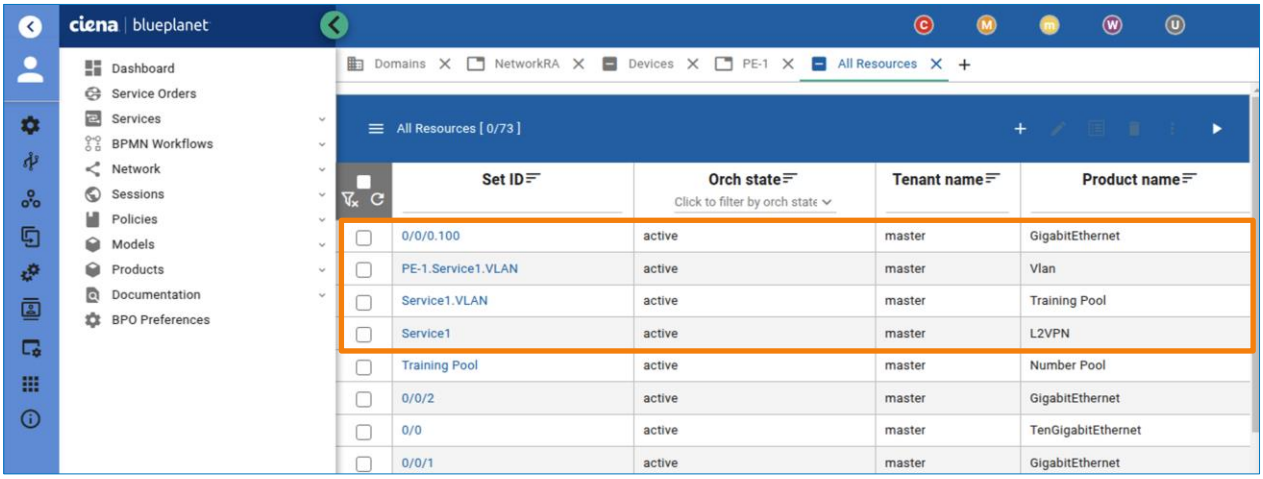
```
class ChangePort(Plan):
    """
    Changes the Port and Device attached to selected Site in L2VPN resource
    Updates the managed VLAN resource
    """
    def run(self):
        log.info(f"ChangePort: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"ChangePort: resourceId {resource_id}")
        log.info(f"ChangePort: {resource}")

        operation = self.bpo.resources.get_operation(resource_id, self.params['operationId'])
        inputs = operation['inputs']

        log.info(f"ChangePort: inputs {inputs}")
        site_label = inputs['siteLabel']

        # Verify that the Site exists
        try:
            site_res = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": f"{site_label}"}
            )
            log.info(f"Site resource for {site_label} found.")
        except:
            raise Exception(f"Site resource for {site_label} not found.")

        # Find the Port resource attached to this Site
        port_res = self.bpo.resources.get_dependency_by_type(site_res['id'], "training.resourceTypes.Port")
        log.info(f"Port resource for {site_label} found - {port_res}")

        # Patch the Port resource
        self.bpo.resources.patch(port_res['id'], {
            "properties": {
                "portName": inputs['portName'],
                "deviceName": inputs['deviceName'],
                "portSpeed": inputs['portSpeed']
            }
        })
        log.info(f"Port resource for {site_label} patched.")

        # Patch the Site resource
        self.bpo.resources.patch(site_res['id'], {
            "properties": {
                "portName": inputs['portName'],
                "deviceName": inputs['deviceName'],
                "portSpeed": inputs['portSpeed']
            }
        })
        log.info(f"Site resource for {site_label} patched.")

        # Update relationship between port and managed port resources
        old_managed_port_resource = PortUtils.get_managed_port_resource(self,
port_res['properties']['portName'], port_res['properties']['deviceName'],
port_res['properties']['portSpeed'])
        self.bpo.relationships.delete_by_source_and_target(port_res['id'], old_managed_port_resource['id'])
        managed_port_resource = PortUtils.get_managed_port_resource(self, inputs['portName'],
inputs['deviceName'], inputs['portSpeed'])
        self.bpo.relationships.add_relationship(port_res['id'], managed_port_resource['id'])
```

```
        # Update managed VLAN resource
        managed_vlan = self.bpo.resources.get_dependencies_with_filters(
            resource_id,
            ExactTypeId("radevsim.resourceTypes.Vlan")
        )[0]
        Terminate.delete_managed_vlan_resource(self, resource_id)
        Activate.create_managed_vlan_resource(self, resource, managed_vlan['properties']['id'])

        log.info("ChangePort: DONE")
        return {}
```

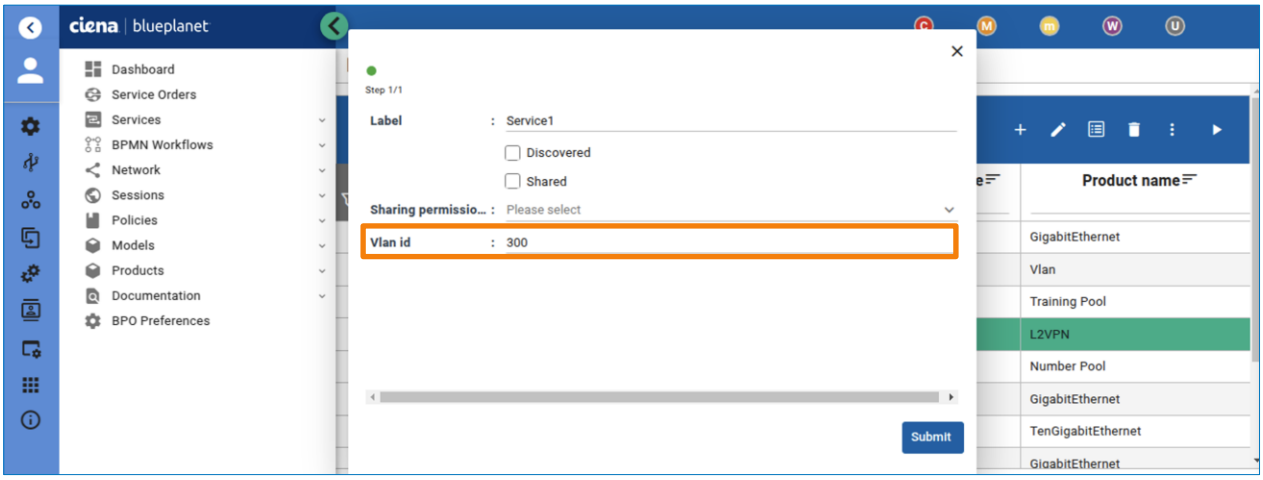
d. Update the managed logical port resource as well, using a combination of deletion and creation like in the **Update** class.

```
        # Update relationship between port and managed port resources
        old_managed_port_resource = PortUtils.get_managed_port_resource(self,
port_res['properties']['portName'], port_res['properties']['deviceName'],
port_res['properties']['portSpeed'])
        self.bpo.relationships.delete_by_source_and_target(port_res['id'], old_managed_port_resource['id'])
        managed_port_resource = PortUtils.get_managed_port_resource(self, inputs['portName'],
inputs['deviceName'], inputs['portSpeed'])
        self.bpo.relationships.add_relationship(port_res['id'], managed_port_resource['id'])

        # Update managed VLAN resource
        managed_vlan = self.bpo.resources.get_dependencies_with_filters(
            resource_id,
            ExactTypeId("radevsim.resourceTypes.Vlan")
        )[0]
        Terminate.delete_managed_vlan_resource(self, resource_id)
        Activate.create_managed_vlan_resource(self, resource, managed_vlan['properties']['id'])

        # Delete and re-create a managed logical port resource
        Terminate.delete_managed_logical_port_resource(self, resource_id)
        Activate.create_managed_logical_port(self, resource, managed_vlan['properties']['id'])

        log.info("ChangePort: DONE")
        return {}
```

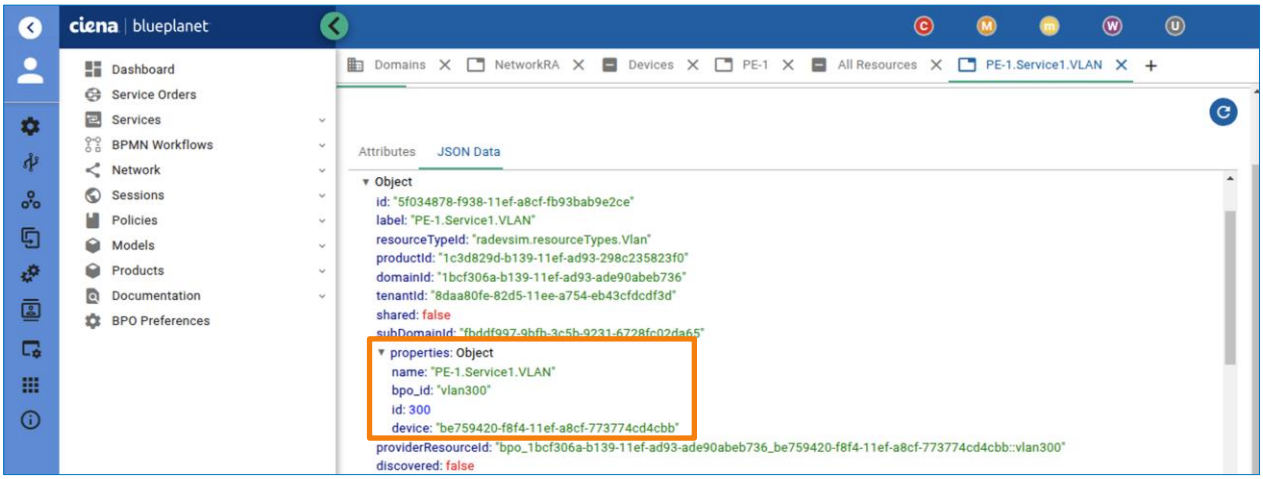
37. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
38. Test out your new changes. Open the **All Resources** tab on the BPO UI. Create a new L2VPN resource. Make sure the resource ends up in an active state.



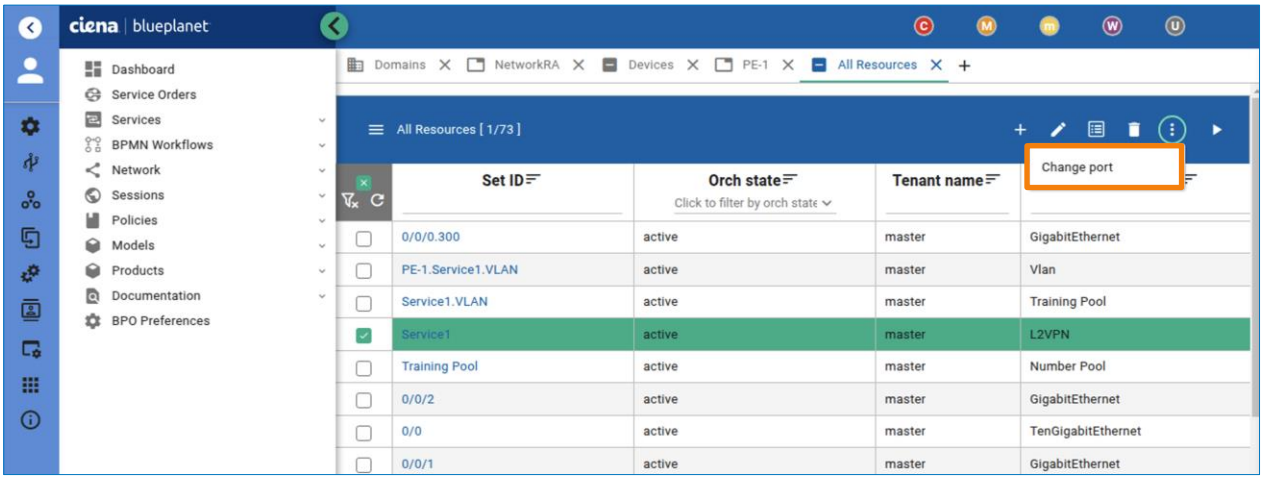
39. Choose the L2VPN resource and edit it. Set the Vlan ID to an arbitrary number, making sure it is different from the one it currently uses.



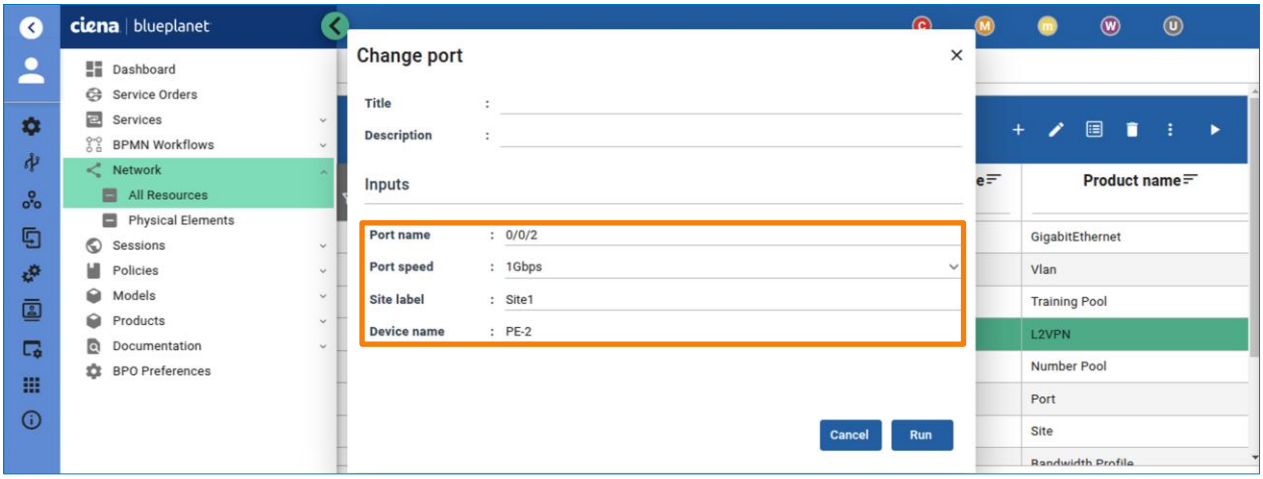
40. Inspect the updated managed VLAN resource. The newly set VLAN should be set on the managed VLAN resource as well.



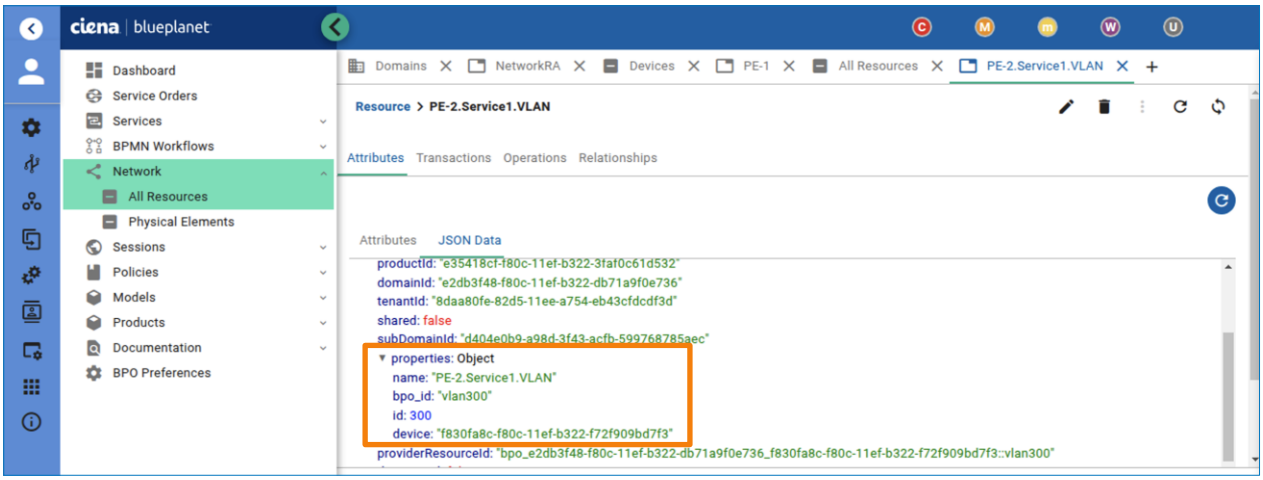
41. Try out the ChangePort operation now. In the **All Resources** tab, choose the L2VPN resource. Choose the three dots in the upper right corner and click the **ChangePort** operation.



42. Change the port to a new port and device combination. Press **Run** when done.



43. The new managed VLAN and logical port resources should reflect the ChangePort operation.



44. Verify that the VLAN has also been changed on the actual simulated device itself.

```
student@POD-XX:~$ ssh admin@PE-2
admin@pe-2's password: admin
admin@PE-2 [/]> show configuration vlan
dev-sim:vlan {
   vlan-list 1 {
      name default;
   }
   vlan-list 300 {
      name PE-2.Service1.VLAN;
   }
}
admin@PE-2 [/]> show configuration interface GigabitEthernet
GigabitEthernet 0/0/0 {
}
GigabitEthernet 0/0/1 {
}
GigabitEthernet 0/0/2 {
}
GigabitEthernet 0/0/2.300 {
   encapsulation {
      dot1Q {
         vlan-id 300;
      }
```

```
    }
}
GigabitEthernet 0/0/3 {
}
```

45. Delete the L2VPN resource.
46. Lastly, to integrate the managed domain with your L2VPN service, create (and delete) a managed Policy Map on the device based on the Bandwidth Profile resource.
    a. Open the **l2vpn.py** file and add a **create_managed_policy_map()** method to the **Activate** class.
    b. Within that method, read both the BandwidthProfile and Site resources, based on the labels passed to the L2VPN as its input parameters.

```
    def create_managed_policy_map(self, resource):
        properties = resource['properties']

        # Read the L2VPN BandwidthProfile resource
        bwp_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.BandwidthProfile"),
            QQuery("label", properties['bandwidthProfile'])
        )

        # Read the L2VPN Site resource
        site_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Site"),
            QQuery("label", properties['siteLabel'])
        )
```

    c. Read the managed device resource with the **PortUtils get_managed_device_resource** method.

```
    def create_managed_policy_map(self, resource):
        properties = resource['properties']

        # Read the L2VPN BandwidthProfile resource
        bwp_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.BandwidthProfile"),
            QQuery("label", properties['bandwidthProfile'])
        )

        # Read the L2VPN Site resource
        site_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Site"),
            QQuery("label", properties['siteLabel'])
        )

        # Read the L2VPN Site Managed Device resource
        site_device = site_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, site_device)
```

    d. Read the managed Policy Map product, then create the managed Policy Map based on the following data parameters:
        - **productId** (managed Policy Map product ID)
        - **label** (<device name>-<L2VPN name>.PolicyMap)
        - **device** (managed device ID)
        - **police**
            - **bc** (Bandwidth Profile cbs)
            - **be** (Bandwidth Profile be)
            - **cir** (Bandwidth Profile cir)
            - **pir** (Bandwidth Profile eir)
        - **class** (array)
            - **name** (<L2VPN name>-policy-class)

```
    def create_managed_policy_map(self, resource):
        properties = resource['properties']

        # Read the L2VPN BandwidthProfile resource
        bwp_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.BandwidthProfile"),
            QQuery("label", properties['bandwidthProfile'])
        )

        # Read the L2VPN Site resource
        site_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Site"),
            QQuery("label", properties['siteLabel'])
        )

        # Read the L2VPN Site Managed Device resource
        site_device = site_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, site_device)

        # Create a Managed Policy Map resource
        managed_policy_map_product =
self.bpo.market.get_products_by_resource_type('radevsim.resourceTypes.PolicyMap')[0]['id']
        try:
            managed_policy_map = self.bpo.resources.create(managed_device_resource["id"], {
                'productId': managed_policy_map_product,
                'label': f"{site_device}.{resource['label']}.PolicyMap",
                'properties': {
                    "name": f"{site_device}.{resource['label']}.PolicyMap",
                    "device": managed_device_resource["id"],
                    "police": {
                        "bc": bwp_resource['properties']['cbs'],
                        "be": bwp_resource['properties']['ebs'],
                        "cir": bwp_resource['properties']['cir'],
                        "pir": bwp_resource['properties']['eir']
                    },
                    "class": [{
                        "name": f"{resource['label']}-policy-class"
                    }]
                }
            })
        except:
            raise Exception(f"Failed to create Policy Map site_device}.{resource['label']}.PolicyMap.")
```

e. At the end, add a relationship between the L2VPN resource and managed Policy Map.

```
    def create_managed_policy_map(self, resource):
        properties = resource['properties']

        # Read the L2VPN BandwidthProfile resource
        bwp_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.BandwidthProfile"),
            QQuery("label", properties['bandwidthProfile'])
        )

        # Read the L2VPN Site resource
        site_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.Site"),
            QQuery("label", properties['siteLabel'])
        )

        # Read the L2VPN Site Managed Device resource
        site_device = site_resource["properties"]["deviceName"]
        managed_device_resource = PortUtils.get_managed_device_resource(self, site_device)

        # Create a Managed Policy Map resource
        managed_policy_map_product =
self.bpo.market.get_products_by_resource_type('radevsim.resourceTypes.PolicyMap')[0]['id']
```

```
        try:
            managed_policy_map = self.bpo.resources.create(managed_device_resource["id"], {
                'productId': managed_policy_map_product,
                'label': f"{site_device}.{resource['label']}.PolicyMap",
                'properties': {
                    "name": f"{site_device}.{resource['label']}.PolicyMap",
                    "device": managed_device_resource["id"],
                    "police": {
                        "bc": bwp_resource['properties']['cbs'],
                        "be": bwp_resource['properties']['ebs'],
                        "cir": bwp_resource['properties']['cir'],
                        "pir": bwp_resource['properties']['eir']
                    },
                    "class": [{
                        "name": f"{resource['label']}-policy-class"
                    }]
                }
            })
            self.bpo.relationships.add_relationship(resource['id'], managed_policy_map[0]['id'])
        except:
            raise Exception(f"Failed to create Policy Map site_device}.{resource['label']}.PolicyMap.")
```

f. Create a **delete_managed_policy_map_resource()** in the **Terminate** class in the **l2vpn.py** file.

g. In a try-catch loop, read the managed Policy Map resource using the **bpo.resources.get_dependencies_with_filters()**

```
    def delete_managed_policy_map_resource(self, resource_id):
        try:
            managed_policy_map = self.bpo.resources.get_dependencies_with_filters(
                resource_id,
                ExactTypeId("radevsim.resourceTypes.PolicyMap")
            )[0]
        except Exception as e:
            raise Exception(f"Could not delete managed Policy Map: {e}")
```

h. Delete both source and target relationships for the Policy Map resource, then delete the managed Policy Map resource itself and awaits termination.

```
    def delete_managed_policy_map_resource(self, resource_id):
        try:
            managed_policy_map = self.bpo.resources.get_dependencies_with_filters(
                resource_id,
                ExactTypeId("radevsim.resourceTypes.PolicyMap")
            )[0]
            self.bpo.relationships.delete_source_relationships(managed_policy_map['id'])
            self.bpo.relationships.delete_relationships(managed_policy_map['id'])
            self.bpo.resources.delete(managed_policy_map['id'])
            self.bpo.resources.await_termination(managed_policy_map['id'], managed_policy_map['label'],
True)
            log.info(f"Managed Policy Map {managed_policy_map['label']} deleted.")
        except Exception as e:
            raise Exception(f"Could not delete managed Policy Map: {e}")
```

47. Call the **create_managed_policy_map** method from the **L2VPN Activate Run** method.

```
class Activate(Plan):
    """
    Create L2VPN resource into market
    Create relationship with Customer, Site, and Bandwidth Profile resource
    Allocate VLAN
    Create managed VLAN resource
    Create managed logical port resource
    Create managed Policy Map resource
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
```

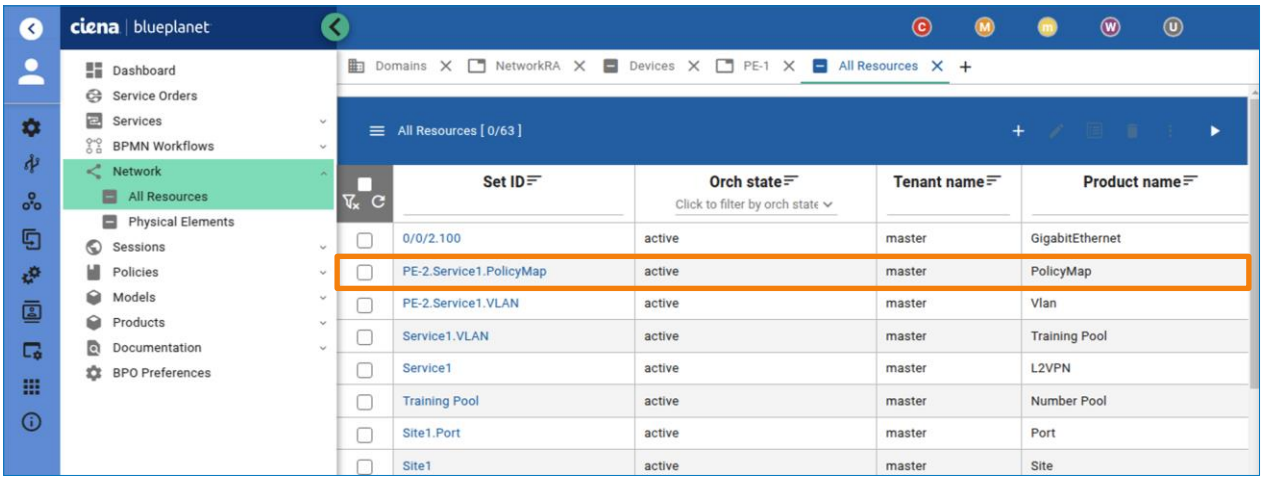
```
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        properties = resource['properties']
        self.assign_customer(properties)
        log.info(f"Assign Customer resource: {properties['customerLabel']}")

        self.assign_site(properties)
        log.info(f"Assign Site resource: {properties['siteLabel']}")

        self.assign_bandwidthprofile(properties)
        log.info(f"Assign Bandwidth Profile resource: {properties['bandwidthProfile']}")

        self.create_vlan_pool()
        allocated_vlan = self.allocate_vlan(resource)

        self.create_managed_vlan_resource(resource, allocated_vlan)
        self.create_managed_policy_map(resource)
        self.create_managed_logical_port(resource, allocated_vlan)

        log.info("Activate: DONE")
        return {}
```

48. Similarly, call the **delete_managed_policy_map** from the **Terminate run** method.

```
class Terminate(Plan):
    """
    Delete L2VPN resource from market
    Delete Customer, Site, and Bandwidthprofile relationships
    Delete managed VLAN resource
    Delete managed logical port resource
    Delete managed Policy Map resource
    """
    def run(self):
        log.info(f"Terminate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Terminate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        label = resource['label']
        try:
            vlan_num_res = self.bpo.resources.get_one_by_filters(
                    resource_type=POOL_NUMBER_RESOURCE_TYPE,
                    q_params={"label": f"{label}.VLAN"}
            )
            self.bpo.resources.delete(vlan_num_res['id'])
            self.bpo.resources.await_termination(vlan_num_res['id'], f"{label}.VLAN", False)
            log.info(f"VLAN for {label} released")
        except:
            log.info(f"VLAN for {label} not found")

        self.delete_managed_vlan_resource(resource_id)
        log.info("Deleting managed VLAN resource")

        self.delete_managed_policy_map_resource(resource_id)
        log.info("Deleting managed Policy Map resource")

        self.delete_managed_logical_port_resource(resource_id)
        log.info("Deleting managed Logical Port resource")

        self.delete_l2vpn_relationships(resource_id)
        log.info("Deleting Customer, Site and BandwidthProfile relationships")
```

```
        dependencies = self.bpo.resources.get_dependencies(resource_id)
        if dependencies:
            raise Exception(f"Site has dependencies ({dependencies})")

        log.info("Terminate: DONE")
        return {}
```

49. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
50. Create a new L2VPN resource. A new managed Policy Map resource will be created.



51. Verify that the policy map has been created on the managed device.

```
admin@PE-2 [/]> show configuration policy
dev-sim:policy {
   policy-map PE-2.Service1.PolicyMap {
      class Service1-policy-class {
      }
      police {
         cir 8000;
         bc 8000;
         pir 9000;
         be 9000;
      }
   }
}
```

52. Delete the L2VPN resource. Managed VLAN and Policy Map should get deleted, as well as the configuration on the devices.

End of Lab

Lab 6: Create Advanced Imperative Plans and Unit Testing

Objectives

- Implement Unit testing for your service code
- Use PlanSDK to change resource state through the Market objects
- Add support for multiple sites for the L2VPN resource

Documentation

PlanSDK documentation - *https://developer.blueplanet.com/docs/plansdk/index.html*

PlanEXT Unit Testing documentation - *https://developer.blueplanet.com/docs/planext/unit_testing.html*

Unittest Python Framework documentation - *https://docs.python.org/3/library/unittest.html*

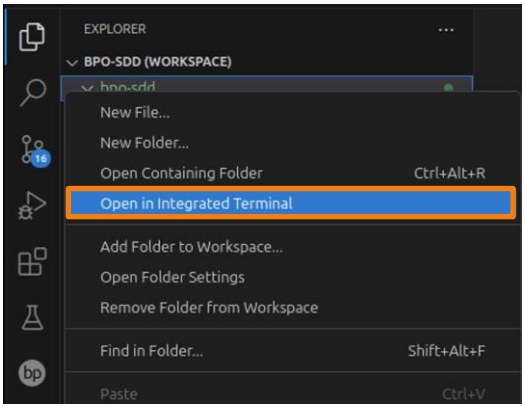
Task 1: Implement Unit Testing for Your Service Code

Unit testing is a fundamental part of software development. While the imperative plan code is relatively tightly coupled with the BPO server and its resources, there are ways to perform unit tests for your code. In this task, you learn how to set up such a unit testing environment using the Python unittest package, the BluePlanet PlanExt package, and its MockPlan class.

1. This lab continues the work done in the previous labs. Make sure that you have successfully completed the previous lab. If you did not complete the previous lab, make sure to onboard the resource definitions, service templates, and code from that lab to bring your BPO server into the initial state required for this lab. You can find them in the **~/Desktop/learning/bpo-sdd/solutions** folder.
2. VS Code, open the **bpo-sdd/requirements_env.txt**. Add the **planext** and **mock** pacakges.

```
-i https://pypi.org/simple
--extra-index-url https://gitlab.bptrn.com/api/v4/projects/2/packages/pypi/simple
plansdk
planext
mock
```

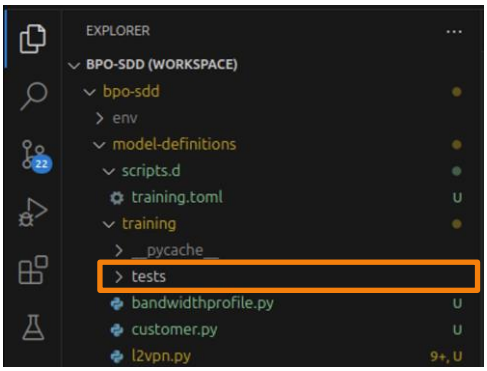
3. Right click the **bpo-sdd** folder and select **Open in Integrated Terminal**.



4. Install these requirements with the **pip install -r requirements_local.txt** command. This installs these Python modules to your virtual environment.

```
(env) student@POD-XX:~/training/bpo-sdd$ pip install -r requirements_env.txt
```

5. Create a new folder called **tests** in the **bpo-sdd/model-definitions/training** folder.



6. You now create a file and add some basic Python tests to learn how the *unittest* framework works in Python and how you can run the test suite in VS Code.
   a. Create a new file called **test.py** in the **bpo-sdd/model-definitions/training/tests** folder.
   b. Open the file and import the **unittest** package.

```
import unittest
```

   c. Open the unittest documentation listed in the documentation section of this lab guide and study the basic example to understand how to structure your test files.
   d. Add a new class to the **test.py** file called **MyFirstTest**. It must be inherited from the **unittest.TestCase** class.

```
import unittest

class MyFirstTest(unittest.TestCase):
```

   e. Add a **test_string_equal** method to this class. This method should declare two identical strings and use the **assertEqual** unittest method to ensure they are equal.

```
import unittest

class MyFirstTest(unittest.TestCase):
    def test_string_equal(self):
        a = 'foo'
        b = 'foo'
        self.assertEqual(a, b)
```

   f. Add an additional **test_string_not_equal** method. This one should declare two non-identical strings and use the **assertNotEqual** unittest method to assert that the strings are not identical.

```
    import unittest

class MyFirstTest(unittest.TestCase):
    def test_string_equal(self):
        a = 'foo'
        b = 'foo'
        self.assertEqual(a, b)

    def test_string_not_equal(self):
        a = 'foo'
        b = 'bar'
        self.assertNotEqual(a, b)
```

   g. Run the unit tests defined in the current Python script when executed as the main program by using the **unittest.main()** command.

```
import unittest

class MyFirstTest(unittest.TestCase):
    def test_string_equal(self):
```

```
        a = 'foo'
        b = 'foo'
        self.assertEqual(a, b)

    def test_string_not_equal(self):
        a = 'foo'
        b = 'bar'
        self.assertNotEqual(a, b)

if __name__ == '__main__':
    unittest.main()
```

h. In the terminal navigate to the **tests** folder.

```
(env) student@POD-XX:~/training/bpo-sdd$ cd model-definitions/training/tests/
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$
```

i. Execute the test suite by running the **python test.py** command. Make sure your virtualenv is activated.

```
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$ python test.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

j. You can also run them in verbose mode to get additional information with the **-v** flag.

```
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$ python test.py -v
test_string_equal (__main__.MyFirstTest) ... ok
test_string_not_equal (__main__.MyFirstTest) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

k. To see how a test looks if it fails, change one of the equal strings in the first test to a different one.

```
import unittest

class MyFirstTest(unittest.TestCase):
    def test_string_equal(self):
        a = 'foo'
        b = 'foobar'
        self.assertEqual(a, b)

    def test_string_not_equal(self):
        a = 'foo'
        b = 'bar'
        self.assertNotEqual(a, b)

if __name__ == '__main__':
    unittest.main()
```

l. Rerun the tests. This time, you get a warning that a test failed and why it did.

```
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$ python test.py
F.
======================================================================
FAIL: test_string_equal (__main__.MyFirstTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 12, in test_string_equal
    self.assertEqual(a, b)
AssertionError: 'foo' != 'foobar'
- foo
+ foobar
```

```
----------------------------------------------------------------------
Ran 2 tests in 0.000s

FAILED (failures=1)
```

m. Change the string back so the tests are successful.

7. You now implement a real unit test for one of your resources. Blue Planet provides a *PlanExt* Python package to enable developers to perform local unit testing using existing imperative plan code, which is invoked locally over the MockPlan class from the planext.mock Python module.
8. Import the BluePlanet's **MockPlan** class and the **mock** package, which is a standard Python package used to mocking objects and functions in Python testing. Import the **JSON** package as well since it is used in mocking the BPO database.

```
import unittest
import mock
import json
from planext.mock import MockPlan

class MyFirstTest(unittest.TestCase):
    def test_string_equal(self):
        a = 'foo'
        b = 'foo'
        self.assertEqual(a, b)

    def test_string_not_equal(self):
        a = 'foo'
        b = 'bar'
        self.assertNotEqual(a, b)

if __name__ == '__main__':
    unittest.main()
```

9. You now implement a suite of both successful and deliberately unsuccessful tests for the activation of your Port resource. When working on real service and resource development, it is considered good practice to have a very high percentage of code covered with unit tests. Still, for this task, only the Port Activate operation is covered.
    a. Import the **Activate** class from the **port.py** file as **PortActivate**. Since your imperative plan code is not a Python package, you must add the parent directory to the system path for the import to work.

```
import unittest
import mock
import json
import sys
from planext.mock import MockPlan

sys.path.append("../.")
from port import Activate as PortActivate
...
```

    b. Open the **bpo-sdd/model-definitions/training/port.py** file and change the util file import from **".util"** to **"util"** to make it work with relative imports.

```
from plansdk.apis.plan import Plan
from plansdk.apis.bpo import ExactTypeId
from util import failure, success
import logging
...
```

    c. Return to the **test.py** file. Create a **MockPortActivate** class, which inherits both from your imported **PortActivate** class and the required **MockPlan** class.

```
import unittest
import mock
```

```
import json
import sys
from planext.mock import MockPlan

sys.path.append("../.")
from port import Activate as PortActivate

class MockPortActivate(PortActivate, MockPlan):
```

10. The **MockPlan** class allows the user to write unit tests with actual data or mocked data easily. The idea behind the MockPlan is to provide a database lookup in the form of JSON files and to mock the behavior of methods that interact directly with the BPO server. When creating a test instance, you usually need to set up your mocked environment using the following methods that are a part of the MockPlan class:
    - setup_test()
    - setup_test_resources()
    - setup_test_bpo_mocking()
11. First, create the **setup_test_resource()** method.
    a. Add the method to the **MockPortActivate** class.

```
import unittest
import mock
import json
import sys
from planext.mock import MockPlan

sys.path.append("../.")
from port import Activate as PortActivate

class MockPortActivate(PortActivate, MockPlan):

    def setup_test_resources(self):
```

    b. You now need to create a JSON file that contains a mocked Port resource on which the test performs an Activate operation. Create a **port_resource.json** file in the **bpo-sdd/model-definitions/training/test** folder.
    c. Inside that file, add a dictionary that represents an arbitrary port resource on the BPO server. The structure that this JSON-based database expects is as follows:
        - items (array)
            - resource (object)
    d. This is how the **port_resource.json** file should look like. Notice that not all resource parameters are mocked – you can use a subset of resource properties that are used in a Plan operation. For Port activation, you need the ID, resource type, and properties.

```
{
    "items": [
        {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "resourceTypeId": "training.resourceTypes.Port",
            "properties": {
                "portName": "9/9/9",
                "deviceName": "PE-99",
                "portSpeed": "1Gbps"
            }
        }
    ]
}
```

e. In the **setup_test_resources()** method, read and add this JSON DB to the MockPlan with the **add_custom_db_by_filters()** method.

```
    def setup_test_resources(self):
        with open('port_resource.json', 'r') as file:
            port_resource = json.load(file)
        MockPlan.add_custom_db_by_filters("training.resourceTypes.Port", port_resource)
```

f. By doing this, any MockPlan “get” operation can now read this mocked data when performing unit tests.

12. Next, you need to implement the **setup_test_bpo_mocking()** method. This method will mock the behavior of desired PlanSDK (in other words, self.bpo.resources…) methods so that they do not have to interact with a live BPO server.

a. Add this method to the **MockPortActivate** class.

```
import unittest
import mock
import json
import sys
from planext.mock import MockPlan

sys.path.append("../.")
from port import Activate as PortActivate

class MockPortActivate(PortActivate, MockPlan):

    def setup_test_resources(self):
        with open('port_resource.json', 'r') as file:
            port_resource = json.load(file)
        MockPlan.add_custom_db_by_filters("training.resourceTypes.Port", port_resource)

    def setup_test_bpo_mocking(self):
```

b. Study your Port Activate class and all the classes (in other words, PortUtils) it depends on. You must mock every PlanSDK method used there so it works locally instead of with the BPO server. These methods are as follows:

- bpo.resources.get
- bpo.resources.create
- bpo.relationships.add_relationship
- bpo.resources.get_with_filters
- bpo.resources.get_dependents_with_filters

Some methods may need to be implemented either through MockPlan methods that know how to work with the mocked JSON database, other methods may need to be implemented manually since not all PlanSDK methods are supported by MockPlan, while others may be mocked so that whenever they are called, they are successful with no action performed.

c. Mock the **self.bpo.resources.get** method. You can use the **MockPlan.mock_resources_get** implementation, which performs a lookup in the JSON database you initialized in the previous step.

```
    def setup_test_bpo_mocking(self):
        self.bpo.resources.get = MockPlan.mock_resources_get
```

d. Continue with **self.bpo.resources.create** method. This one can be mocked with the **mock.Mock()** method, which will be successful every time without doing anything.

```
    def setup_test_bpo_mocking(self):
        self.bpo.resources.get = MockPlan.mock_resources_get
        self.bpo.resources.create = mock.Mock()
```

e. The same goes for the **self.bpo.relationships.add_relationship** method.

```
def setup_test_bpo_mocking(self):
    self.bpo.resources.get = MockPlan.mock_resources_get
    self.bpo.resources.create = mock.Mock()
    self.bpo.relationships.add_relationship = mock.Mock()
```

NOTE: Testing if resources get created and if there is a conflict with existing and duplicate resources is usually considered a part of integration testing. Although, with enough work, this can also be simulated in unit tests.

f. For the **self.bpo.resources.get_with_filters** method, you need to implement the method yourself. Create a **mock_get_with_filters** method in the **MockPortActivate** class and set it up as a mocked method in the **setup_test_bpo_mocking** method.

```
def setup_test_bpo_mocking(self):
    self.bpo.resources.get = MockPlan.mock_resources_get
    self.bpo.resources.create = mock.Mock()
    self.bpo.relationships.add_relationship = mock.Mock()
    self.bpo.resources.get_with_filters = MockPortActivate.mock_get_with_filters

def mock_get_with_filters(filter):
```

g. Return an array of objects with that method since the **self.bpo.resources.get_with_filters** method is used only to get a list of devices in the **Port Activate** operation, it only needs to return that list. If used multiple times, you must adapt the return based on the filter parameter. The devices only need their ID and label set for this unit test.

```
def mock_get_with_filters(filter):
    return [
        {
            "id": "a9d4666a-e414-11ed-a6db-e330bea54777",
            "label": "PE-77",
            "properties": {}
        },
        {
            "id": "a9d4666a-e414-11ed-a6db-e330bea54788",
            "label": "PE-88",
            "properties": {}
        },
        {
            "id": "a9d4666a-e414-11ed-a6db-e330bea54799",
            "label": "PE-99",
            "properties": {}
        }
    ]
```

h. Next, mock the **self.bpo.resources.get_dependents_with_filters** method. This one needs a mocked implementation as well. Mock it with the **mock_get_dependents_with_filters** method.

```
def setup_test_bpo_mocking(self):
    self.bpo.resources.get = MockPlan.mock_resources_get
    self.bpo.resources.create = mock.Mock()
    self.bpo.relationships.add_relationship = mock.Mock()
    self.bpo.resources.get_with_filters = MockPortActivate.mock_get_with_filters
    self.bpo.resources.get_dependents_with_filters = MockPortActivate.mock_get_dependents_with_filters

def mock_get_with_filters(filter):
    return [
        {
            "id": "a9d4666a-e414-11ed-a6db-e330bea54777",
            "label": "PE-77",
            "properties": {}
        },
        {
            "id": "a9d4666a-e414-11ed-a6db-e330bea54788",
            "label": "PE-88",
```

```
                "properties": {}
            },
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54799",
                "label": "PE-99",
                "properties": {}
            }
        ]

    def mock_get_dependents_with_filters(id, filter):
```

i. Since this method is used in reading the managed Port resources, return a set of them in this method. Make sure that both the mocked managed devices from the previous step and mocked managed ports from this step contain a device/port that was set as a property in the mocked JSON database.

```
    def setup_test_bpo_mocking(self):
        self.bpo.resources.get = MockPlan.mock_resources_get
        self.bpo.resources.create = mock.Mock()
        self.bpo.relationships.add_relationship = mock.Mock()
        self.bpo.resources.get_with_filters = MockPortActivate.mock_get_with_filters
        self.bpo.resources.get_dependents_with_filters = MockPortActivate.mock_get_dependents_with_filters

    def mock_get_with_filters(filter):
        return [
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54777",
                "label": "PE-77",
                "properties": {}
            },
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54788",
                "label": "PE-88",
                "properties": {}
            },
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54799",
                "label": "PE-99",
                "properties": {}
            }
        ]

    def mock_get_dependents_with_filters(id, filter):
        return [
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54777",
                "label": "7/7/7",
                "properties": {}
            },
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54788",
                "label": "8/8/8",
                "properties": {}
            },
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54799",
                "label": "9/9/9",
                "properties": {}
            }
        ]
```

13. Great, your mocked BPO environment is ready and set up. You can now add some test cases. Ideally, when testing, you want to test a “happy path” – a scenario where everything works as expected, and an “unhappy path” – a scenario where a failure in the test occurs. In the scenario of Port Activation, you will test 3 cases:
    - Port is activated successfully.
    - Port activation fails due to Port resource missing.
    - Port activation fails due to managed port resource missing.
14. Implement the first test case where the Activation is successful.
    a. Add a **PortTest** class, which is inherited from the **unittest.TestCase** class.

```
class PortTest(unittest.TestCase):
```

    b. Within that class, add a **setup_port_test** method, which is a reusable method for setting up a test environment for every test case.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
```

    c. Inside, create a **test_record** dictionary with the **id** set to the resource ID from the mocked JSON database. This is a record that the plan is executed against. Include an empty **properties** object as well, but there is no need for any properties in this test suite.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
        }
```

    d. Add the **setup_test**, **setup_test_resources,** and **setup_test_bpo_mocking** methods to set up the test environment. The **setup_test** methods require the **test_record** as the first parameter and an optional absolute path to the JSON DB file as well. Since you took care of the DB in the **setup_test_resources** method, there is no need for this absolute path.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
        }
        self.activate_port.setup_test(test_record, None)
        self.activate_port.setup_test_resources()
        self.activate_port.setup_test_bpo_mocking()
```

    e. Add your first test case by adding a **test_port_activate_success()** method.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
        }
        self.activate_port.setup_test(test_record, None)
        self.activate_port.setup_test_resources()
        self.activate_port.setup_test_bpo_mocking()

    def test_port_activate_success(self):
```

f. To run a test, you first need to instantiate a **MockPortActivate** class. It also requires inputs based on the actual imperative plan code. Add a dictionary with an arbitrary **uri** parameter and a **resourceId** parameter, which is the same as the one defined in the JSON db.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
        }
        self.activate_port.setup_test(test_record, None)
        self.activate_port.setup_test_resources()
        self.activate_port.setup_test_bpo_mocking()

    def test_port_activate_success(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
```

g. Call the **setup_port_test()** method afterward and execute the **run()** method from the **MockPortActivate** class – this method has all the Activate logic from the **port.py** file.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
        }
        self.activate_port.setup_test(test_record, None)
        self.activate_port.setup_test_resources()
        self.activate_port.setup_test_bpo_mocking()

    def test_port_activate_success(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
        self.setup_port_test()
        self.activate_port.run()
```

h. You can now run your first BPO unit test. Open the command terminal and run the test suite. The **test_port_activate_success** test should be completed successfully.

```
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$ python test.py -v
test_boolean (__main__.MyFirstTest) ... ok
test_string (__main__.MyFirstTest) ... ok
test_port_activate_success (__main__.PortTest) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.004s

OK
```

15. Next, you implement a failure scenario – for example, it might happen that the resource is missing, or the resource ID specified is incorrect. You want the operation to fail at that point.

a. Copy the test_port_activate_success method to a new test_port_activate_failure_no_resource method.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
```

```
        }
        self.activate_port.setup_test(test_record, None)
        self.activate_port.setup_test_resources()
        self.activate_port.setup_test_bpo_mocking()

    def test_port_activate_success(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
        self.setup_port_test()
        self.activate_port.run()

    def test_port_activate_failure_no_resource(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
        self.setup_port_test()
        self.activate_port.run()
```

b. Change the **resourceId** to an arbitrary different string.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
        }
        self.activate_port.setup_test(test_record, None)
        self.activate_port.setup_test_resources()
        self.activate_port.setup_test_bpo_mocking()

    def test_port_activate_success(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
        self.setup_port_test()
        self.activate_port.run()

    def test_port_activate_failure_no_resource(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "this-resource-id-does-not-exist"}
        )
        self.setup_port_test()
        self.activate_port.run()
```

c. Rerun the test suite. The test suite will fail, but this is expected.

```
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$ python test.py -v
test_boolean (__main__.MyFirstTest) ... ok
test_string (__main__.MyFirstTest) ... ok
test_port_activate_failure_no_resource (__main__.PortTest) ... ERROR
test_port_activate_success (__main__.PortTest) ... ok

======================================================================
ERROR: test_port_activate_failure_no_resource (__main__.PortTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/student/Git/model-definitions/training/test/test.py", line 106, in
test_port_activate_failure_no_resource
    self.activate_port.run()
  File "/home/student/Git/model-definitions/training/test/.././port.py", line 87, in run
    properties = resource['properties']
TypeError: 'NoneType' object is not subscriptable
```

```
----------------------------------------------------------------------
Ran 4 tests in 0.005s

FAILED (errors=1)
```

d. Since this failure, with this specific error and message, is expected with such input, you must ensure that the test suite acknowledges that as well. Use the **assertRaises** method from the **unittest** package to capture the exception and then assert its error message with the **assertEqual** method.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
        }
        self.activate_port.setup_test(test_record, None)
        self.activate_port.setup_test_resources()
        self.activate_port.setup_test_bpo_mocking()

    def test_port_activate_success(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
        self.setup_port_test()
        self.activate_port.run()

    def test_port_activate_failure_no_resource(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "this-resource-id-does-not-exist"}
        )
        self.setup_port_test()
        with self.assertRaises(Exception) as context:
            self.activate_port.run()
        self.assertEqual(str(context.exception), "'NoneType' object is not subscriptable")
```

e. Rerun the test suite. It should be successful since the expected exception is caught and verified.

```
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$ python test.py -v
test_boolean (__main__.MyFirstTest) ... ok
test_string (__main__.MyFirstTest) ... ok
test_port_activate_failure_no_resource (__main__.PortTest) ... ok
test_port_activate_success (__main__.PortTest) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.005s

OK
```

16. For the final test, you implement a scenario in which one of the managed port resources that the Port resource forms a relationship with is missing. Since your mocked **mock_get_dependents_with_filters()** method returns a correct list, you need an additional method that returns an empty list.

    a. Add a **mock_get_dependents_with_filters_empty** method to the **MockPortActivate** class, which returns an empty array.

```
    def mock_get_dependents_with_filters(id, filter):
        return [
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54777",
                "label": "7/7/7",
                "properties": {}
            },
```

```
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54788",
                "label": "8/8/8",
                "properties": {}
            },
            {
                "id": "a9d4666a-e414-11ed-a6db-e330bea54799",
                "label": "9/9/9",
                "properties": {}
            }
        ]

    def mock_get_dependents_with_filters_empty(id, filter):
        return []
```

b. You need a way to tell the **setup_test_bpo_mocking** method which method it should use in a specific test setup. Add a **mock_ports** variable to the **MockPortActivate** class and set its value to **True**.

```
class MockPortActivate(PortActivate, MockPlan):

    mock_ports = True
```

c. Update the **setup_test_bpo_mocking** class to use the empty-return method if the **mock_ports** variable is set to **False.**

```
class MockPortActivate(PortActivate, MockPlan):

    mock_ports = True

    def setup_test_resources(self):
        with open('port_resource.json', 'r') as file:
            port_resource = json.load(file)
        MockPlan.add_custom_db_by_filters("training.resourceTypes.Port", port_resource)

    def setup_test_bpo_mocking(self):
        self.bpo.resources.get = MockPlan.mock_resources_get
        self.bpo.resources.create = mock.Mock()
        self.bpo.relationships.add_relationship = mock.Mock()
        self.bpo.resources.get_with_filters = MockPortActivate.mock_get_with_filters
        if self.mock_ports:
            self.bpo.resources.get_dependents_with_filters =
MockPortActivate.mock_get_dependents_with_filters
        else:
            self.bpo.resources.get_dependents_with_filters =
MockPortActivate.mock_get_dependents_with_filters_empty
```

d. Add a new test case – create a **test_port_activate_failure_no_port** method in the **PortTest** class. Copy the contents from the **test_port_activate_failure_no_resource** method.

```
class PortTest(unittest.TestCase):

    def setup_port_test(self):
        test_record = {
            "id": "a9d4666a-e414-11ed-a6db-e330bea5476c",
            "properties": {}
        }
        self.activate_port.setup_test(test_record, None)
        self.activate_port.setup_test_resources()
        self.activate_port.setup_test_bpo_mocking()

    def test_port_activate_success(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
        self.setup_port_test()
        self.activate_port.run()
```

```
    def test_port_activate_failure_no_resource(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "this-resource-id-does-not-exist"}
        )
        self.setup_port_test()
        with self.assertRaises(Exception) as context:
            self.activate_port.run()
        self.assertEqual(str(context.exception), "'NoneType' object is not subscriptable")

    def test_port_activate_failure_no_port(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "this-resource-id-does-not-exist"}
        )
        self.setup_port_test()
        with self.assertRaises(Exception) as context:
            self.activate_port.run()
        self.assertEqual(str(context.exception), "'NoneType' object is not subscriptable")
```

e. Set the correct **resourceId** and set the **mock_ports** variable for that class instance to **False**.

```
    def test_port_activate_failure_no_port(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
        self.activate_port.mock_ports = False
        self.setup_port_test()
        with self.assertRaises(Exception) as context:
            self.activate_port.run()
        self.assertEqual(str(context.exception), "'NoneType' object is not subscriptable")
```

f. Rerun the test suite. The last test will fail, telling you that the managed port does not exist.

```
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$ python test.py -v
test_boolean (__main__.MyFirstTest) ... ok
test_string (__main__.MyFirstTest) ... ok
test_port_activate_failure_no_port (__main__.PortTest) ... FAIL
test_port_activate_failure_no_resource (__main__.PortTest) ... ok
test_port_activate_success (__main__.PortTest) ... ok

======================================================================
FAIL: test_port_activate_failure_no_port (__main__.PortTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/student/Git/model-definitions/training/test/test.py", line 119, in
test_port_activate_failure_no_port
    self.assertEqual(str(context.exception), "'NoneType' object is not subscriptable")
AssertionError: 'Managed port 9/9/9 does not exist!' != "'NoneType' object is not subscriptable"
- Managed port 9/9/9 does not exist!
+ 'NoneType' object is not subscriptable


----------------------------------------------------------------------
Ran 5 tests in 0.007s

FAILED (failures=1)
```

g. Update the assertion statement with this expected error message.

```
    def test_port_activate_failure_no_port(self):
        self.activate_port = MockPortActivate(
            {"uri": "http://bpocore",
            "resourceId": "a9d4666a-e414-11ed-a6db-e330bea5476c"}
        )
        self.activate_port.mock_ports = False
        self.setup_port_test()
```

```
        with self.assertRaises(Exception) as context:
            self.activate_port.run()
        self.assertEqual(str(context.exception), 'Managed port 9/9/9 does not exist on device PE-99!')
```

h. Rerun the test suite. All tests should be successful now.

```
(env) student@POD-XX:~/training/bpo-sdd/model-definitions/training/tests$ python test.py -v
test_boolean (__main__.MyFirstTest) ... ok
test_string (__main__.MyFirstTest) ... ok
test_port_activate_failure_no_port (__main__.PortTest) ... ok
test_port_activate_failure_no_resource (__main__.PortTest) ... ok
test_port_activate_success (__main__.PortTest) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.007s

OK
(local)
```

17. Revert the dependency change you made in the **port.py** file. Use a relative import once again for the **utils** file.

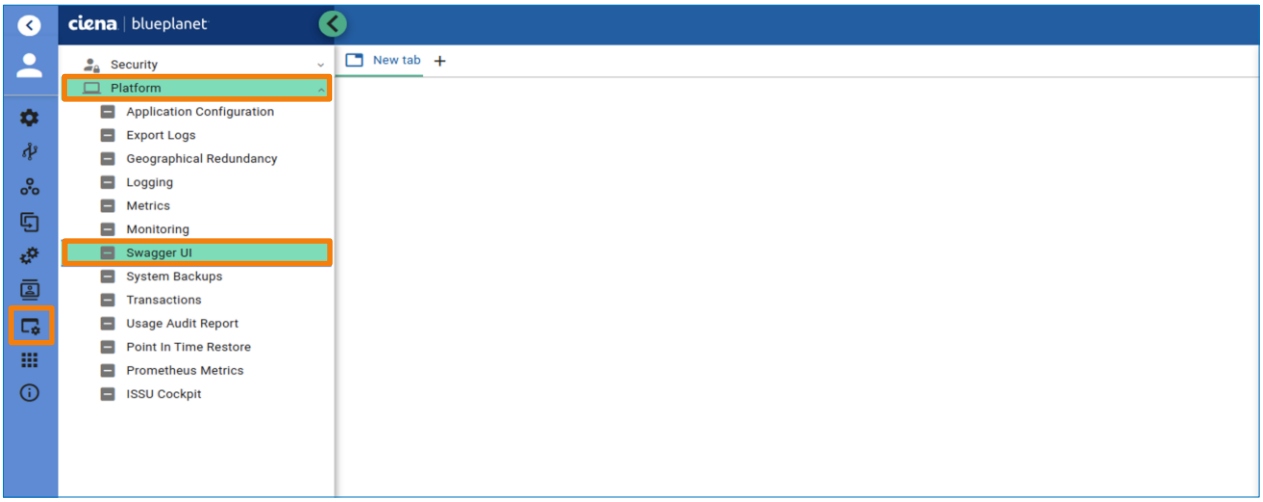
```
from plansdk.apis.plan import Plan
from plansdk.apis.bpo import ExactTypeId
from .util import failure, success
import logging

log = logging.getLogger(__name__)

class PortUtils():
```

18. You can also remove the entire **bpo-sdd/model-definitions/training/tests** folder since it will not be needed in the following tasks.

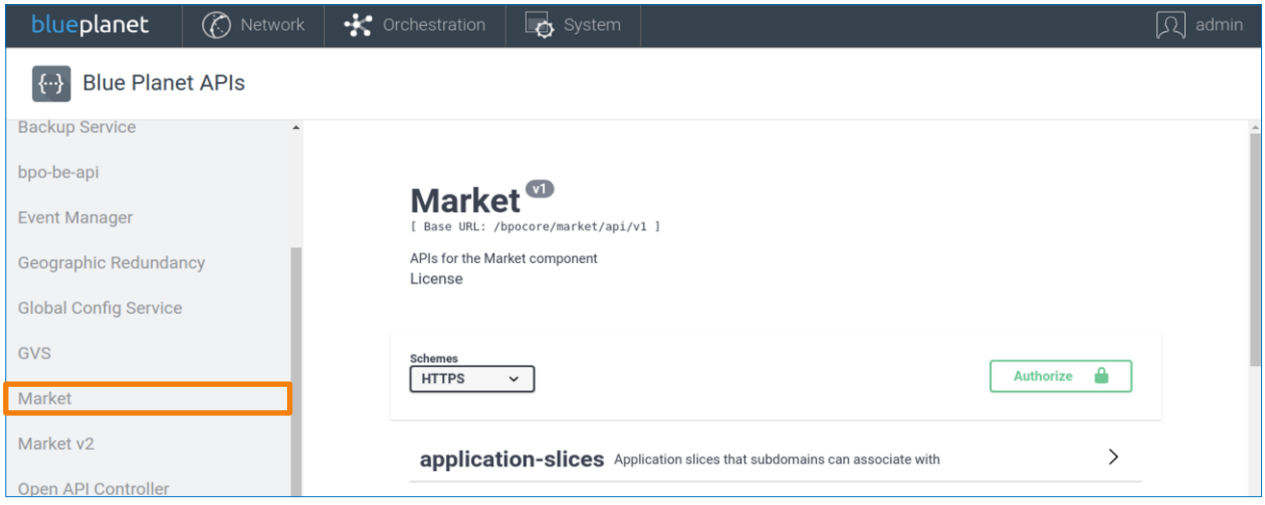
Task 2: Use PlanSDK to Change Resource State Through the Market Objects

The PlanSDK Python package features many classes and modules that allow you to work with resources and relationships. While you have mostly worked with **bpo.resources** and **bpo.relationships** up until now, you can try out the **bpo.market** module in this task, allowing you to directly manipulate the market objects. You first try to create the Customer resource manually through the API calls in the Swagger UI, then rework the imperative plan for the Site resource for the Port resource to be created through the Market objects.

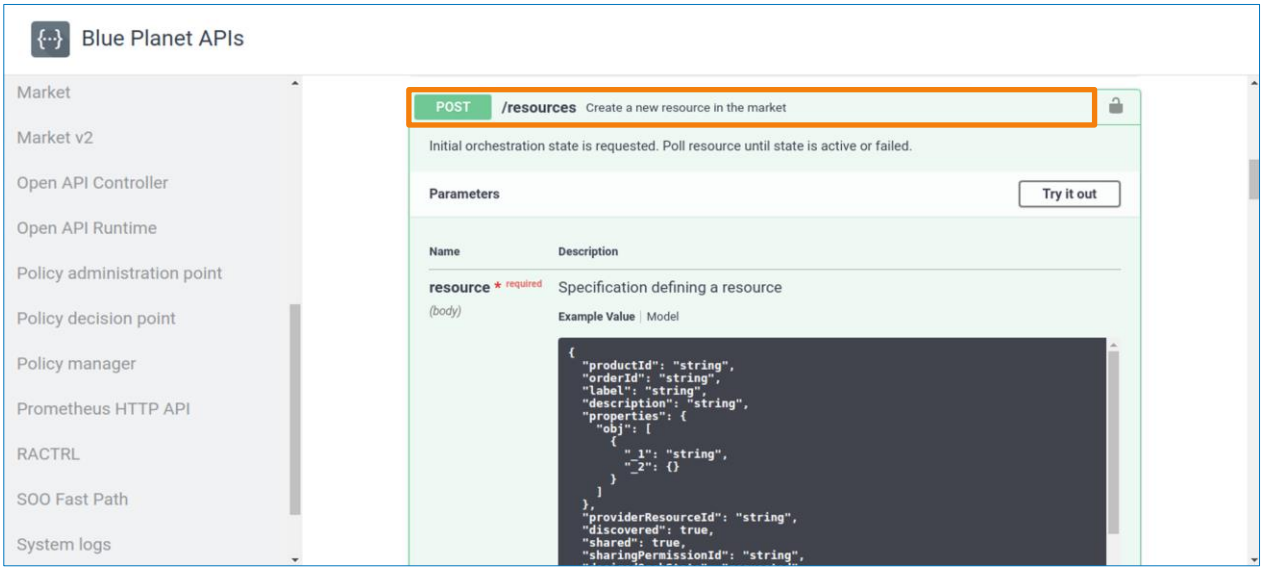
1. Navigate to **System > Platform > Swagger UI** to open a new tab that will provide you access to the Swagger UI for the REST API gateway.



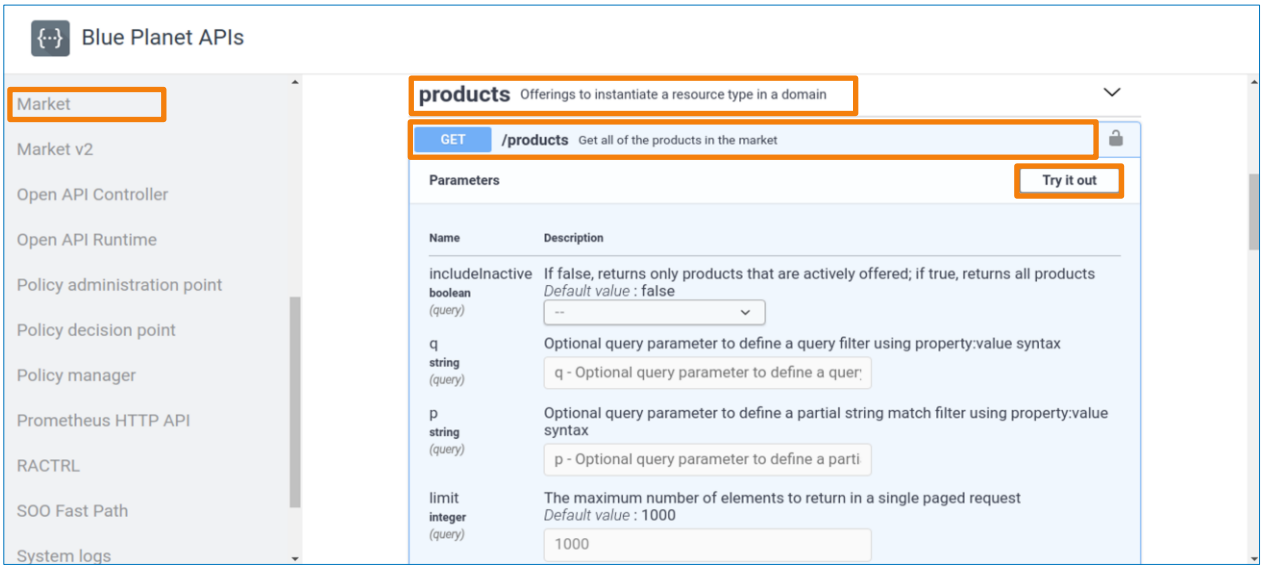
2. Click the **Market** REST API component.



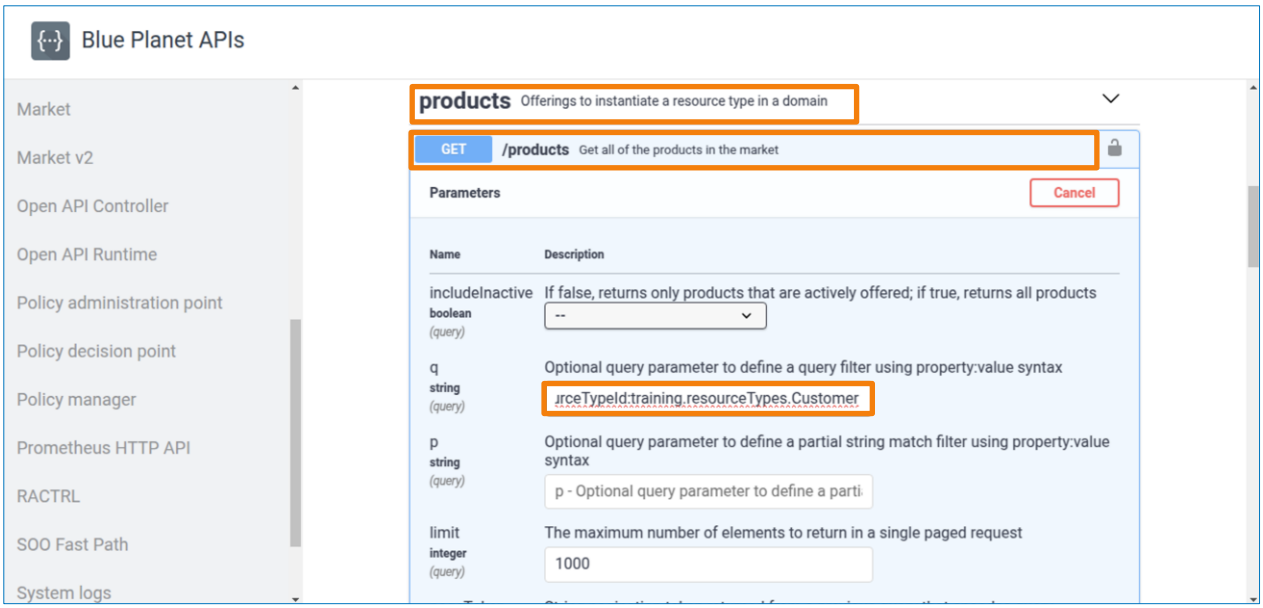
3. Click on the **resources** and then on **POST /resources**. This expands the window, which allows you to create resources with fine-grained parameter control. Study the required and available inputs that are available to you. There is a wide selection of available parameters, but you usually do not need to fill out all of them for each object you are creating.



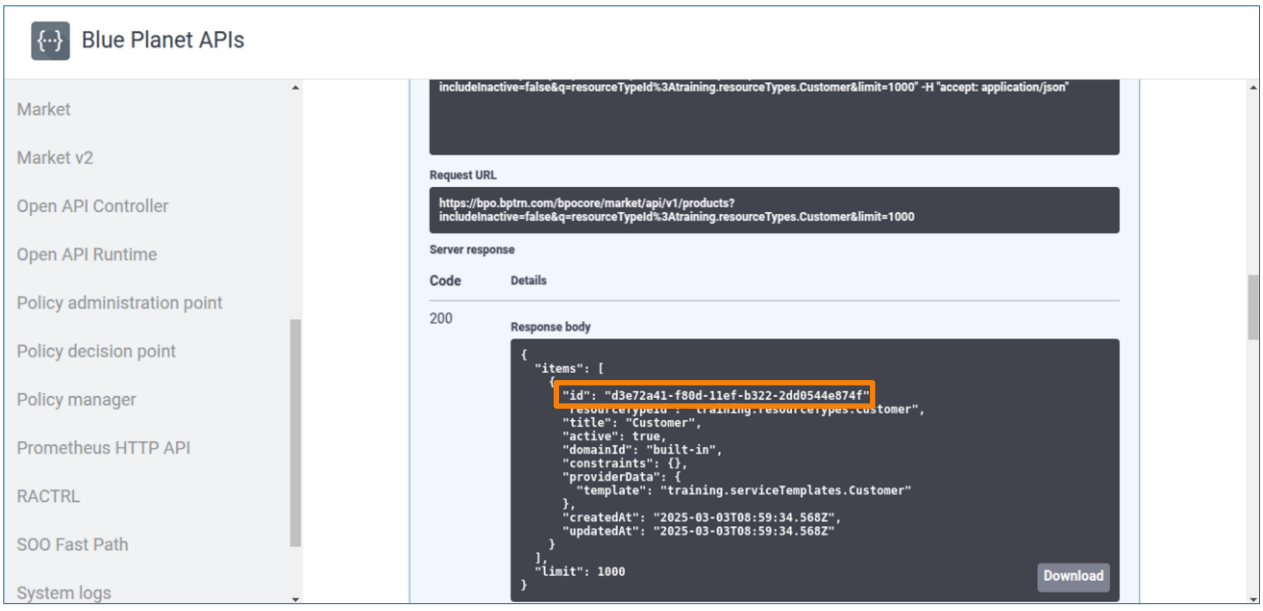
4. Create a new Customer resource.
   a. First, find out the Product ID for your Customer resource. Click the **products > GET /products** REST API call, then click the **Try it out** button, scroll down, and click the **Execute** button to initiate the GET request to list the existing products in the *Market* database.



c. Add the following **q** query to match a specific product by its **resourceTypeId:training.resourceTypes.Customer** and click the **Execute** button to initiate the GET request.



d. Scroll down to the Response body and note down the **id** value, which should only contain a single product. This is the ID you use in the creation of a Customer resource.



e. Return to the **resources > POST /resources** REST API call and click the **Try it out** button. Modify the request body so that it contains the following parameters:

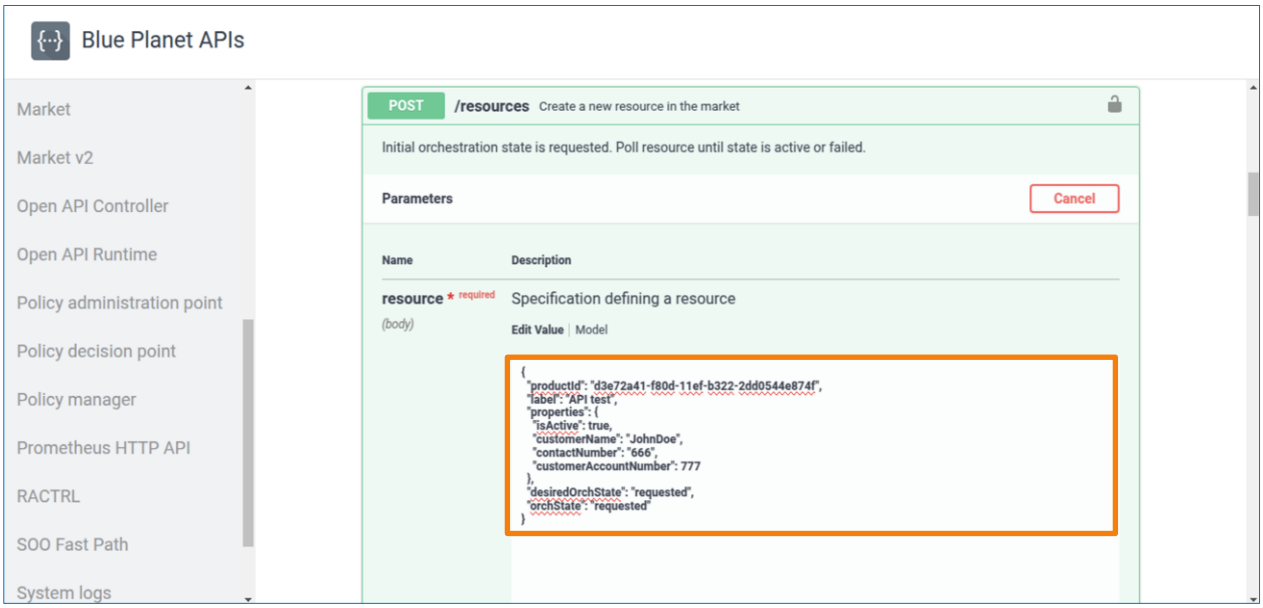
- **productId** (replace the **<PRD_ID>** in the following snippet with your product ID)
- **label** (a name for your resource)
- **properties** (dictionary containing the Customer properties)
- **desiredOrchState** (desired orchestration state)
- **orchState** (actual orchestration state)

For now, set the **desiredOrchState** and **orchState** to **requested** and fill in the other parameters as you see fit.

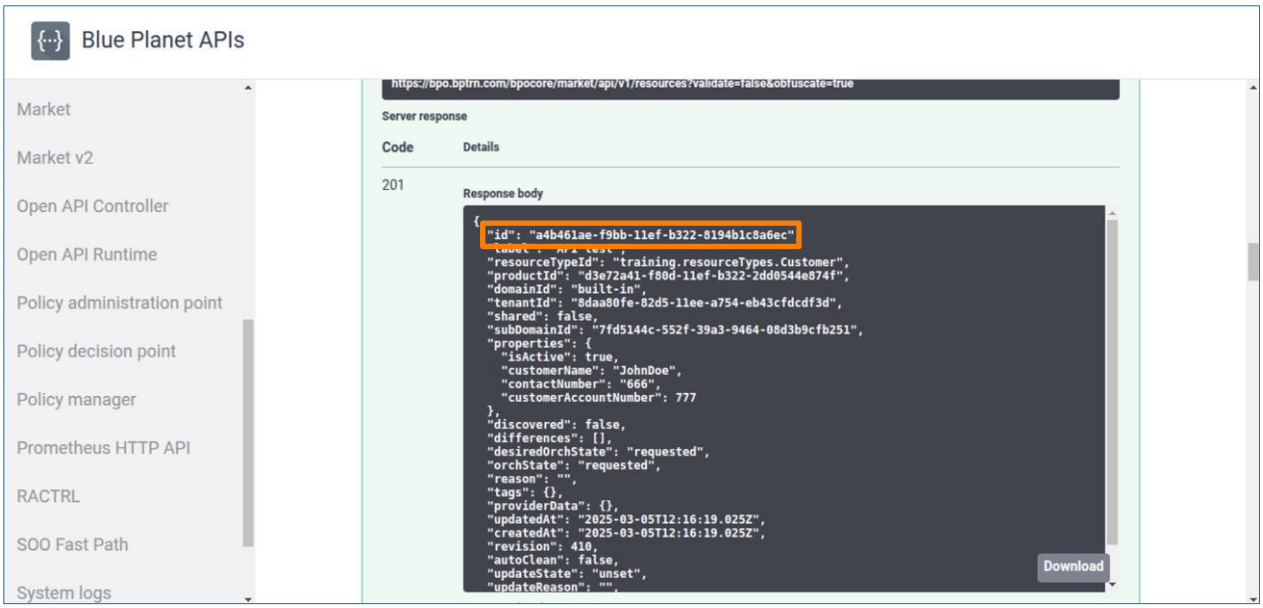
f. Use the following payload in the POST request's body:

```
{
  "productId": "<PRD_ID>",
  "label": "API test",
  "properties": {
    "isActive": true,
    "customerName": "JohnDoe",
    "contactNumber": "666",
    "customerAccountNumber": 777
  },
  "desiredOrchState": "requested",
  "orchState": "requested"
}
```

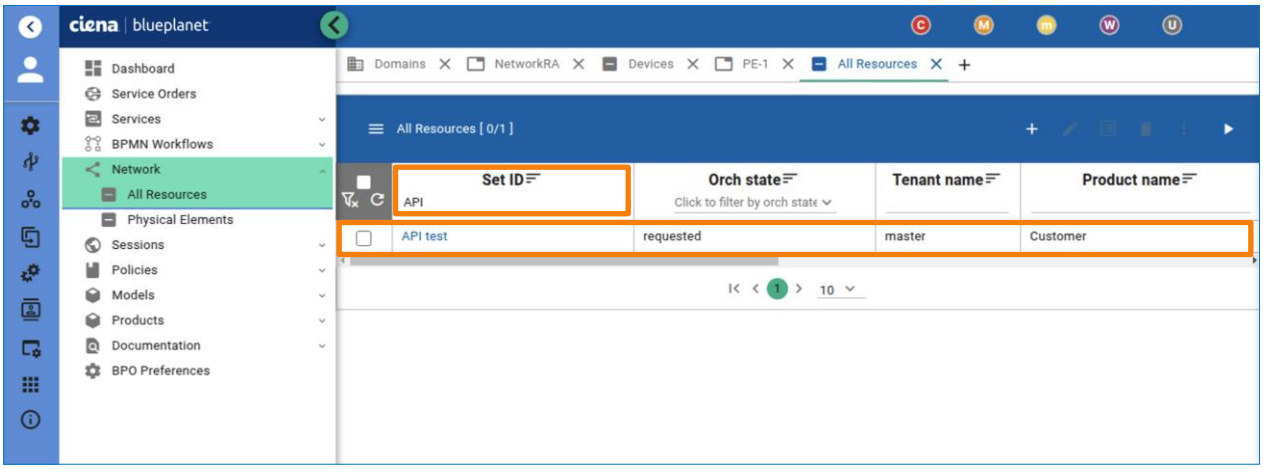
g. You should have something similar in Swagger.



h. Click the **Execute** button. The API call should return a 201 Created HTTP Response Code, meaning that the resource was created successfully. Examine the response body for resource details. Note down the **id** value since this is your Customer resource ID.



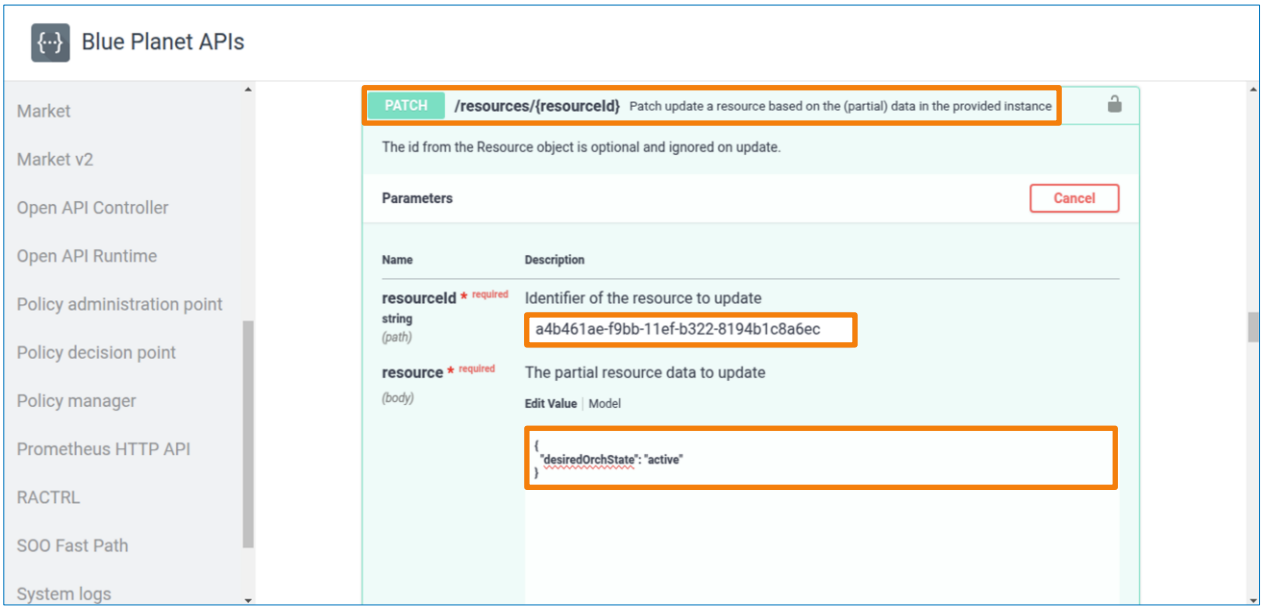
i. Open the BPO UI and navigate to the **Network > All Resources**. Find your newly created Customer resource there. You will notice that it is still in the **requested** stage since that is the desired orchestration state you set it to.



5. You can also change the resource states through the Market API. Expand the **resources > PATCH /resources/{resourceId}** REST API call.
   Use the resource ID from the **POST /resources** API call and set the *desiredOrchState* to **active.**
   a. Use the following payload in the POST request's body:

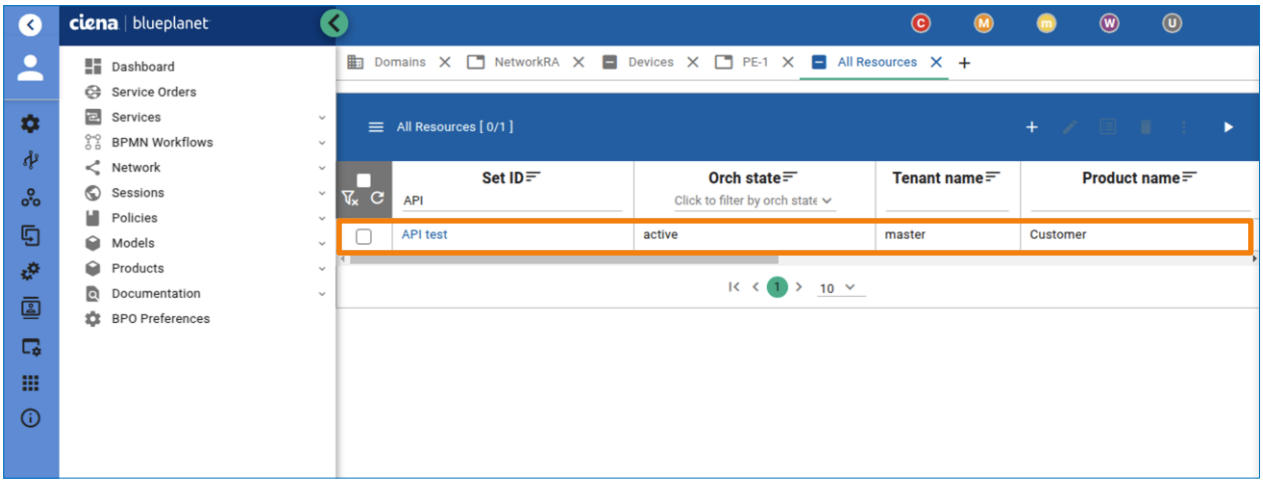
```
{
  "desiredOrchState": "active"
}
```

   b. You should have something similar in Swagger.



   c. Click the **Execute** button and make sure you get a ***200 OK*** response back.

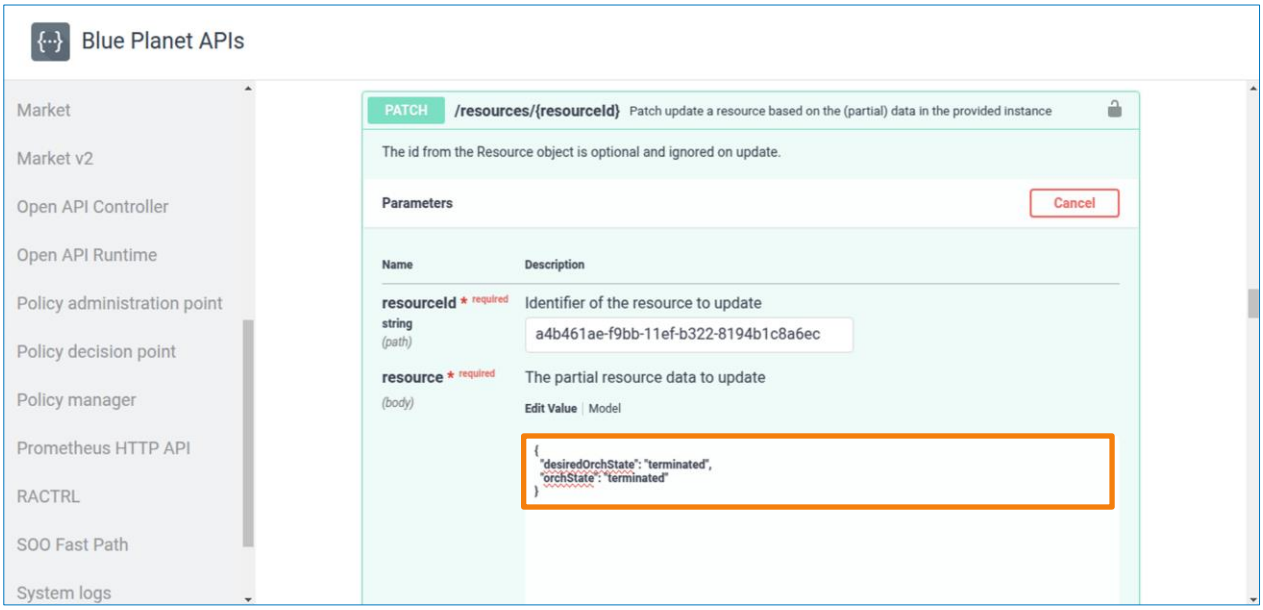
   d. Open the **BPO UI** and find your Customer resource. The state should now be **active.**



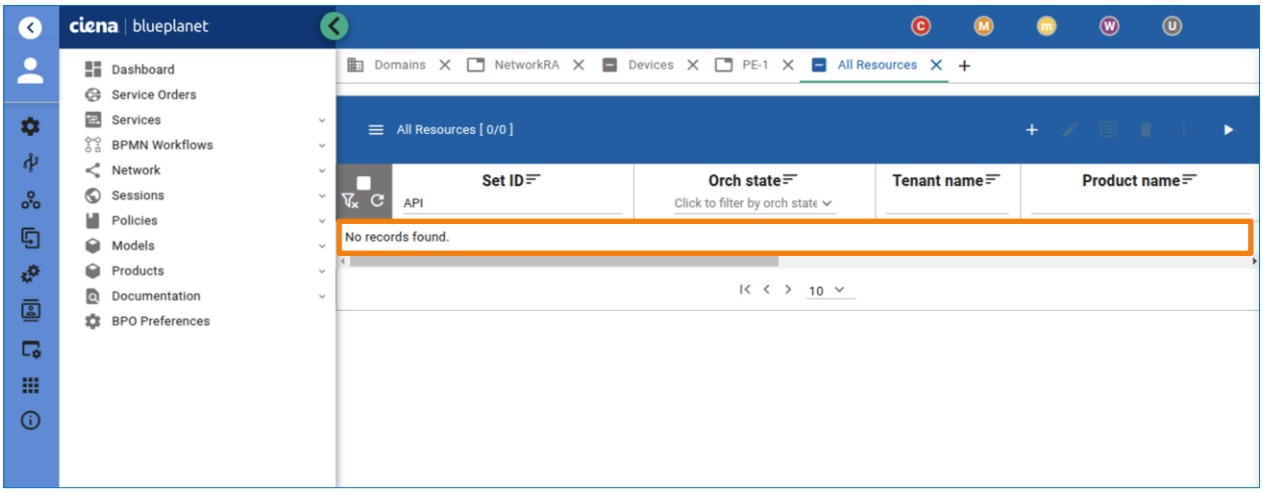
6. Finally, create another request for **resources > PATCH /resources/{resourceId}** REST API call to patch your Customer resource. This time, set both the **desiredOrchState** and **orchState** to **terminated**.
   a. Use the following payload in the POST request's body:

```
{
  "desiredOrchState": "terminated",
  "orchState": "terminated"
}
```

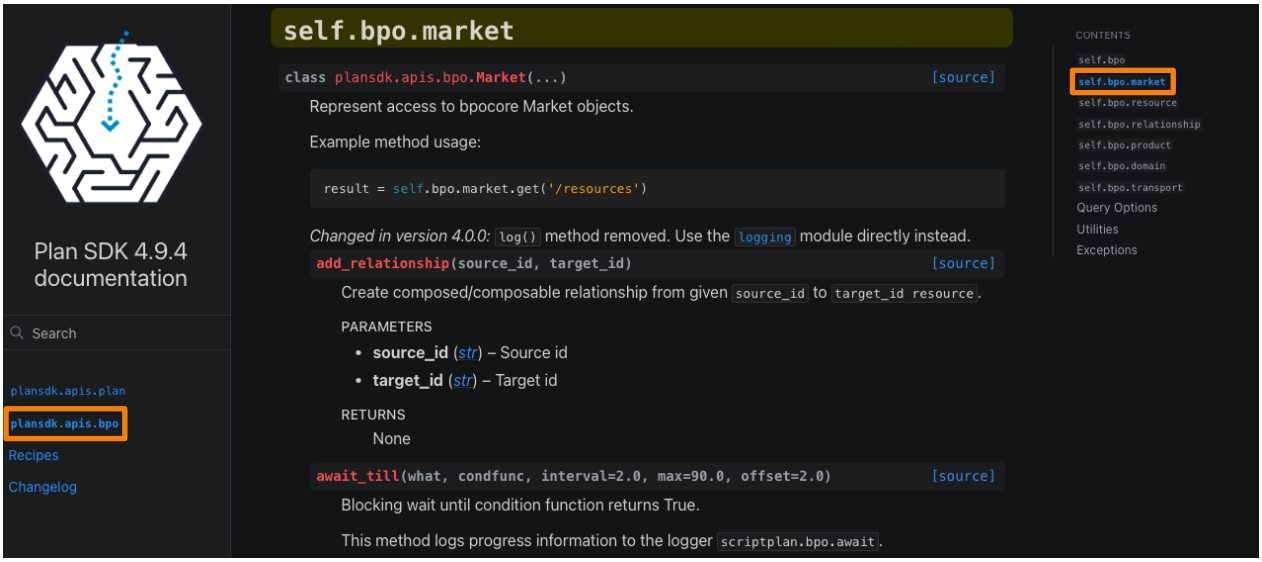
   b. You should have something similar in Swagger.



   c. Execute the API call. Open the **BPO UI**. The Customer resource is now gone since it has been terminated.



7. You now perform a similar set of actions to create a Port resource from the Site resource imperative plan in Python, using the PlanSDK Market Objects.
   a. Study the **self.bpo.market** section of the *PlanSDK* documentation to understand what methods are available to you.



   b. Open the **site.py** file from the **bpo-sdd/model-definitions/training** folder. Delete the contents of the **create_port()** method in the **Activate** class. You will re-create the same method, imitating the manual API calls you did over the Swagger UI.

```
class Activate(Plan):
    """
    Creates Site resource into market
    Creates Port resource into market
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        port = self.create_port(resource)
        log.info(f"Create Port resource: {port}")

        log.info("Activate: DONE")
        return {}

    def create_port(self, resource):
```

   c. Read the resource properties and the Port product ID. This part can stay identical to the previous implementation since it uses the Market API Object.

```
    def create_port(self, resource):
        properties = resource['properties']
        port_product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Port')[0]
```

d. Implement the creation of the resource over with the **bpo.market.post()** method.

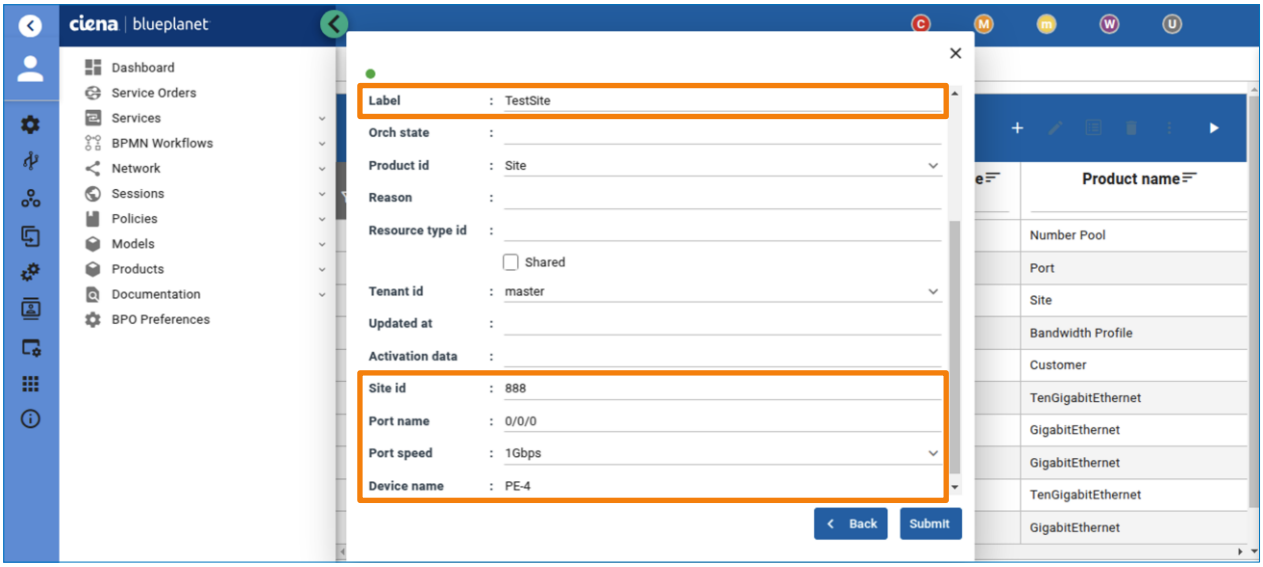
```
def create_port(self, resource):
    properties = resource['properties']
    port_product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Port')[0]

    # Create the resource
    port_resource = self.bpo.market.post("/resources", {
        'productId': port_product['id'],
        'label': f"{resource['label']}.Port",
        'properties': {'deviceName': properties["deviceName"],
                       'portName': properties["portName"],
                       'portSpeed': properties["portSpeed"]},
        "desiredOrchState": "requested",
        "orchState": "requested"
    })
```

e. Activate the Port resource with the **bpo.market.patch()** method.

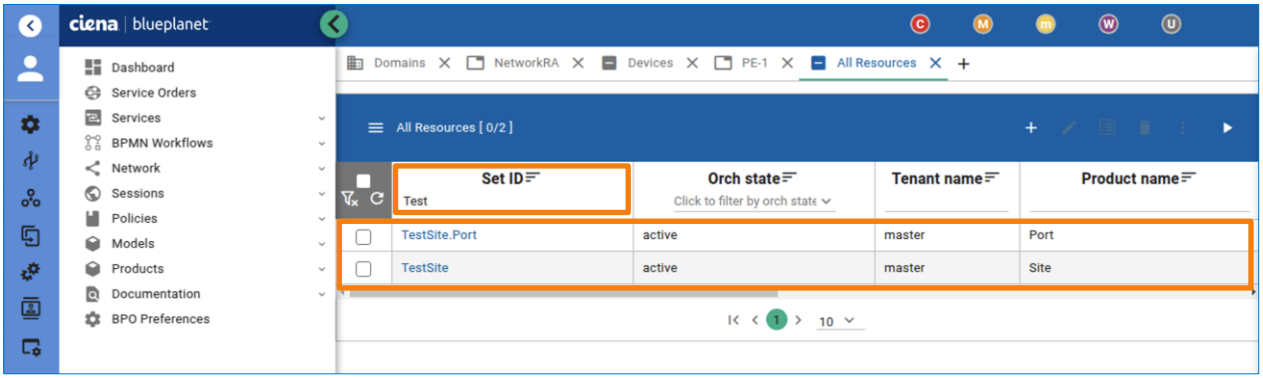
```
def create_port(self, resource):
    properties = resource['properties']
    port_product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Port')[0]

    # Create the resource
    port_resource = self.bpo.market.post("/resources", {
        'productId': port_product['id'],
        'label': f"{resource['label']}.Port",
        'properties': {'deviceName': properties["deviceName"],
                       'portName': properties["portName"],
                       'portSpeed': properties["portSpeed"]},
        "desiredOrchState": "requested",
        "orchState": "requested"
    })

    # Patch the desired operational state
    self.bpo.market.patch(f"/resources/{port_resource['id']}", {
        "desiredOrchState": "active"
    })
```

f. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

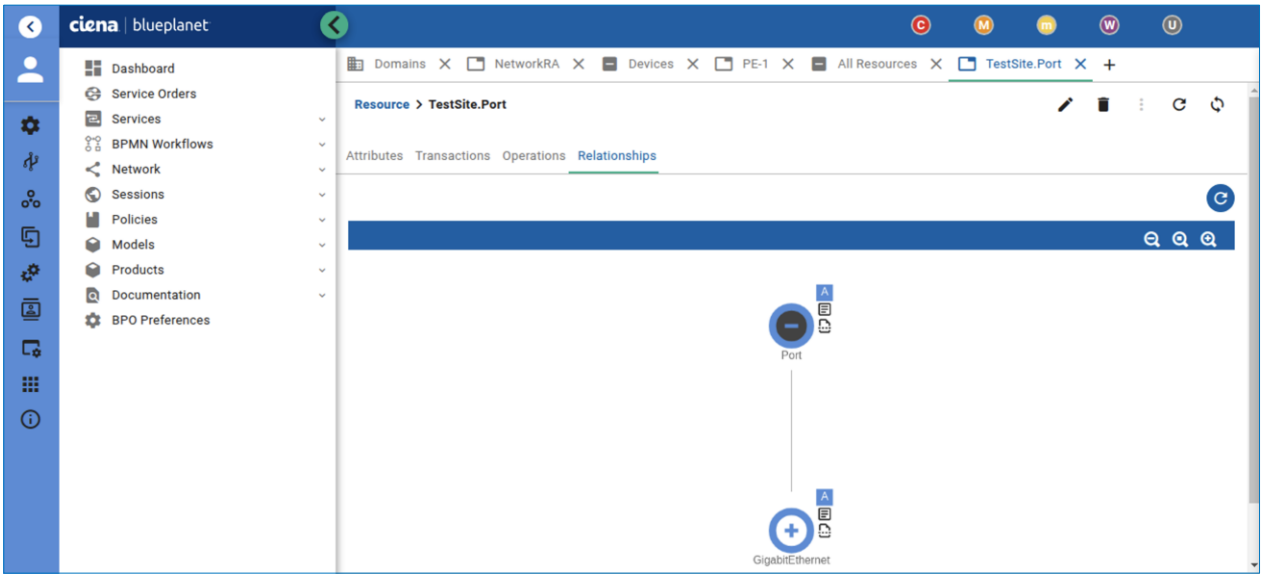
g. Create a new Site resource over BPO UI.



h. Both Site and Port resources should be created successfully.



i. Inspect the Port resource and open the Relationships tab. You notice that the only relationship attached to this Port resource is the one to the managed port resource (created in the imperative plan for the Port resource). You might remember that the relationship formed between the Site and the Port resource automatically when creating resources over non-Market Objects.



j. Delete both the Port and the Site resource.

k. Open the **site.py** file again. Add another **bpo.market.post** call, this time towards the **/relationships** URL. In addition to the sourceId and targetId parameters, you also need to include the following parameters:

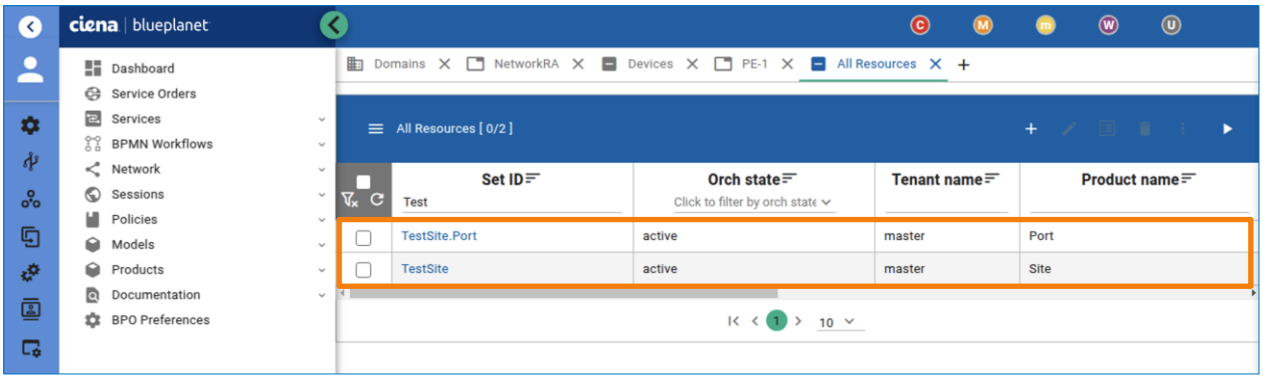
- relationshipTypeId: **tosca.relationshipTypes.DependsOn**
- requirementName: **composed**
- capabilityName: **composable**

```
def create_port(self, resource):
    properties = resource['properties']
    port_product = self.bpo.market.get_products_by_resource_type('training.resourceTypes.Port')[0]

    # Create the resource
    port_resource = self.bpo.market.post("/resources", {
        'productId': port_product['id'],
        'label': f"{resource['label']}.Port",
        'properties': {'deviceName': properties["deviceName"],
                       'portName': properties["portName"],
```

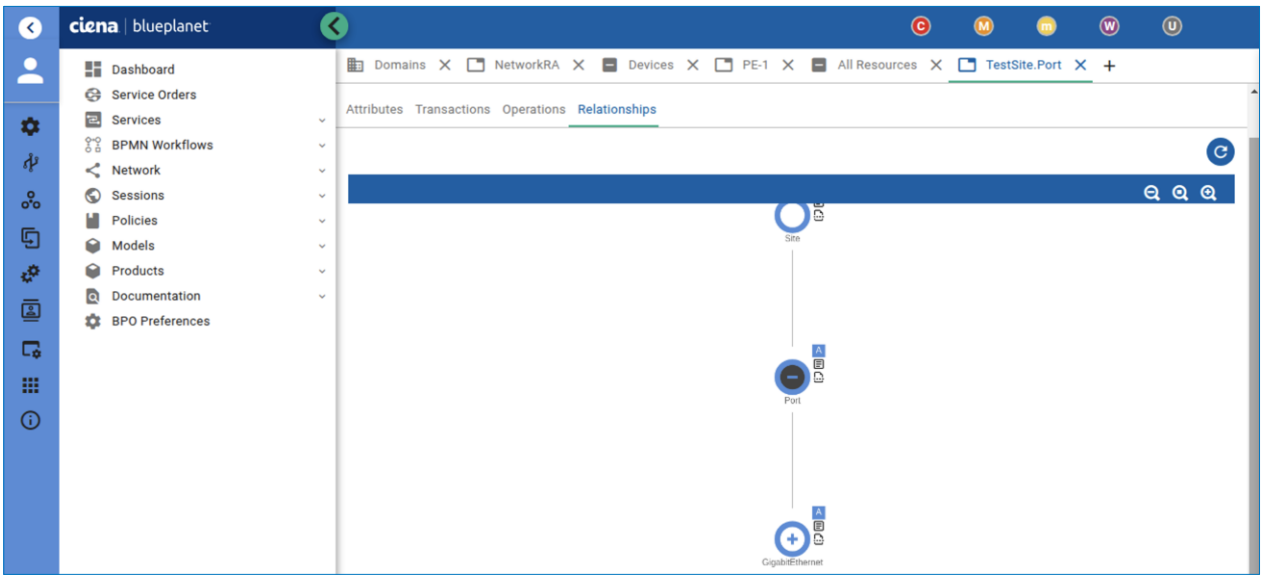
```
                        'portSpeed': properties["portSpeed"]},
            "desiredOrchState": "requested",
            "orchState": "requested"
        })

        # Patch the desired operational state
        self.bpo.market.patch(f"/resources/{port_resource['id']}", {
            "desiredOrchState": "active"
        })

        # Create a relationship between the Site and the Port
        self.bpo.market.post("/relationships", {
            "relationshipTypeId": "tosca.relationshipTypes.DependsOn",
            "sourceId": resource['id'],
            "targetId": port_resource['id'],
            "requirementName": "composed",
            "capabilityName": "composable"
        })

        return port_resource
```

l. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

m. Create a new Site resource.



n. Inspect the Port resource again. This time, you notice that a relationship has been formed with the Site resource as well.



o. Delete the Site resource. Both resources should be deleted.

Task 3: Add Support for Multiple Sites in the L2VPN Resource

In this task, you further improve your L2VPN resource by adding support for multiple sites to be added to your L2VPN service. This way, you are one step closer to fully implementing your first BPO service.

1. Update the L2VPN resource type:
   a. Open the **resource_type_l2vpn.tosca** file in the **bpo-sdd/model-definitions/types/tosca/training** folder.
   b. Change the **siteLabel** property to **sites** property and the **type** to **array.** For now, you will add support for two sites (forming a site-to-site VPN)**.** Set both **minItems** and **maxItems** number to 2. Adjust the description accordingly.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)"  ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {

      vlanId {
        title = "VLAN Identifier"
        description = "Specify the VLAN to be used by the L2VPN"
        type = integer
        optional = true
        updatable = true
        minimum = 2
        maximum = 4095
      }

      customerLabel {
        title = "Customer Label"
        description = "Label of the customer"
        type = string
      }

      sites {
        title = "Site list"
        description = "Array of sites to be used for the service creation"
        type = array
        minItems = 2
        maxItems = 2
      }

      bandwidthProfile {
        title = "Bandwidth Profile"
        description = "Label of the bandwidth profile"
        type = string
      }
    }
```

2. While changing a string to an array is not backward incompatible, renaming a property is. You learn to handle backward compatible/incompatible changes in the next lab. For now, on board the resource type change to BPO:
    - Delete all the L2VPN resources in BPO. If some fail to delete, you manually patch their orchestration and desired orchestration state to terminated over the Swagger Market API.
    - Delete the L2VPN product from BPO.
    - Onboard the changes to BPO.
    - Re-create the L2VPN product.
3. Update your imperative plan code to support multiple sites in a L2VPN resource. This entails the following actions:
    - Verify all input sites exist in the ValidateActivate class.
    - Make sure a relationship forms between L2VPN and each site.
    - Make sure that a managed VLAN resource is created in all sites (devices belonging to that site).
    - Make sure that a managed Policy Map is created on all sites as well.
    - Make sure that a managed logical port is created on the correct site.
    - Delete all relationships with sites when deleting the L2VPN resource.
4. Add an additional constraint where no two Sites that belong to the same L2VPN resource can use a port that belongs to the same managed Device. Implement the support for this functionality as a challenge to yourself if you finish with this set of labs early.
5. Improve the L2VPN resource validation in the **ValidateActivate** class:
    a. Loop through all the input sites and make sure that they all exist as Site resources. As with much of this lab, most of the code can be reused from a single site functionality. It just needs to loop through multiple sites instead of one now.

```
class ValidateActivate(Plan):
    """
    Verify that BandwidthProfile resource exists
    Verify that Customer resource exists
    Verify that all Site resources exist
    Verify that a L2VPN resource does not exist with the same label
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs['resource']
        properties = resource['properties']

        try:
            customer = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Customer",
                q_params={"label": properties['customerLabel']}
            )
        except:
            return failure(f"Customer with label ({properties['customerLabel']}) does not exist")

        try:
            bp = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.BandwidthProfile",
                q_params={"label": properties['bandwidthProfile']}
            )
        except:
            return failure(f"BandwidthProfile with label ({properties['bandwidthProfile']}) does not
exist")

        # Verify that all Site resources exist
```

```
        for site_label in properties['sites']:
            try:
                site = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": site_label}
                )
            except:
                return failure(f"Site with label ({site_label}) does not exist")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

b. Add a snippet of code that will fail the validation process if two or more sites use the same managed device underneath.

```
class ValidateActivate(Plan):
    """
    Verify that BandwidthProfile resource exists
    Verify that Customer resource exists
    Verify that all Site resources exist
    Verify that a L2VPN resource does not exist with the same label
    """
    def run(self):
        log.info(f"ValidateActivate: Input params: {self.params}")
        inputs = self.params['inputs']
        resource = inputs['resource']
        properties = resource['properties']

        try:
            customer = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Customer",
                q_params={"label": properties['customerLabel']}
            )
        except:
            return failure(f"Customer with label ({properties['customerLabel']}) does not exist")

        try:
            bp = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.BandwidthProfile",
                q_params={"label": properties['bandwidthProfile']}
            )
        except:
            return failure(f"BandwidthProfile with label ({properties['bandwidthProfile']}) does not
exist")

        # Verify that all Site resources exist
        sites = []
        for site_label in properties['sites']:
            try:
                site = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": site_label}
                )
            except:
                return failure(f"Site with label ({site_label}) does not exist")
            sites.append(site)

        for site in sites:
            for compare in sites:
                if site['properties']['deviceName'] == compare['properties']['deviceName'] and
site['label'] != compare['label']:
                    return failure(f"Site {site['label']} and {compare['label']} use the same device -
{site['properties']['deviceName']}!")

        log.info("ValidateActivate: DONE")
        return success("Validation successful")
```

6. Add relationships with all site resources during the L2VPN activation process. Navigate to the **Activate** class and implement this functionality.

```
class Activate(Plan):
    """
    Create L2VPN resource into market
    Create relationship with Customer, Sites, and Bandwidth Profile resource
    Allocate VLAN
    Create managed VLAN resource
    Create managed logical port resource
    Create managed Policy Map resource
    """
    def run(self):
        log.info(f"Activate: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"Activate: resourceId {resource_id}")
        log.info(f"Resource: {resource}")

        properties = resource['properties']
        self.assign_customer(properties)
        log.info(f"Assign Customer resource: {properties['customerLabel']}")

        for site_label in properties['sites']:
            self.assign_site(site_label)
            log.info(f"Assign Site resource: {site_label}")

        self.assign_bandwidthprofile(properties)
        log.info(f"Assign Bandwidth Profile resource: {properties['bandwidthProfile']}")

...

    def assign_site(self, site_label):
        resource_id = self.params['resourceId']
        site = self.bpo.resources.get_one_by_filters(
                resource_type="training.resourceTypes.Site",
                q_params={"label": site_label}
        )
        self.bpo.relationships.add_relationship(resource_id, site['id'])
...
```

7. Improve the **create_managed_vlan_resource()** method to create a managed VLAN resource for all the sites instead of just one.

```
    def create_managed_vlan_resource(self, resource, allocated_vlan):
        properties = resource['properties']

        # Create a managed VLAN on managed device in each Site
        for site_label in properties['sites']:
            # Read the L2VPN Port resource
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )

            # Read the L2VPN Port Managed Device resource
            device_name = port_resource["properties"]["deviceName"]
            managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)

            # Create a Managed VLAN resource
            managed_vlan_product =
self.bpo.market.get_products_by_resource_type('radevsim.resourceTypes.Vlan')[0]['id']
            try:
                managed_vlan = self.bpo.resources.create(managed_device_resource["id"], {
                    'productId': managed_vlan_product,
```

```
                    'label': f"{device_name}.{resource['label']}.VLAN",
                    'properties': {
                        "id":  allocated_vlan,
                        "name": f"{device_name}.{resource['label']}.VLAN",
                        "device": managed_device_resource["id"]
                    }
                })
                self.bpo.relationships.add_relationship(resource['id'], managed_vlan[0]['id'])
            except Exception as e:
                raise Exception(f"Failed to create VLAN with ID {allocated_vlan}. Make sure that it is
available on device!")
```

8. Update the **create_managed_policy_map()** to support this functionality.

```
    def create_managed_policy_map(self, resource):
        properties = resource['properties']

        # Read the L2VPN BandwidthProfile resource
        bwp_resource = self.bpo.resources.get_one_with_filters(
            ExactTypeId("training.resourceTypes.BandwidthProfile"),
            QQuery("label", properties['bandwidthProfile'])
        )

        # Create a managed Policy Map on managed device in each Site
        for site_label in properties['sites']:
            # Read the Managed Device resource for each L2VPN Site
            site_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Site"),
                QQuery("label", site_label)
            )

            # Read the L2VPN Site Managed Device resource
            site_device = site_resource["properties"]["deviceName"]
            managed_device_resource = PortUtils.get_managed_device_resource(self, site_device)

            # Create a Managed Policy Map resource
            managed_policy_map_product =
self.bpo.market.get_products_by_resource_type('radevsim.resourceTypes.PolicyMap')[0]['id']
            try:
                managed_policy_map = self.bpo.resources.create(managed_device_resource["id"], {
                    'productId': managed_policy_map_product,
                    'label': f"{site_device}.{resource['label']}.PolicyMap",
                    'properties': {
                        "name": f"{site_device}.{resource['label']}.PolicyMap",
                        "device": managed_device_resource["id"],
                        "police": {
                            "bc": bwp_resource['properties']['cbs'],
                            "be": bwp_resource['properties']['ebs'],
                            "cir": bwp_resource['properties']['cir'],
                            "pir": bwp_resource['properties']['eir']
                        },
                        "class": [{
                            "name": f"{resource['label']}-policy-class"
                        }]
                    }
                })
                self.bpo.relationships.add_relationship(resource['id'], managed_policy_map[0]['id'])
            except:
                raise Exception(f"Failed to create Policy Map
{site_device}.{resource['label']}.PolicyMap.")
```

9. Do the same for the **create_managed_logical_port()**. It must support creating a logical port on a managed device for each site.

```
    def create_managed_logical_port(self, resource, allocated_vlan):
        properties = resource['properties']

        # Create a managed logical port on managed device in each Site
        for site_label in properties['sites']:
            # Read the L2VPN Port resource
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )

            # Read the L2VPN Port Managed Device and port resource
            device_name = port_resource["properties"]["deviceName"]
            managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
            managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])

            # Create managed logical port resource
            logical_port_product_id = managed_port_resource['productId']
            managed_logical_port_resource = self.bpo.resources.create(
                managed_port_resource['id'], {
                'productId': logical_port_product_id,
                'label': f"{managed_port_resource['label']}.{allocated_vlan}",
                'properties': {
                    "name":  f"{managed_port_resource['label']}.{allocated_vlan}",
                    "device": managed_device_resource['id'],
                    "encapsulation": {
                        "dot1Q": {
                            "vlan-id": allocated_vlan
                        }
                    }
                }
            })
            self.bpo.relationships.add_relationship(port_resource['id'],
managed_logical_port_resource[0]['id'])
```

10. Improve the **Terminate** class. It also must support multiple sites, so delete all the managed VLANs, Policy Maps, and Logical Ports that belong to a site:
    a. Modify the **delete_managed_vlan_resources()** method.

```
    def delete_managed_vlan_resource(self, resource_id):
        try:
            managed_vlans = self.bpo.resources.get_dependencies_with_filters(
                resource_id,
                ExactTypeId("radevsim.resourceTypes.Vlan")
            )
            for managed_vlan in managed_vlans:
                self.bpo.relationships.delete_source_relationships(managed_vlan['id'])
                self.bpo.relationships.delete_relationships(managed_vlan['id'])
                self.bpo.resources.delete(managed_vlan['id'])
                self.bpo.resources.await_termination(managed_vlan["id"], managed_vlan["label"], True)
                log.info(f"Managed VLAN {managed_vlan['label']} deleted.")
        except: Exception as e
            raise Exception(f"Could not delete managed VLAN: {e}")
```

b. Update the **delete_managed_policy_map_resources()** method.

```
       def delete_managed_policy_map_resource(self, resource_id):
        try:
            managed_policy_maps = self.bpo.resources.get_dependencies_with_filters(
                resource_id,
                ExactTypeId("radevsim.resourceTypes.PolicyMap")
            )
            for managed_policy_map in managed_policy_maps:
                self.bpo.relationships.delete_source_relationships(managed_policy_map['id'])
                self.bpo.relationships.delete_relationships(managed_policy_map['id'])
                self.bpo.resources.delete(managed_policy_map['id'])
                self.bpo.resources.await_termination(managed_policy_map["id"], managed_policy_map["label"],
True)
            log.info(f"Managed Policy Map {managed_policy_map['label']} deleted.")
        except: Exception as e:
            raise Exception(f"Could not delete managed Policy Map: {e}")
```

c. Update the **delete_managed_logical_port_resource()** method.

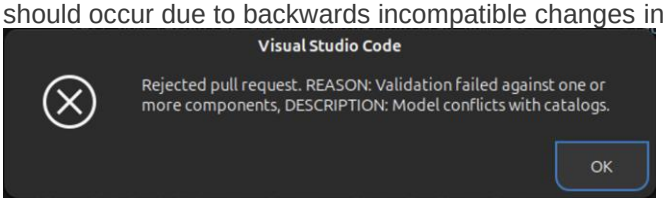
```
    def delete_managed_logical_port_resource(self, resource_id):
        try:
            site_resources = self.bpo.resources.get_dependencies_with_filters(resource_id,
ExactTypeId("training.resourceTypes.Site"))
            for site_resource in site_resources:
                port_resource = self.bpo.resources.get_dependency_with_filters(site_resource['id'],
ExactTypeId("training.resourceTypes.Port"))
                managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])
                managed_logical_port_resources =
self.bpo.resources.get_dependencies_by_type(port_resource['id'], managed_port_resource['resourceTypeId'])
                managed_logical_port_resource = [port for port in managed_logical_port_resources if "." in
port['label']][0]
                self.bpo.relationships.delete_source_relationships(managed_logical_port_resource['id'])
                self.bpo.relationships.delete_relationships(managed_logical_port_resource['id'])
                self.bpo.resources.delete(managed_logical_port_resource['id'])
                self.bpo.resources.await_termination(managed_logical_port_resource['id'],
managed_logical_port_resource['label'], True)
                log.info(f"Managed Logical Port {managed_logical_port_resource['label']} deleted.")
        except: Exception as e:
            raise Exception(f"Could not delete managed Logical Port: {e}")
```

11. Take care of the Update lifecycle operation. Since the operation updates the allocated VLAN, update all the managed VLAN resources. Because this is done via the **create_managed_vlan_resource** and the **delete_managed_vlan_resource** methods, and you have already updated those, no action is needed for this functionality.
12. Update the ChangePort operation. Here, ensure that the newly selected managed device is not already attached to any of the sites belonging to the current L2VPN instance.

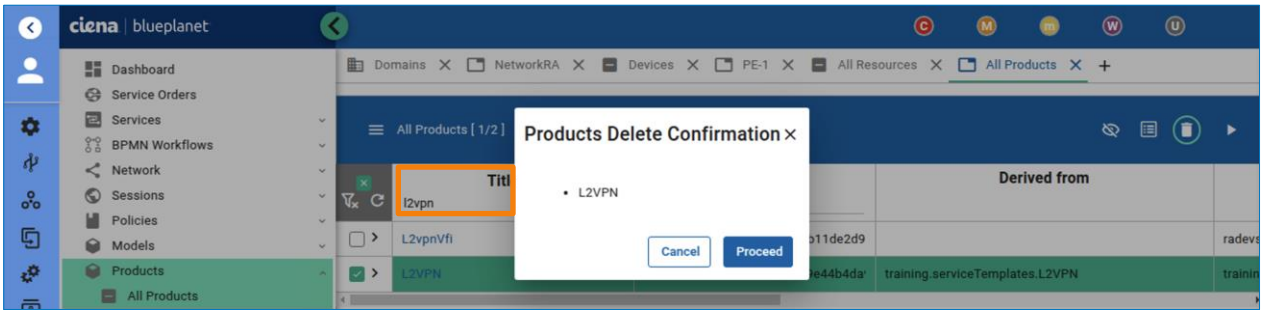
```
class ChangePort(Plan):
    """
    Changes the Port and Device attached to selected Site in L2VPN resource
    Updates the managed VLAN resource
    Updates the managed logical port resource
    """
    def run(self):
        log.info(f"ChangePort: Input params: {self.params}")

        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"ChangePort: resourceId {resource_id}")
        log.info(f"ChangePort: {resource}")

        operation = self.bpo.resources.get_operation(resource_id, self.params['operationId'])
        inputs = operation['inputs']
```

```
        log.info(f"ChangePort: inputs {inputs}")
        site_label = inputs['siteLabel']

        # Verify that the Site exists
        try:
            site_res = self.bpo.resources.get_one_by_filters(
                    resource_type="training.resourceTypes.Site",
                    q_params={"label": f"{site_label}"}
            )
            log.info(f"Site resource for {site_label} found.")
        except:
            raise Exception(f"Site resource for {site_label} not found.")

        # Verify that each site has its own device
        sites = self.bpo.resources.get_dependencies_with_filters(
            resource_id,
            ExactTypeId("training.resourceTypes.Site")
        )
        for site in sites:
            if inputs['deviceName'] == site['properties']['deviceName']:
                raise Exception(f"Cannot use device {inputs['deviceName']} in site {inputs['siteLabel']}
since it is already used in {site['label']}!")

        # Find the Port resource attached to this Site
        try:
            port_res = self.bpo.resources.get_dependency_by_type(
                site_res["id"], "training.resourceTypes.Port"
            )
            log.info(f"Port resource for {site_label} found - {port_res}")
        except:
            raise Exception(f"Port resource for {site_label} not found!")
```

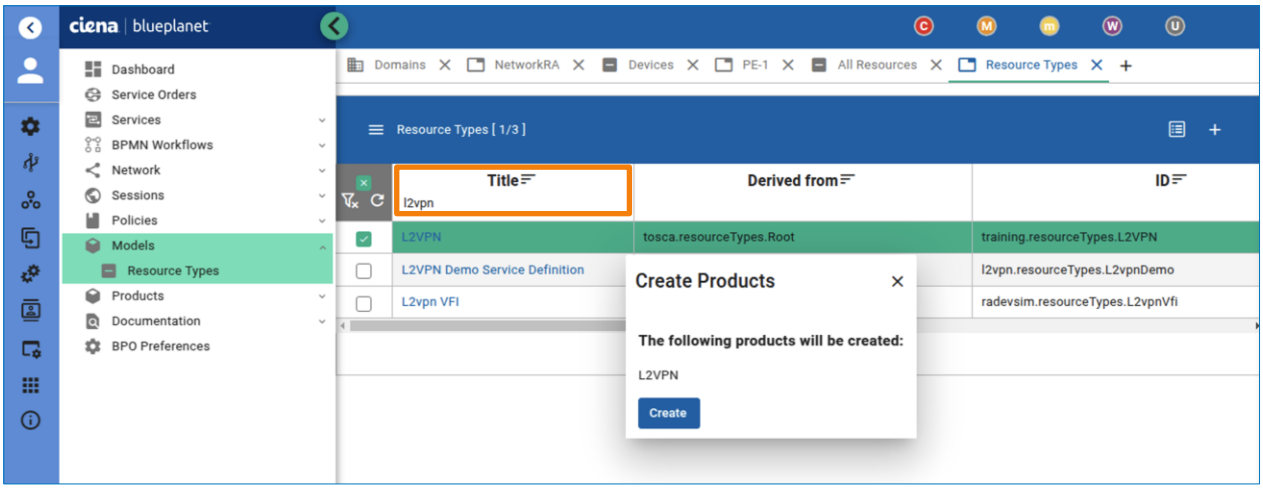
13. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension. Model conflict should occur due to backwards incompatible changes in training.resourceTypes.L2VPN.



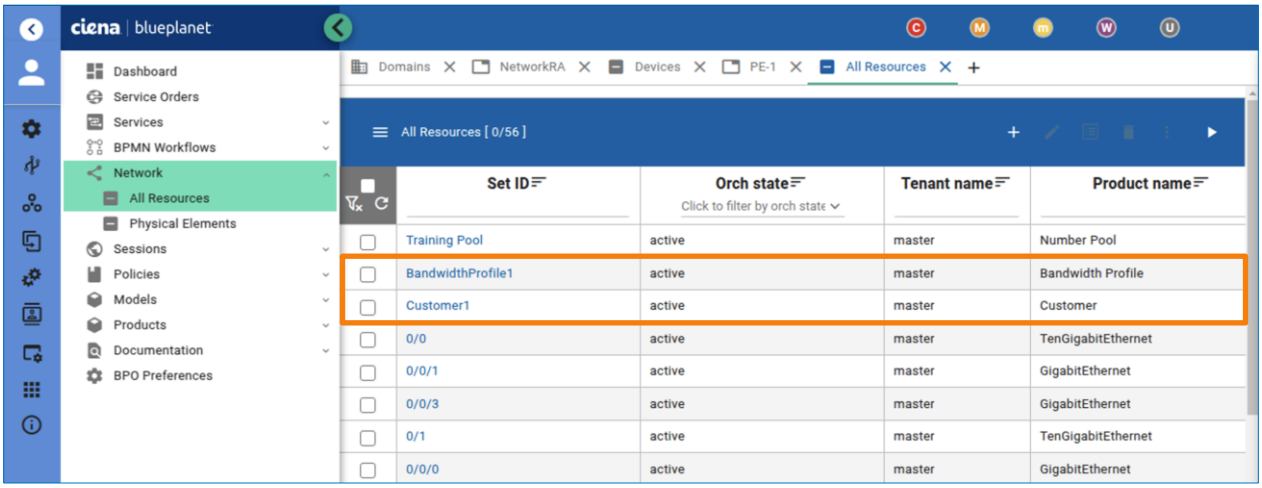
14. Delete the L2VPN product by going under **Products > All Products,** choosing the L2VPN Product and deleting it.



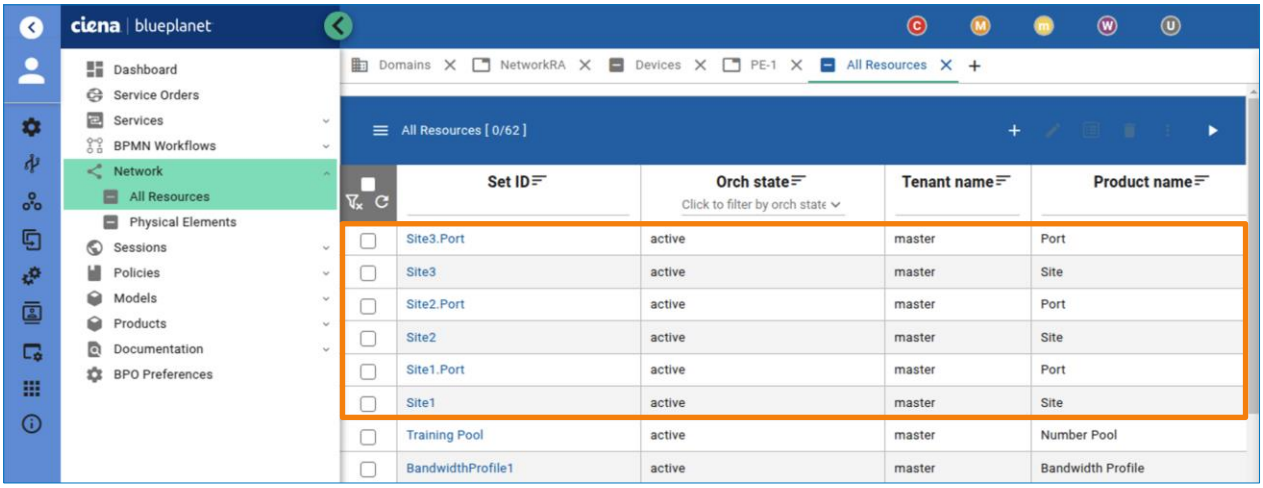
15. **Onboard** the changes to the BPO server again. This time, the changes should be onboarded with no issues. If there are any, troubleshoot and fix them.
16. Re-create the L2VPN product by going under **Models > Resource Types**, choosing the L2VPN resource type, and clicking the **+** symbol in the upper right corner of the screen to create the product.



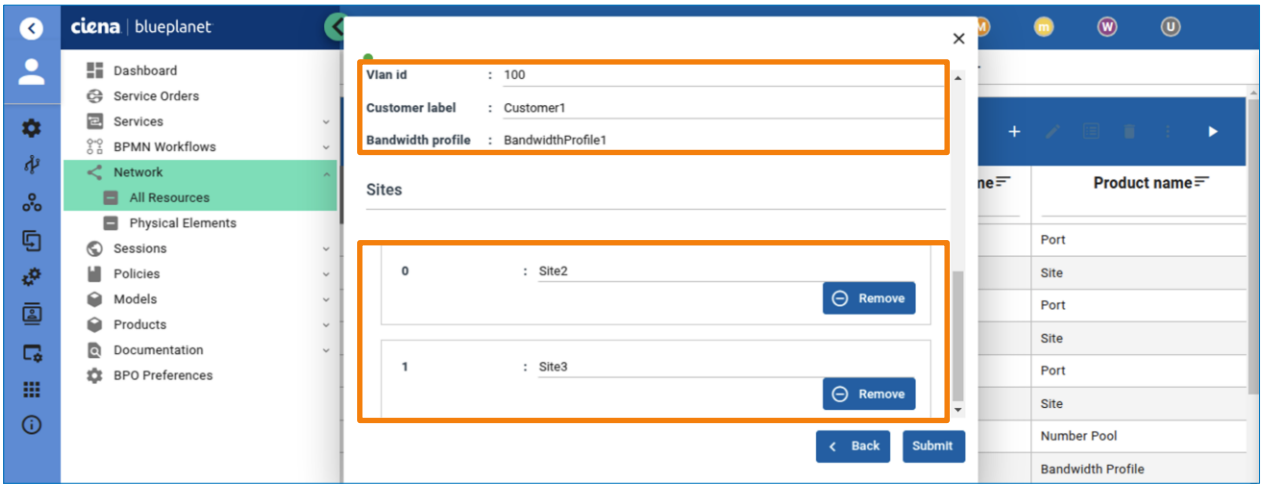
17. Delete the **Site1** resource. Make sure that you have Customer and Bandwidth Profile resource.



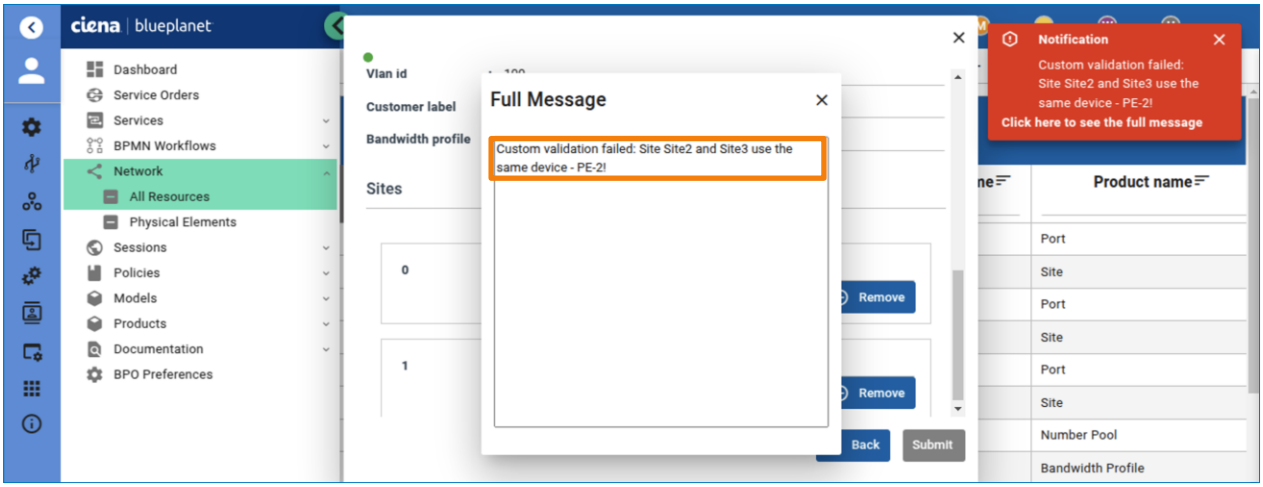
18. Create at least three Site resources. Two of them should have the same managed device used:
    - **Site1** (Port name : 0/0/0, Port speed: 1Gbps, Device name : PE-1)
    - **Site2** (Port name : 0/0/0, Port speed: 1Gbps, Device name : PE-2)
    - **Site3** (Port name : 0/0/1, Port speed: 1Gbps, Device name : PE-2)



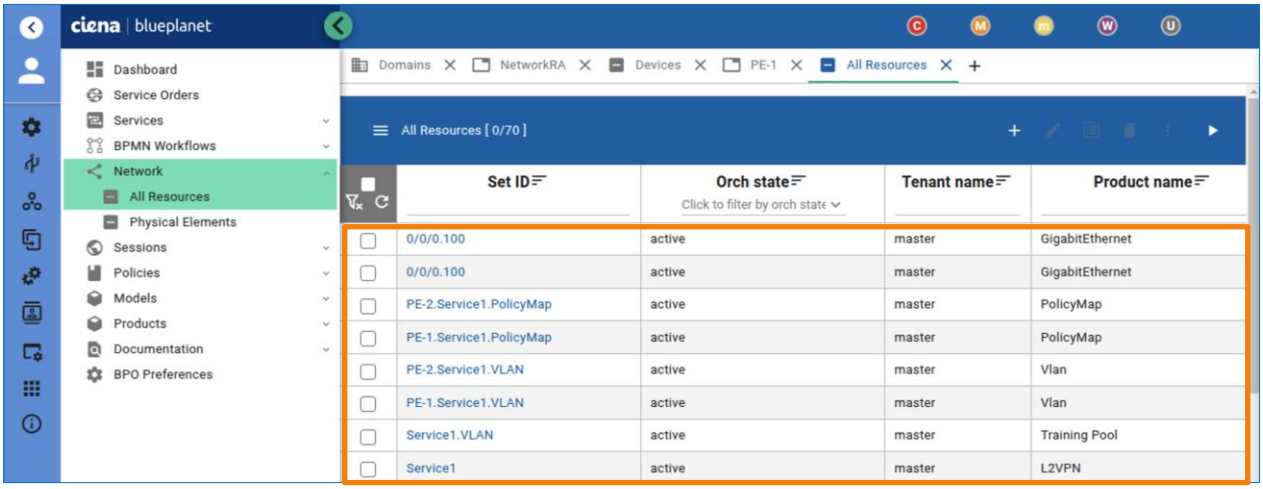
19. Try out your new multi-site supported L2VPN. First, create a L2VPN resource where two sites share the same device (Site2 and Site3).



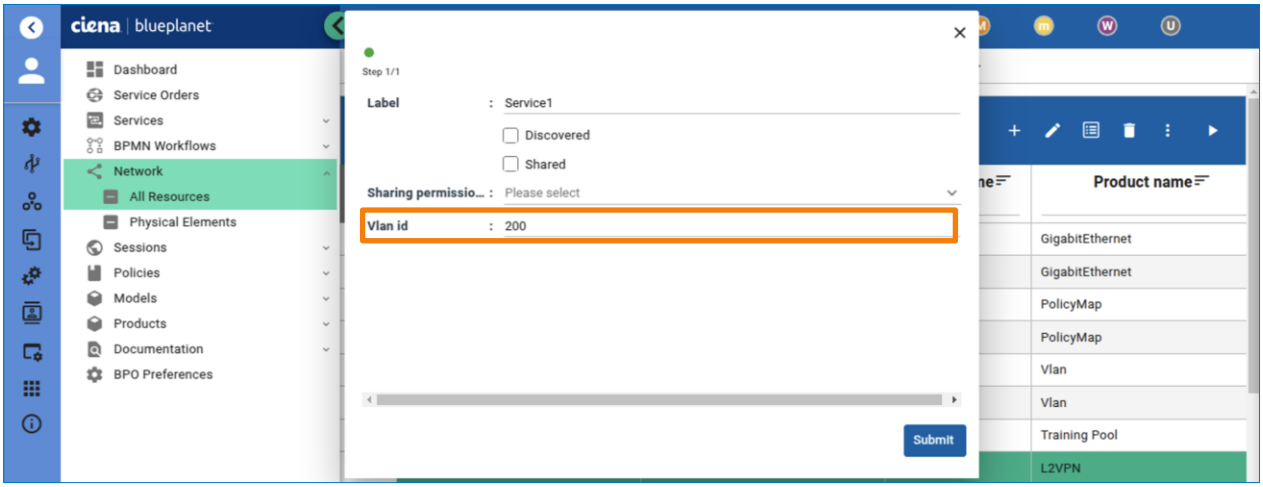
20. This causes an intended validation failure.



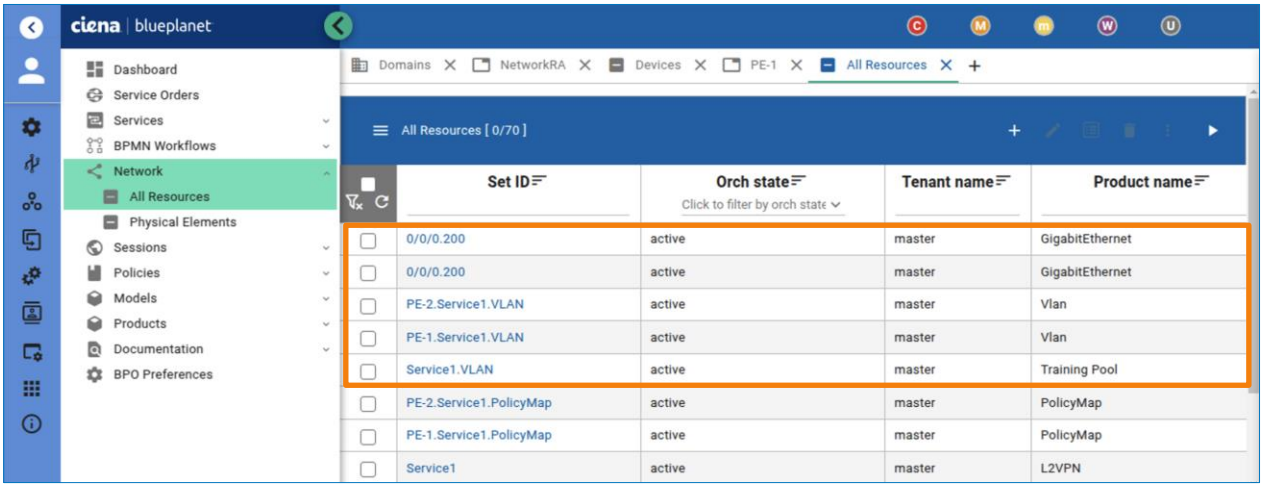
21. Create a L2VPN which includes Site1 and Site2. Multiple managed VLAN, Policy Map, and managed logical port resources now exist – one for each site attached to your L2VPN resource.



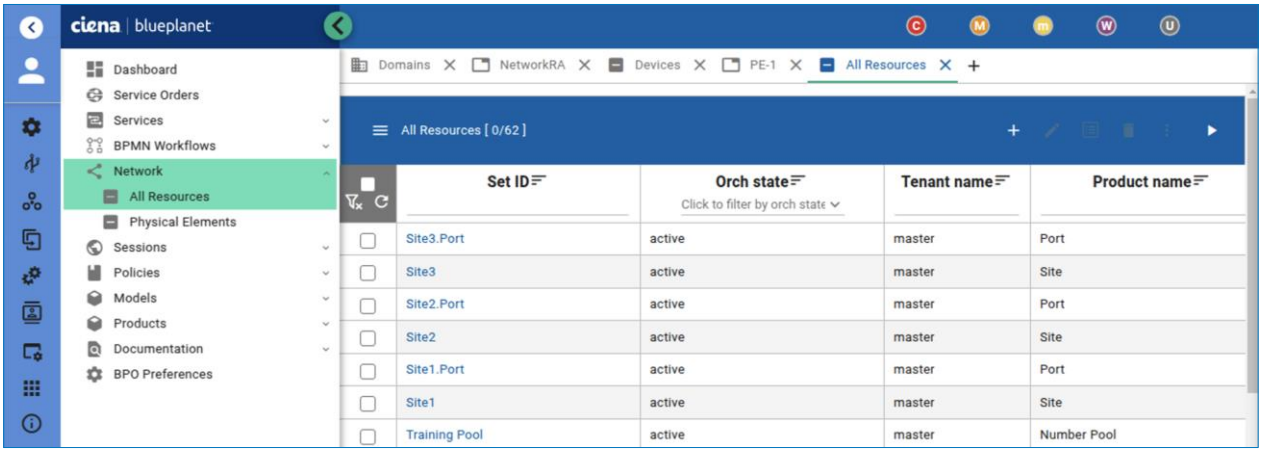
22. Update your L2VPN service by setting a new and different VLAN.



23. Notice that the managed VLAN and managed logical port resources have been recreated with updated VLAN values.



24. Check the updated versions on the simulated dev-sim devices.
25. Delete Service1 resource used in this lab.



End of Lab

Lab 7: Update a Service Model

Objectives

- Add, remove, and update properties
- Finalize the implementation of the L2VPN service
- Add a new custom operation for checking service status

Documentation

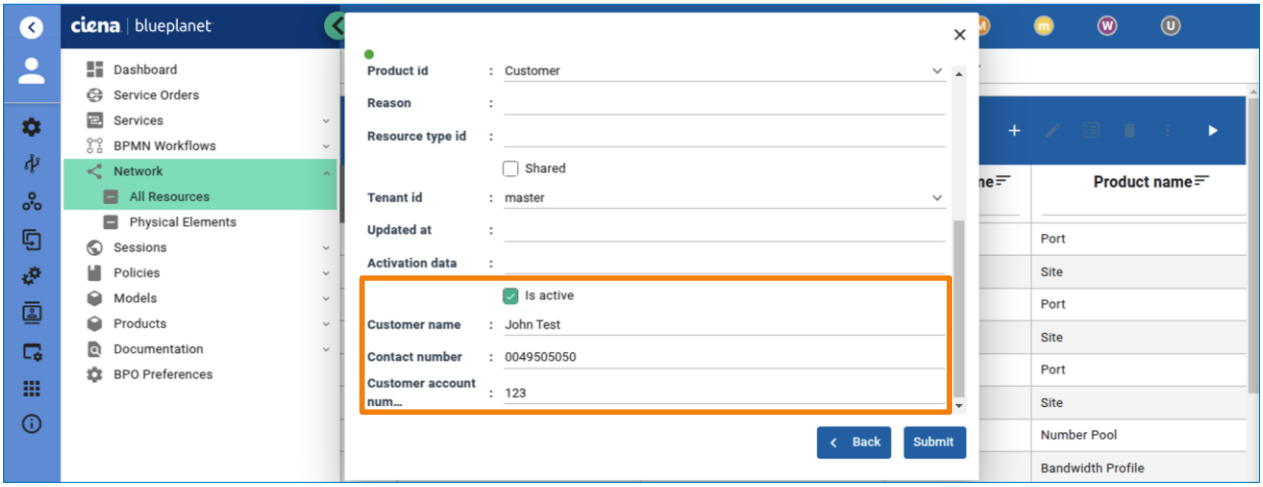
PlanSDK documentation - *https://developer.blueplanet.com/docs/plansdk/index.html*

Resource Definition changes - *https://developer.blueplanet.com/docs/bpocore-docs/references/onboarding.html#_definition_module_changes*

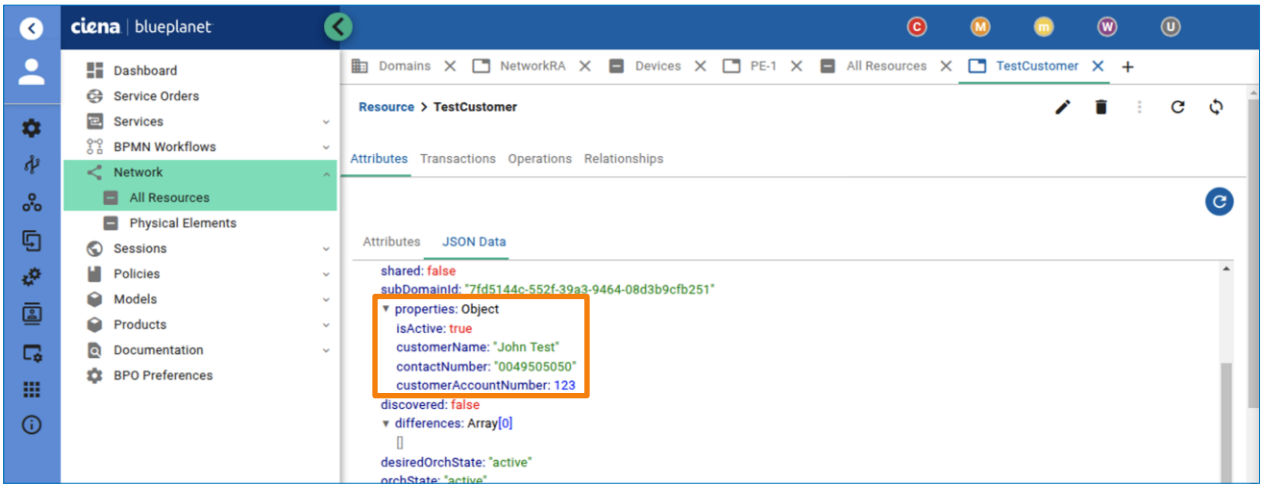
Task 1: Add, Remove, and Update Properties

In software development and service modeling, changes are expected. Even if your resources and services are designed perfectly to fit your current and future needs, it can always happen that the service structure needs to be updated and that the resource properties need to be updated, ideally while preserving existing products and resources. This is why you need to know what changes are supported, what the caveats are, and what types of changes are backward incompatible. In this task, you delete, update, and add properties for the Customer resource type.

1. Open BPO UI. Navigate to the **Network > All Resources** tab. Create a new Customer resource by clicking on the **+** symbol in the upper right corner. Throughout this task, you update the Customer resource type and use this customer resource to verify resource type changes.
2. Set the Customer label to **TestCustomer**, check the **Is Active** checkmark, and set the **Customer name**, **Contact number**, and **Customer account number** property values to any arbitrary values. All values here are mandatory since the **optional = true** property is not set in the resource type definition.



3. When the Customer resource is created, open and inspect the newly created resource. Open the **JSON Data** tab to see the resource properties. Note them down for future reference and comparison.



4. In VS Code, open the Customer resource type definition file **resource_type_customer.tosca** in the **bpo-sdd/model-definitions/types/tosca/training** folder. The file should look like the following output.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer resource type."
authors     = [ "Developer (developer@bptrn.com)"   ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {

  Customer {
    derivedFrom = Root
    title = "Customer"
    description = "The Customer resource is used to define customer information"

    properties {

      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as an integer"
        type = integer
      }

      contactNumber {
        title = "Contact Phone Number"
        description = "Contact phone number of the customer"
        type = string
      }

      isActive {
        title = "Active"
        description = "Whether the Customer is active or not"
```

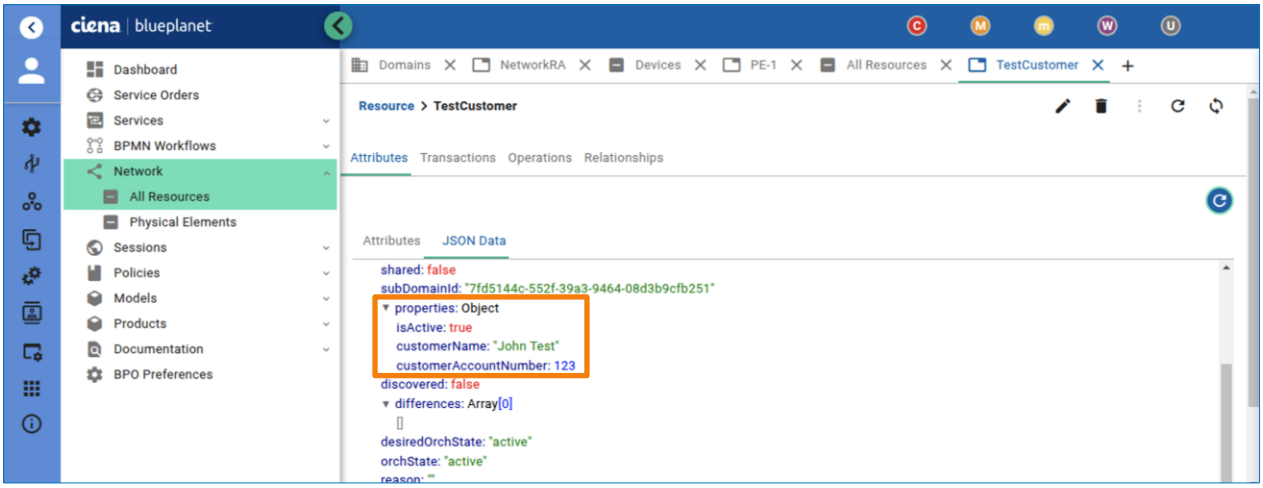
```
        type = boolean
        updatable = true
        optional = true
      }

    }
  }
}
```

5. Now imagine that due to privacy reasons, the Customer contact number property can no longer be stored in the resource definition. Your job is to remove that property without deleting any existing Customer contacts.
   a. In the **resource_type_customer.tosca** file, delete the **contactNumber** property. The file should look like the following output.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer resource type."
authors     = [ "Developer (developer@bptrn.com)"   ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {

  Customer {
    derivedFrom = Root
    title = "Customer"
    description = "The Customer resource is used to define customer information"

    properties {

      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as an integer"
        type = integer
      }

      isActive {
        title = "Active"
        description = "Whether the Customer is active or not"
        type = boolean
        updatable = true
        optional = true
      }

    }
  }
}
```

   b. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
   c. In BPO UI, open the **Onboarding > Asset Management** tab and choose the **model-definitions** area. Your latest change should be *accepted* since deleting properties is always allowed. The validation does not check the code, so it cannot figure out if the property you are deleting is used in an imperative plan – this is up to the developer to verify.

d. Open and inspect the **TestCustomer** resource (click refresh if you have the TestCustomer tab still open) and its **JSON Data**. You see that the deleted **contactNumber** property is no longer there on your existing resource.



6. The contact number has been removed, but you still need to add a way to contact a customer. You will need to add a new property in which a customer's email address will be added.

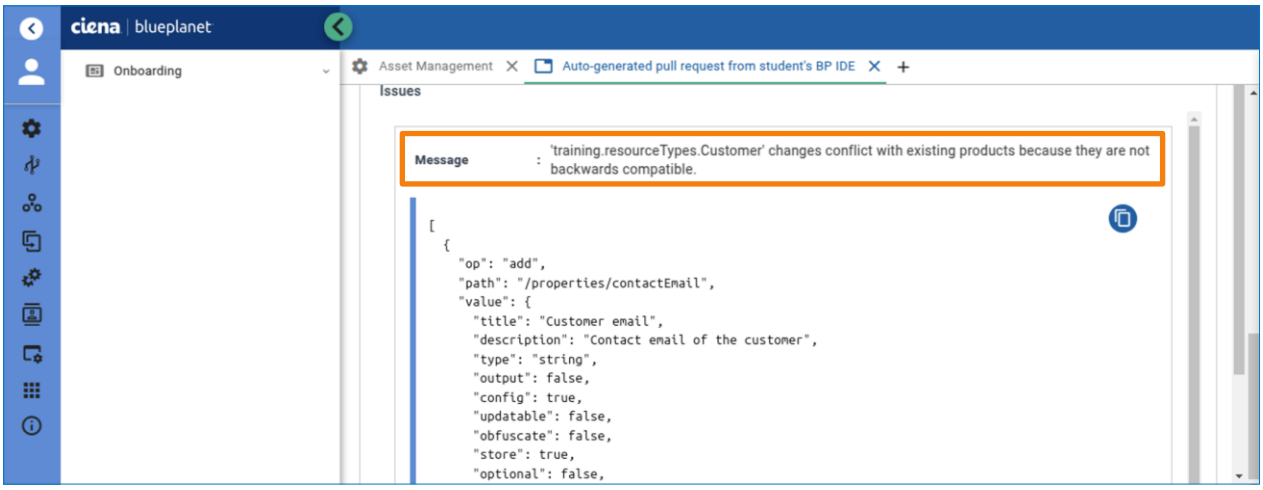
   a. Open the **resource_type_customer.tosca** file. Add a new property called **contactEmail** of the type **string**.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer resource type."
authors     = [ "Developer (developer@bptrn.com)"   ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {

  Customer {
    derivedFrom = Root
    title = "Customer"
    description = "The Customer resource is used to define customer information"

    properties {

      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as either an integer"
        type = integer
      }

      isActive {
        title = "Active"
        description = "Whether the Customer is active or not"
        type = boolean
        updatable = true
```

```
        optional = true
      }

      contactEmail {
        title = "Customer email"
        description = "Contact email of the customer"
        type = string
      }

    }
  }
}
```

b. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension. Your pull request will be rejected and you will get an error.

c. Open the Asset Manager to view the detailed errors:



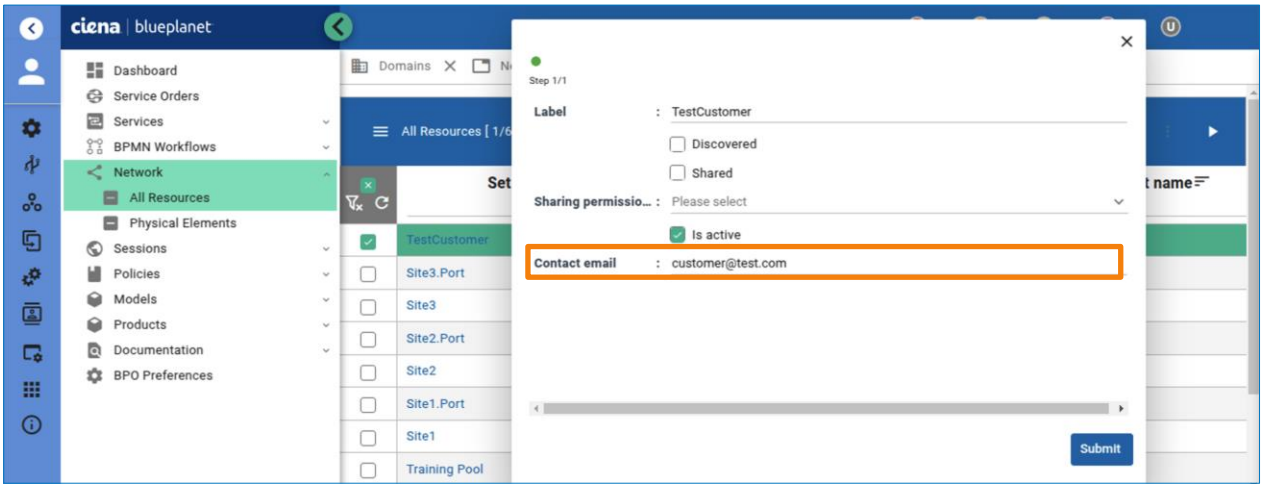
d. The action of adding a new mandatory resource property is not backward compatible. You must either set the property as optional or provide a default value. In this case, a default value does not work since you cannot have a default email address for new customers. Your only option is to set the property to **optional** and allow it to be **updatable**.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer resource type."
authors     = [ "Developer (developer@bptrn.com)"   ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {

  Customer {
    derivedFrom = Root
    title = "Customer"
    description = "The Customer resource is used to define customer information"
```

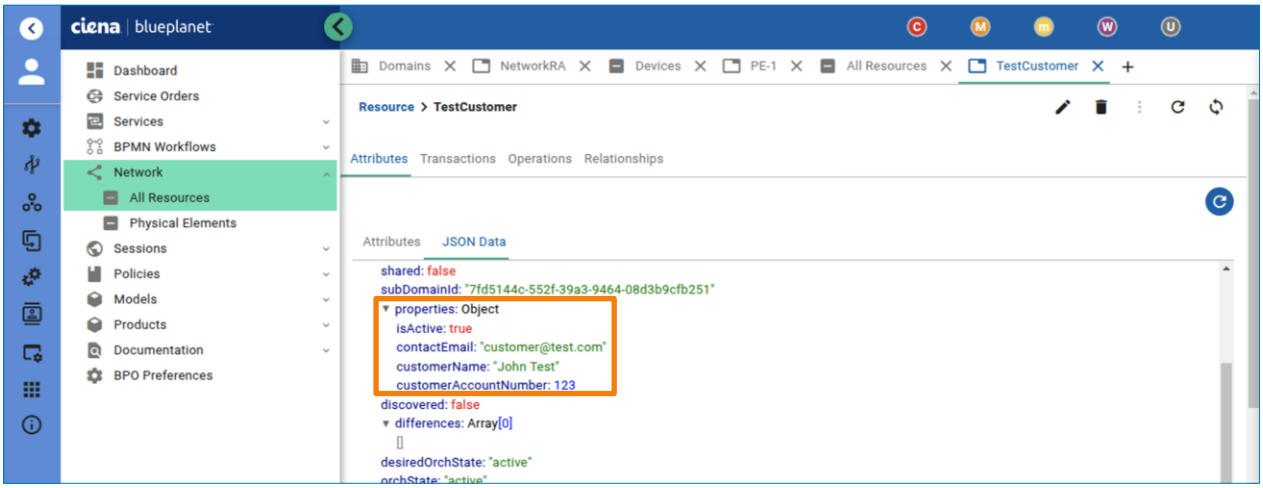
```
    properties {

      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as either an integer"
        type = integer
      }

      isActive {
        title = "Active"
        description = "Whether the Customer is active or not"
        type = boolean
        updatable = true
        optional = true
      }

      contactEmail {
        title = "Customer email"
        description = "Contact email of the customer"
        type = string
        optional = true
        updatable = true
      }

    }
  }
}
```

e. **Onboard** the changes again to the BPO server. The onboarding should be accepted now.

f. Set the email address for your **TestCustomer**. Choose the resource in BPO UI and click the **Edit** button in the upper right corner of the screen.



g. Inspect the updated resource's **JSON Data**. Find the email address that was added to this resource.



7. Next, you find out that your company is changing how the Customer accounts are handled due to your company acquiring a different company. The customer database will be merged, and each Customer can now have multiple account numbers. This means that you need to change the **customerAccountNumber** property type to support **arrays**.

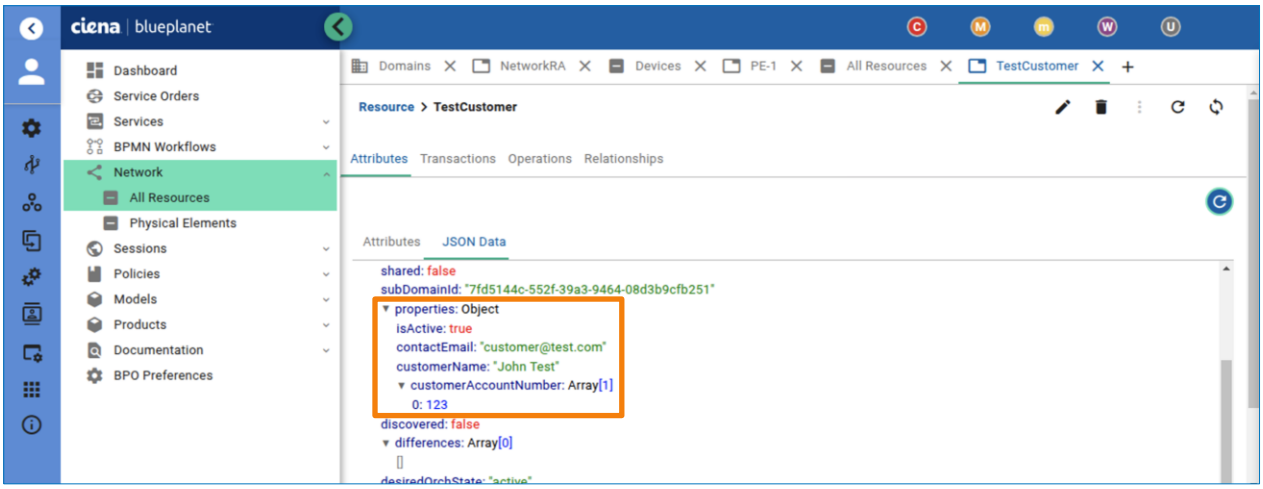
   a. Open the customer resource definition file again. You need to change the **customerAccountNumber** property so it supports multiple integer values.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer resource type."
authors     = [ "Developer (developer@bptrn.com)"   ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {

  Customer {
    derivedFrom = Root
    title = "Customer"
    description = "The Customer resource is used to define customer information"

    properties {

      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as an array of integers"
        type = array
        items = {
          type = integer
        }
        updatable = true
      }
```

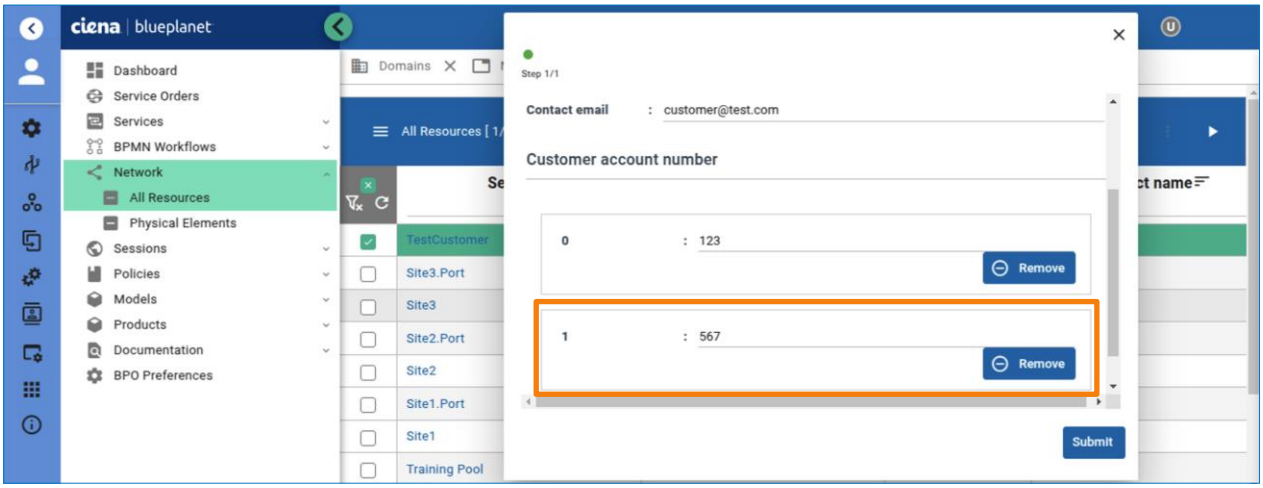
```
      isActive {
        title = "Active"
        description = "Whether the Customer is active or not"
        type = boolean
        updatable = true
        optional = true
      }

      contactEmail {
        title = "Customer email"
        description = "Contact email of the customer"
        type = string
        optional = true
        updatable = true
      }

    }
  }
}
```

b. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

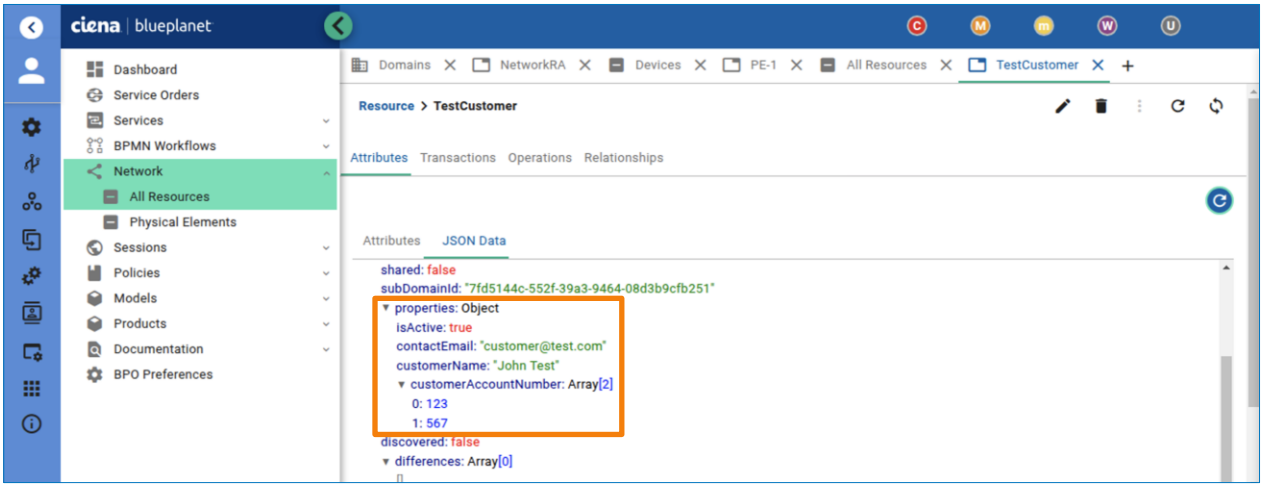
c. Inspect how the existing customer data transformed from an integer to an array of integers.



d. Add an additional customer account number to this array by editing the Customer resource.



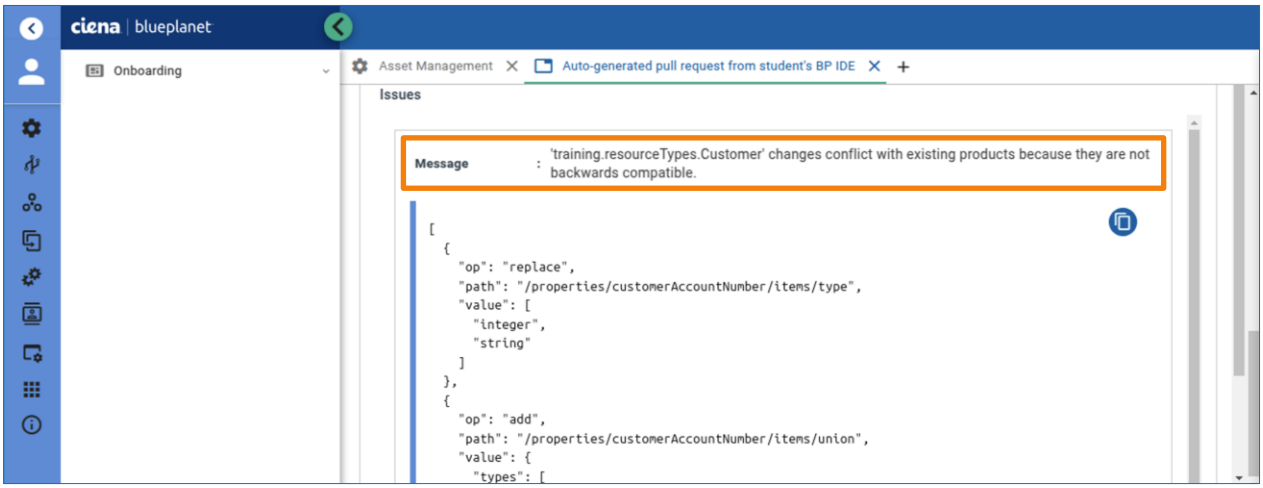
e. Inspect the Customer resource again to find the added account number.



f. Somebody from the new company reminds you that they store some of their customer account numbers in the format <two letters>-<number>, for example, AB-123, and asks you to see if this can be implemented. Create a union of integer and string types for array items.

```
    customerAccountNumber {
      title = "Customer Account Number"
      description = "Account number of the customer as an array of integers"
      type = array
      items = {
        union = [
          {
            type = integer
          },
          {
            type = string
          }
        ]
      }
      updatable = true
    }
```

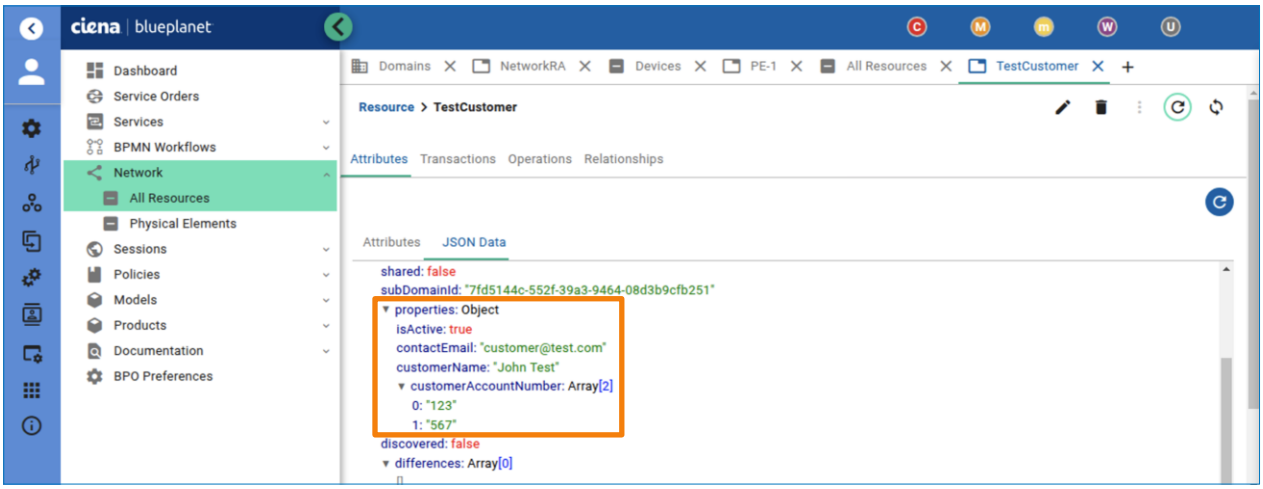
g. **Onboard** the changes to the BPO server. The onboarding will fail with the following reason:



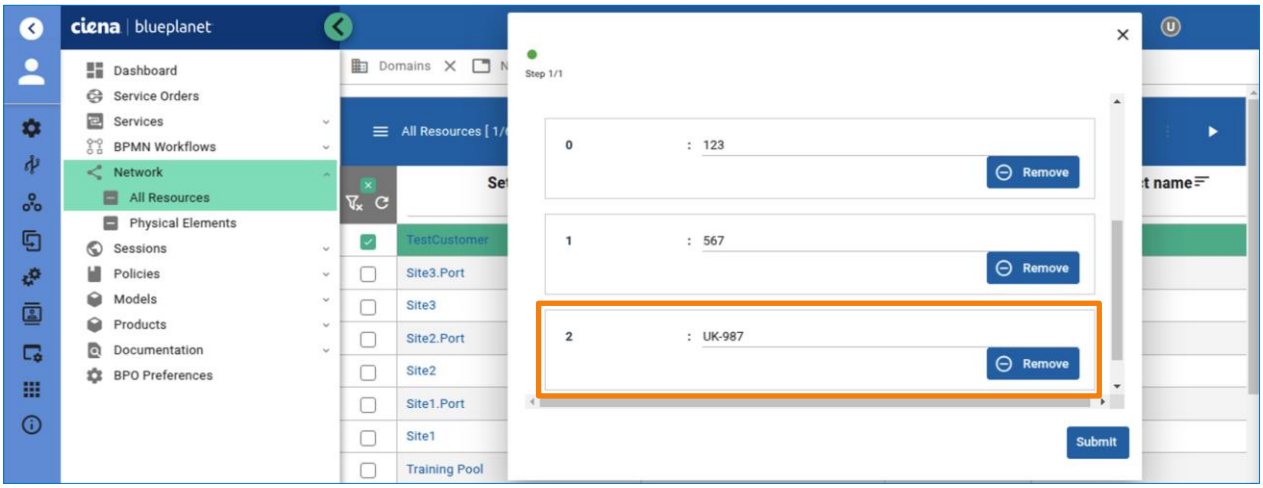
h. Such changes are unfortunately not supported. If such a change is needed, you can change the type of array items to a string.

```
customerAccountNumber {
  title = "Customer Account Number"
  description = "Account number of the customer as an array of strings"
  type = array
  items = {
    type = string
  }
  updatable = true
}
```

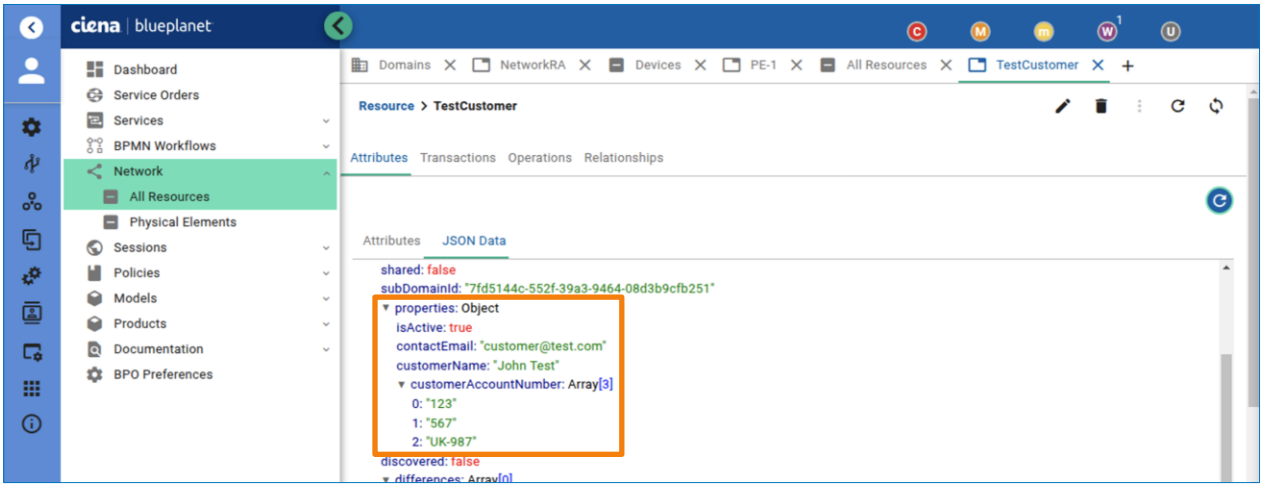
i. **Onboard** the changes to the BPO server. The onboarding should be successful. Inspect the Customer resource and notice how the integer value is now quoted, meaning it has transformed into a string.



i. Add an additional customer account number in the form desired by the new company and inspect the JSON Data again.



j. Inspect the Customer resource again to find the added account number.



NOTE: An additional improvement would be to add a patternMatcher on the array items. Such a change is backwards incompatible, though. The solution to such a requirement is to implement it as a Validation operation when you want to Activate/Update a resource. This does not validate existing customers, though.

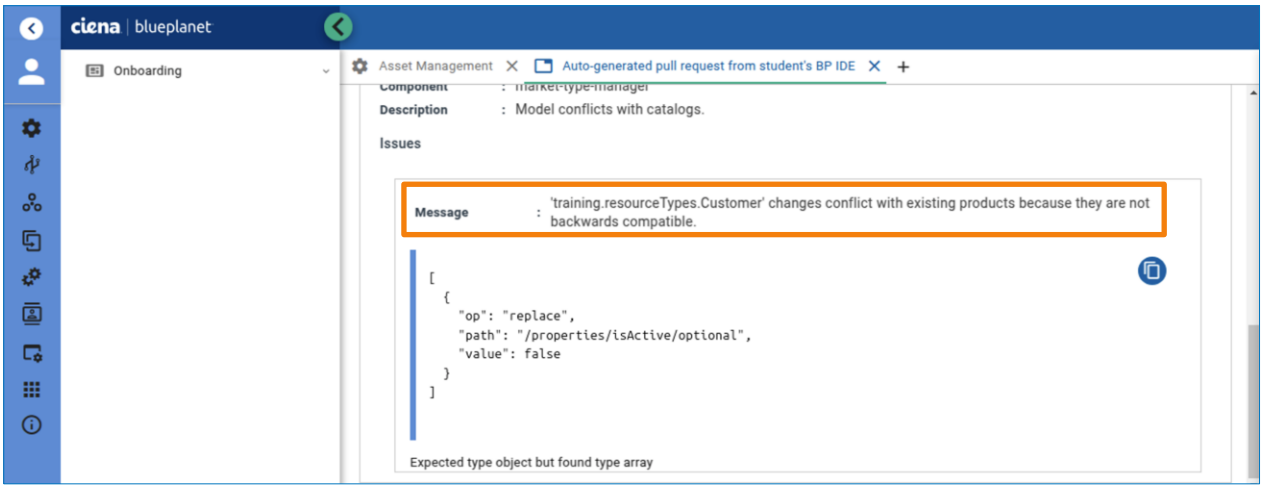
8. As a final step for updating the resources, you are told that the **isActive** property should no longer be optional since all the customers added as resources are Active.

   a. Set the **optional** parameter for the **isActive** property to **false.**

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "Customer resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the Customer resource type."
authors     = [ "Developer (developer@bptrn.com)"   ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {

  Customer {
    derivedFrom = Root
    title = "Customer"
    description = "The Customer resource is used to define customer information"

    properties {

      customerName {
        title = "Customer Name"
        description = "Name of the customer"
        type = string
      }

      customerAccountNumber {
        title = "Customer Account Number"
        description = "Account number of the customer as an array of strings"
        type = array
        items = {
          type = string
        }
        updatable = true
      }
```

```
    isActive {
      title = "Active"
      description = "Whether the Customer is active or not"
      type = boolean
      updatable = true
      optional = false
    }

    contactEmail {
      title = "Customer email"
      description = "Contact email of the customer"
      type = string
      optional = true
      updatable = true
    }

  }
 }
}
```

b. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension. You get the following error message:

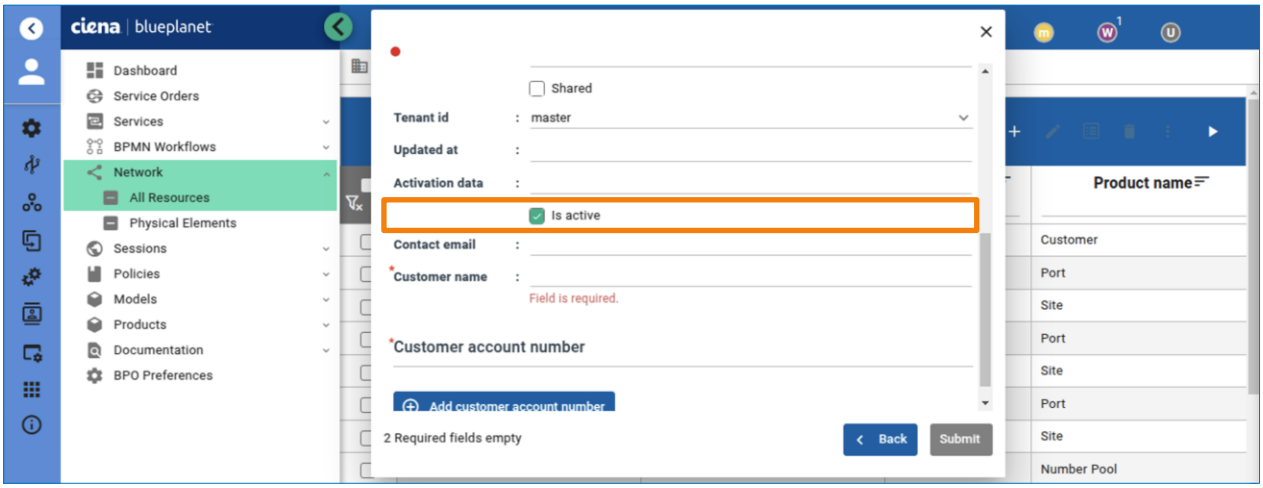


c. Since resources already exist, changing the property type to mandatory requires a default value as well. Your job is to set the default **isActive** state to Active, meaning that the default value for this property should be True. Update the resource definition file.

```
    isActive {
      title = "Active"
      description = "Whether the Customer is active or not"
      type = boolean
      updatable = true
      optional = false
      default = true
    }
```

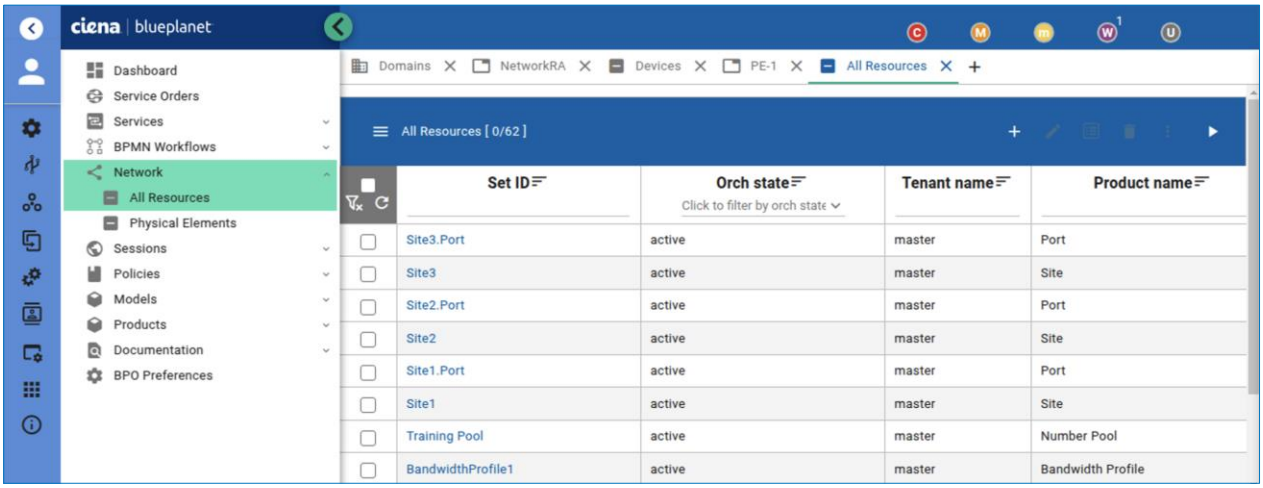
d. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

e. Try to create a new Customer resource **TestCustomer1**. You see that the *Is Active* field is already checked now.



f. The new Customer resource should be created successfully.

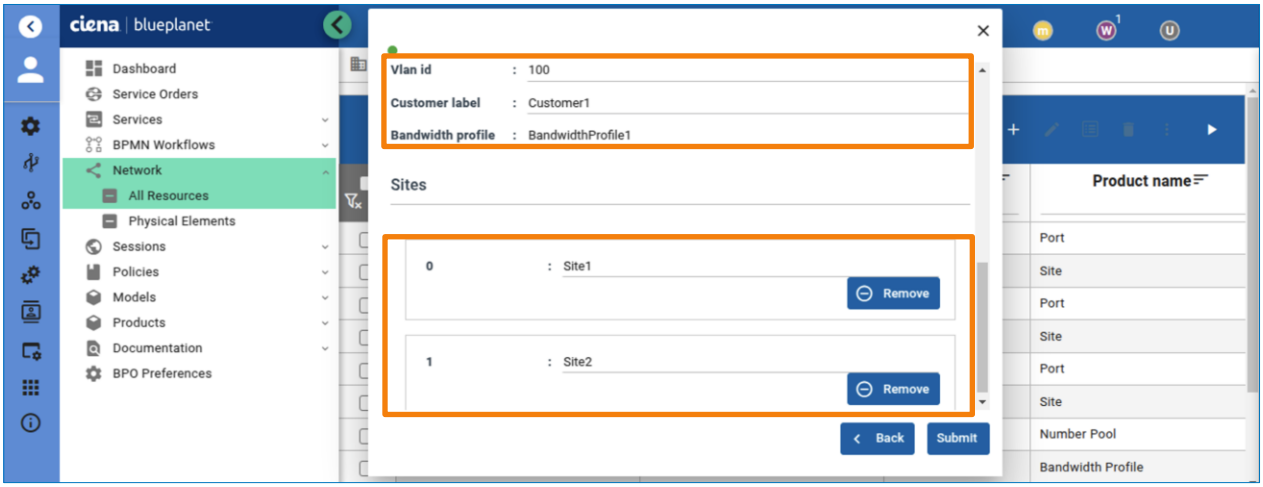
g. Delete both Customer resources when done.



Task 2: Finalize the Implementation of the L2VPN Service

This task covers the last steps needed to finalize the implementation of your L2VPN service. Until now, you have created managed logical ports, policy maps, and VLAN resources. Now, it is time to connect all of them. You create a cross-connect (xconnect) connection, which creates a virtual circuit between two network devices. This virtual circuit can be thought of as a logical connection that allows data to be transmitted between these devices as if they were directly connected, even if they are physically separated and connected through a service provider network.

1. First, you must determine what parts of a final L2VPN configuration are already implemented and what parts you need to add.
   a. Open BPO UI. Navigate to the **Network > All Resources** tab. Create a new L2VPN resource. Make sure that you have Customer, Bandwidth Profile and two Site resources.



   b. Open your terminal and connect to a managed device, which is attached to a Site resource you used in the L2VPN resource.

```
student@POD-XX:~$ ssh admin@PE-1
admin@pe-1's password: admin
admin@PE-1 [/]>
```

   c. Display device configuration with the **show configuration** [**interface** | **policy** | **vlan**] commands.

```
admin@PE-1 [/]> show configuration interface
dev-sim:interface {
   Loopback 0 {
      ip {
         address 127.0.0.1;
      }
      shutdown no;
   }
   Loopback 110 {
      ip {
         address 10.255.255.1;
      }
      shutdown no;
   }
   FastEthernet 0/0 {
   }
   FastEthernet 0/1 {
   }
   GigabitEthernet 0/0/0 {
   }
   GigabitEthernet 0/0/0.100 {
      encapsulation {
```

```
         dot1Q {
            vlan-id 100;
         }
      }
   }
   GigabitEthernet 0/0/1 {
   }
   GigabitEthernet 0/0/2 {
   }
   GigabitEthernet 0/0/3 {
   }
   TenGigabitEthernet 0/0 {
   }
   TenGigabitEthernet 0/1 {
   }
}admin@PE-1 [/]> show configuration policy
dev-sim:policy {
   policy-map PE-1.Service1.PolicyMap {
      class Service1-policy-class {
      }
      police {
         cir 8000;
         bc 8000;
         pir 9000;
         be 9000;
      }
   }
}admin@PE-1 [/]> show configuration vlan
dev-sim:vlan {
   vlan-list 1 {
      name default;
   }
   vlan-list 100 {
      name PE-1.Service1.VLAN;
   }
}
```

h. Next, study an example of a complete point-to-point (p2p) L2VPN configuration. Identify the missing parts in your service.

```
   GigabitEthernet <interface-id>.<vlan> {
      encapsulation {
         dot1Q {
            vlan-id <vlan>;
         }
      }
      service-policy {
         input <device>.<service>.PolicyMap;
      }
      xconnect {
         address <neighbor-address>;
         vcid <vlan>;
         encapsulation mpls;
      }
   }

   policy-map <device>.<service>.PolicyMap {
      class <service>-policy-class {
      }
      police {
         cir <rate>;
         bc <rate>;
         pir <rate>;
         be <rate>;
      }
   }
```

```
   vlan-list <vlan> {
      name <device>.<service>.VLAN;
   }
```

d. You notice that you have applied most of the necessary configuration through the managed resources. The only things that are missing are in the following output:

```
   GigabitEthernet <interface-id>.<vlan> {
      encapsulation {
         dot1Q {
            vlan-id <vlan>;
         }
      }
      service-policy {
         input <device>.<service>.PolicyMap;
      }
      xconnect {
         address <neighbor-address>;
         vcid <vlan>;
         encapsulation mpls;
      }
   }
```

i. This means that you need to update the logical interface creation code to include setting an input service policy for your logical interface, as well as creating an xconnect object that contains the following parameters:

- **address**, which should be set to the IP address of a specific Loopback interface on a neighboring device (in this case, Loopback110).
- **vcId** (virtual circuit ID), which should be set to the VLAN ID.
- **encapsulation** type, which should be set to use mpls.

2. Open the **l2vpn.py** file in the **model-definitions/training** folder and add these changes.

a. Edit the **create_managed_port()** method. Add the **xconnect** and **service-policy** objects to the properties of the **data** parameter used in the **bpo.resources.create()** call.

```
    def create_managed_logical_port(self, resource, allocated_vlan):

        properties = resource['properties']

        # Create a managed logical port on managed device in each Site
        for site_label in properties['sites']:
            # Read the L2VPN Port resource
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )

            # Read the L2VPN Port Managed Device and port resource
            device_name = port_resource["properties"]["deviceName"]
            managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
            managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])

            # Create managed logical port resource
            logical_port_product_id = managed_port_resource['productId']
            managed_logical_port_resource = self.bpo.resources.create(
                managed_port_resource['id'], {
                'productId': logical_port_product_id,
                'label': f"{managed_port_resource['label']}.{allocated_vlan}",
                'properties': {
                    "name":  f"{managed_port_resource['label']}.{allocated_vlan}",
                    "device": managed_device_resource['id'],
                    "encapsulation": {
                        "dot1Q": {
```

```
                            "vlan-id": allocated_vlan
                        }
                    },
                    "xconnect": {},
                    "service-policy": {}
                }
            })
            self.bpo.relationships.add_relationship(port_resource['id'],
managed_logical_port_resource[0]['id'])
```

b. Set the xconnect parameters – set the address to an empty string (for now), set the **vcid** to the ID of the allocated VLAN, and set the **encapsulation** to **mpls**.

```
    def create_managed_logical_port(self, resource, allocated_vlan):

        properties = resource['properties']

        # Create a managed logical port on managed device in each Site
        for site_label in properties['sites']:
            # Read the L2VPN Port resource
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )

            # Read the L2VPN Port Managed Device and port resource
            device_name = port_resource["properties"]["deviceName"]
            managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
            managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])

            # Create managed logical port resource
            logical_port_product_id = managed_port_resource['productId']
            managed_logical_port_resource = self.bpo.resources.create(
                managed_port_resource['id'], {
                'productId': logical_port_product_id,
                'label': f"{managed_port_resource['label']}.{allocated_vlan}",
                'properties': {
                    "name":  f"{managed_port_resource['label']}.{allocated_vlan}",
                    "device": managed_device_resource['id'],
                    "encapsulation": {
                        "dot1Q": {
                            "vlan-id": allocated_vlan
                        }
                    },
                    "xconnect": {
                        "address": "",
                        "vcid": allocated_vlan,
                        "encapsulation": "mpls"
                    },
                    "service-policy": {}
                }
            })
            self.bpo.relationships.add_relationship(port_resource['id'],
managed_logical_port_resource[0]['id'])
```

c. Set the **service-policy input** parameter to the name of the L2VPN managed policy map in the format of <managed_device_label>.<l2vpn_label>.PolicyMap.

```
    def create_managed_logical_port(self, resource, allocated_vlan):

        properties = resource['properties']

        # Create a managed logical port on managed device in each Site
        for site_label in properties['sites']:
            # Read the L2VPN Port resource
            port_resource = self.bpo.resources.get_one_with_filters(
```

```
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )

            # Read the L2VPN Port Managed Device and port resource
            device_name = port_resource["properties"]["deviceName"]
            managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
            managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])

            # Create managed logical port resource
            logical_port_product_id = managed_port_resource['productId']
            managed_logical_port_resource = self.bpo.resources.create(
                managed_port_resource['id'], {
                'productId': logical_port_product_id,
                'label': f"{managed_port_resource['label']}.{allocated_vlan}",
                'properties': {
                    "name":  f"{managed_port_resource['label']}.{allocated_vlan}",
                    "device": managed_device_resource['id'],
                    "encapsulation": {
                        "dot1Q": {
                            "vlan-id": allocated_vlan
                        }
                    },
                    "xconnect": {
                        "address": "",
                        "vcid": allocated_vlan,
                        "encapsulation": "mpls"
                    },
                    "service-policy": {
                        "input": f"{managed_device_resource['label']}.{resource['label']}.PolicyMap"
                    }
                }
            })
            self.bpo.relationships.add_relationship(port_resource['id'],
managed_logical_port_resource[0]['id'])
```

d. Create a new method in the *Activate* class that will read the address of a specific Loopback interface on the neighboring device and call it **get_neightbor_device_loopback_address()**. Add the l2vpn resource properties, managed device object, and a loopback interface ID to the method parameters. The loopback ID default value should be set to 110 since this is the default Loopback interface that should be used in this scenario.

```
    def get_neighbor_device_loopback_address(self, properties, managed_device_resource, loopback_id="110"):
```

e. Implement this method. First, loop through all the sites in the l2vpn resource properties and store the device name in each loop.

```
    def get_neighbor_device_loopback_address(self, properties, managed_device_resource, loopback_id="110"):
        # Loop through sites and find the "neighbor" device
        for site_label in properties['sites']:
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )
            device_name = port_resource["properties"]["deviceName"]
```

f. Next, find the neighbor device managed resource. Since this l2vpn resource supports only a p2p service between two sites, the neighbor device is not the managed device you are creating the configuration for, but the other one.

```
    def get_neighbor_device_loopback_address(self, properties, managed_device_resource, loopback_id="110"):
        # Loop through sites and find the "neighbor" device
        for site_label in properties['sites']:
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
```

```
                QQuery("label", f"{site_label}.Port")
            )
            device_name = port_resource["properties"]["deviceName"]

            # Get the address of a loopback interface
            # Since you are limited to 2 sites/devices per service, the names must not match
            if device_name != managed_device_resource['label']:
                neighbor_device_resource = PortUtils.get_managed_device_resource(self, device_name)
```

g. Find the Loopback IP address for this device, by searching through its dependents for a resource with **radevsim.resourceTypes.Loopback** resource type and the **loopback_id** as its label.

```
    def get_neighbor_device_loopback_address(self, properties, managed_device_resource, loopback_id="110"):
        # Loop through sites and find the "neighbor" device
        for site_label in properties['sites']:
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )
            device_name = port_resource["properties"]["deviceName"]

            # Get the address of a loopback interface
            # Since you are limited to 2 sites/devices per service, the names must not match
            if device_name != managed_device_resource['label']:
                neighbor_device_resource = PortUtils.get_managed_device_resource(self, device_name)
                loopback_resources = self.bpo.resources.get_dependent_with_filters(
                    neighbor_device_resource['id'],
                    ExactTypeId("radevsim.resourceTypes.Loopback"),
                    QQuery("label", loopback_id)
                )
```

h. Finally, return the loopback IP address as the neighbor device loopback address. It can be found under the loopback resource **properties.ip.address** value. Raise an Exception in case the neighbor device loopback address is not found.

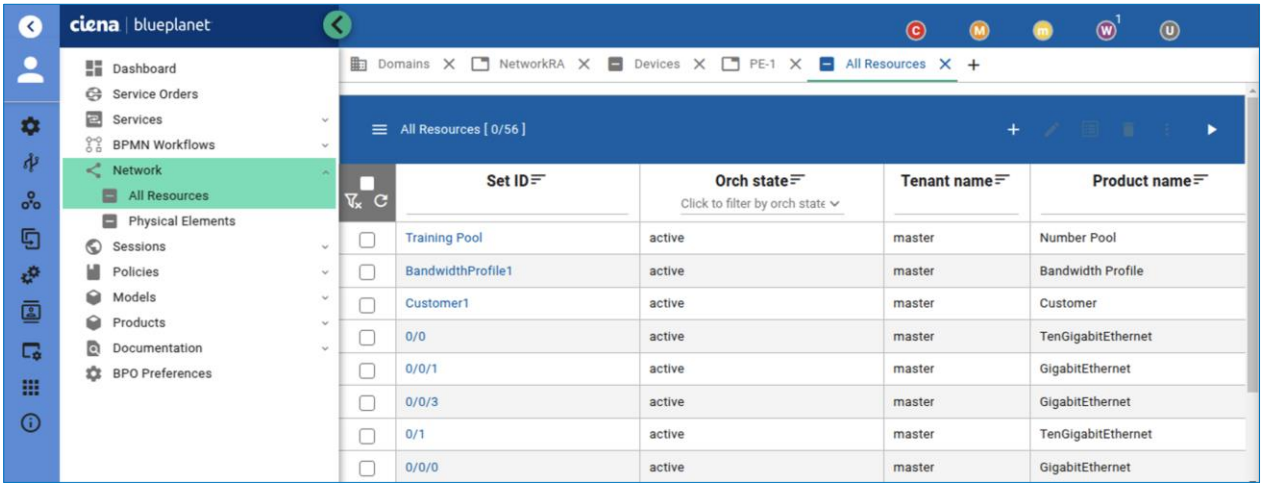
```
    def get_neighbor_device_loopback_address(self, properties, managed_device_resource, loopback_id="110"):
        # Loop through sites and find the "neighbor" device
        for site_label in properties['sites']:
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )
            device_name = port_resource["properties"]["deviceName"]

            # Get the address of a loopback interface
            # Since you are limited to 2 sites/devices per service, the names must not match
            if device_name != managed_device_resource['label']:
                neighbor_device_resource = PortUtils.get_managed_device_resource(self, device_name)
                loopback_resources = self.bpo.resources.get_dependent_with_filters(
                    neighbor_device_resource['id'],
                    ExactTypeId("radevsim.resourceTypes.Loopback"),
                    QQuery("label", loopback_id)
                )
                return loopback_resources['properties']['ip']['address']
        raise Exception(f"Neighbor device loopback address not found for
{managed_device_resource['label']}")
```

i. Use this newly created method in the **create_managed_port_resource** method and set the **properties.xconnect.address** with it. To make this method work with **Update** and **ChangePort** classes as well (since they also rely on the **create_managed_port_resource** method), call it directly from the *Activate* class instead of the current ***self*** object.

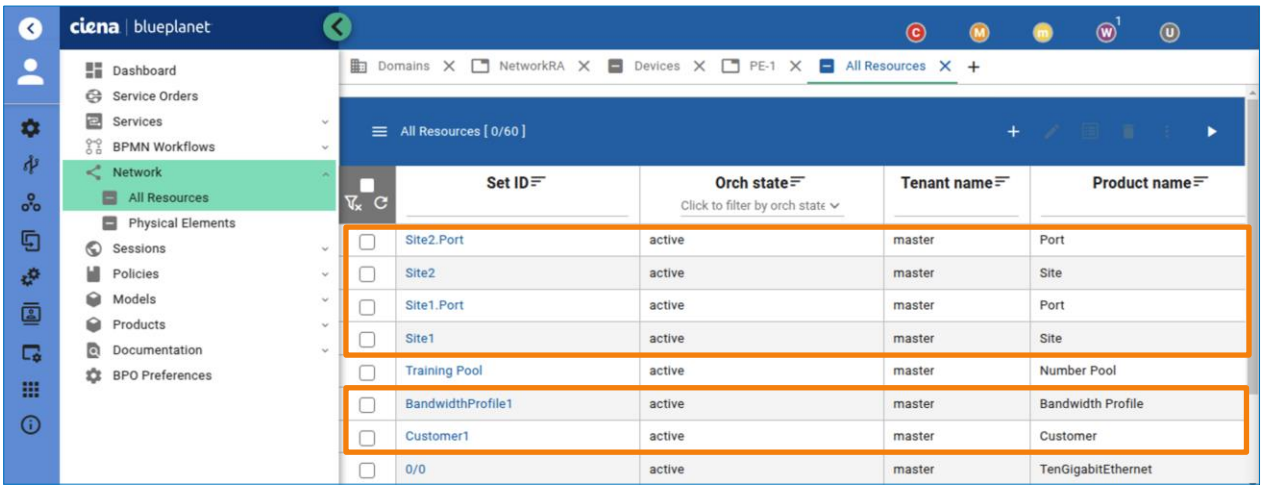
```
    def create_managed_logical_port(self, resource, allocated_vlan):

        properties = resource['properties']

        # Create a managed logical port on managed device in each Site
```

```
        for site_label in properties['sites']:
            # Read the L2VPN Port resource
            port_resource = self.bpo.resources.get_one_with_filters(
                ExactTypeId("training.resourceTypes.Port"),
                QQuery("label", f"{site_label}.Port")
            )

            # Read the L2VPN Port Managed Device and port resource
            device_name = port_resource["properties"]["deviceName"]
            managed_device_resource = PortUtils.get_managed_device_resource(self, device_name)
            managed_port_resource = PortUtils.get_managed_port_resource(self,
port_resource['properties']['portName'], port_resource['properties']['deviceName'],
port_resource['properties']['portSpeed'])

            # Create managed logical port resource
            logical_port_product_id = managed_port_resource['productId']
            managed_logical_port_resource = self.bpo.resources.create(
                managed_port_resource['id'], {
                'productId': logical_port_product_id,
                'label': f"{managed_port_resource['label']}.{allocated_vlan}",
                'properties': {
                    "name":  f"{managed_port_resource['label']}.{allocated_vlan}",
                    "device": managed_device_resource['id'],
                    "encapsulation": {
                        "dot1Q": {
                            "vlan-id": allocated_vlan
                        }
                    },
                    "xconnect": {
                        "address": self.get_neighbor_device_loopback_address(self, properties,
managed_device_resource),
                        "vcid": allocated_vlan,
                        "encapsulation": "mpls"
                    },
                    "service-policy": {
                        "input": f"{managed_device_resource['label']}.{resource['label']}.PolicyMap"
                    }
                }
            })
            self.bpo.relationships.add_relationship(port_resource['id'],
managed_logical_port_resource[0]['id'])
```

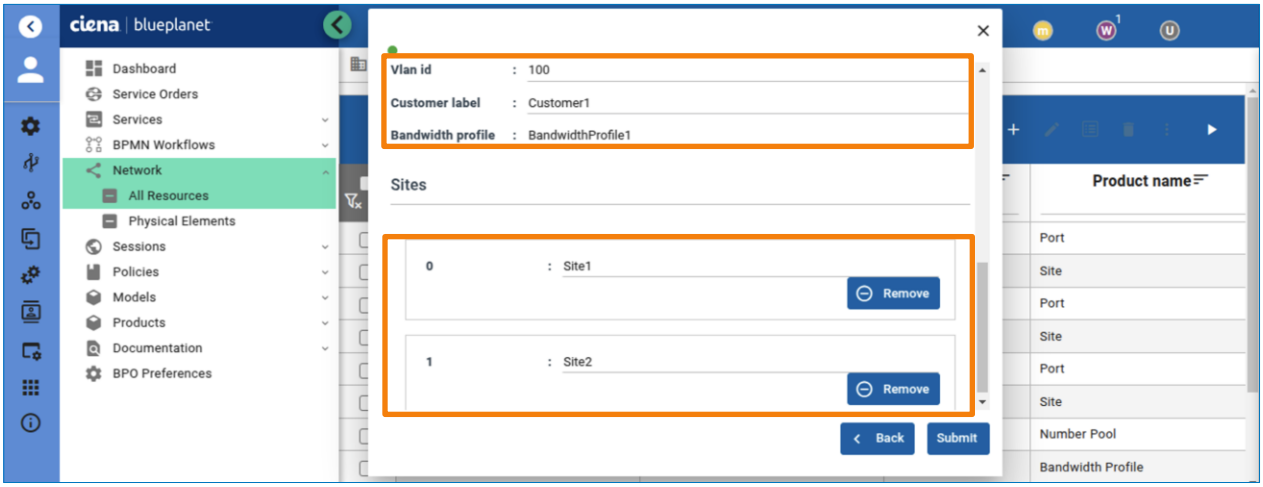
3. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.
4. In BPO UI, remove the before created L2VPN resource and all Site resources.



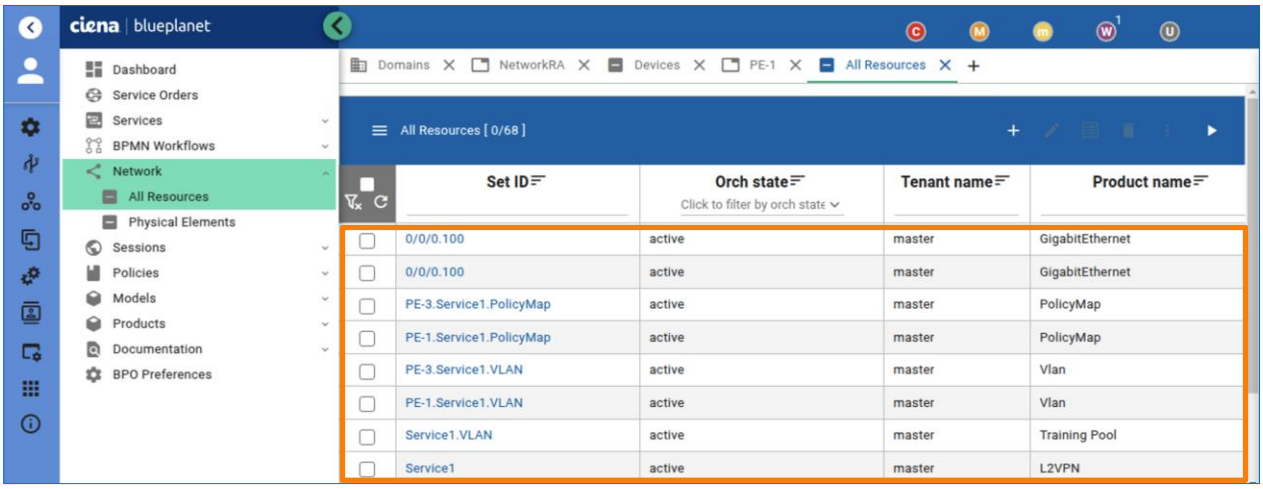
5. Perform a final test of your L2VPN resource. Open the BPO UI and go to **Network > All Resources.**
   a. Make sure that you have Customer resource **Customer1** and Bandwidth Profile resource **BandwidthProfile1**.
   b. Create two new Site resources:
      - **Site1**, which should use the **GigabitEthernet 0/0/0** port on the **PE-1** device.
      - **Site2**, which should use the **GigabitEthernet 0/0/0** port on the **PE-3** device.
   c. These are the resources that you should end up with.



   d. Create a new L2VPN resource that is based on these sub-resources.



e. You should again end up with two new managed logical port interfaces, two new managed policy maps, two new managed VLAN resources, an allocated VLAN resource, and an L2VPN resource – all in an active state. If not, inspect the error messages on failed resources and troubleshoot them.



f. Open the command line and connect to the **PE-1** device.

```
student@POD-XX:~$ ssh admin@PE-1
admin@pe-1's password: admin
admin@PE-1 [/]>
```

e. Inspect the entire configuration. Identify the parts of the configuration you have added to with the **show configuration** [**interface** | **policy** | **vlan**] commands.

```
admin@PE-1 [/]> show configuration interface
dev-sim:interface {
   Loopback 0 {
      ip {
         address 127.0.0.1;
      }
      shutdown no;
   }
   Loopback 110 {
      ip {
         address 10.255.255.1;
      }
      shutdown no;
   }
   FastEthernet 0/0 {
   }
   FastEthernet 0/1 {
   }
   GigabitEthernet 0/0/0 {
   }
   GigabitEthernet 0/0/0.100 {
      encapsulation {
         dot1Q {
            vlan-id 100;
         }
      }
      service-policy {
         input PE-1.Service1.PolicyMap;
      }
      xconnect {
         address 10.255.255.3;
         vcid 100;
         encapsulation mpls;
```

```
      }
   }
   GigabitEthernet 0/0/1 {
   }
   GigabitEthernet 0/0/2 {
   }
   GigabitEthernet 0/0/3 {
   }
   TenGigabitEthernet 0/0 {
   }
   TenGigabitEthernet 0/1 {
   }
}
admin@PE-1 [/]> show configuration policy
dev-sim:policy {
   policy-map PE-1.Service1.PolicyMap {
      class Service1-policy-class {
      }
      police {
         cir 8000;
         bc 8000;
         pir 9000;
         be 9000;
      }
   }
}
admin@PE-1 [/]> show configuration vlan
dev-sim:vlan {
   vlan-list 1 {
      name default;
   }
   vlan-list 100 {
      name PE-1.Service1.VLAN;
   }
}
```

g. Mark down the logical port xconnect address.

```
   GigabitEthernet 0/0/0.100 {
      encapsulation {
         dot1Q {
            vlan-id 100;
         }
      }
      service-policy {
         input PE-1.Service1.PolicyMap;
      }
      xconnect {
         address 10.255.255.3;
         vcid 100;
         encapsulation mpls;
      }
   }
```

h. Connect to the second device that composes this L2VPN service – PE-3.

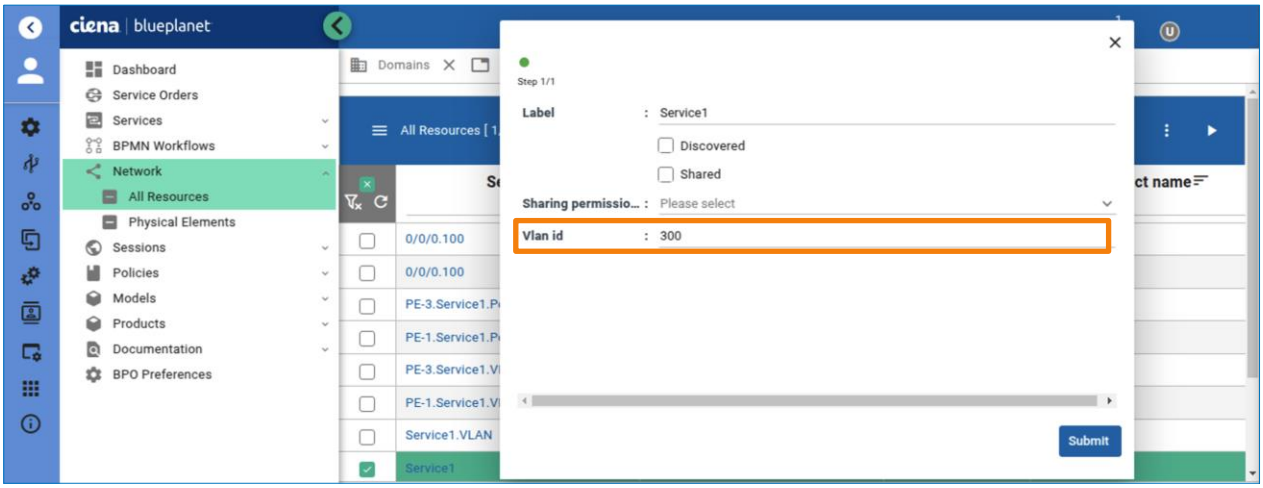
```
student@POD-XX:~$ ssh admin@PE-3
admin@pe-3's password: admin
admin@PE-3 [/]> show
```

i. Inspect the entire configuration here too. Ensure that the xconnect address you marked down two steps ago matches the IP address on the Loopback110 interface and vice versa.

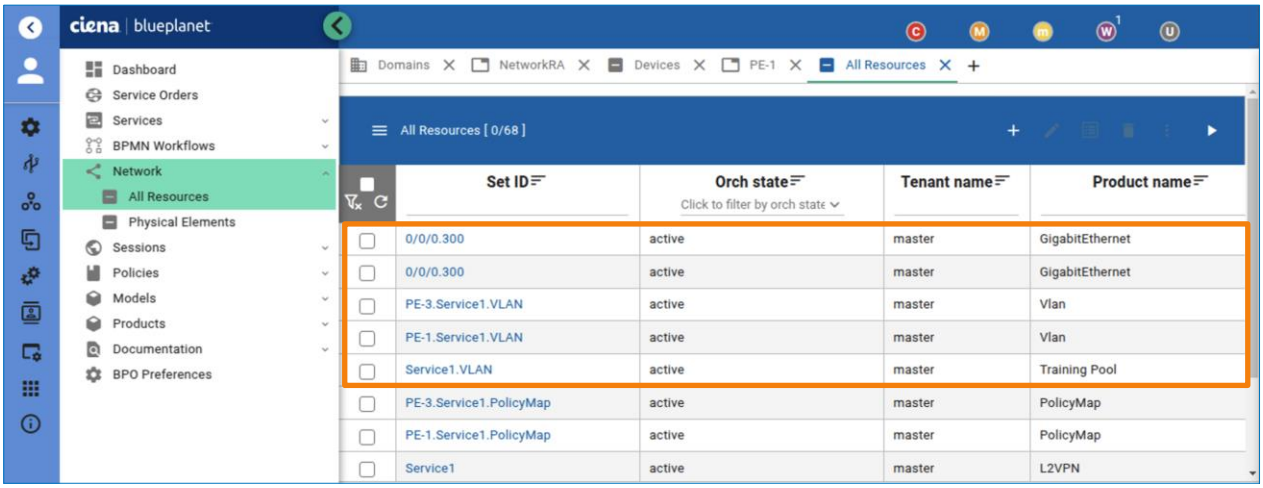
```
admin@PE-3 [/]> show configuration interface
dev-sim:interface {
   Loopback 0 {
      ip {
         address 127.0.0.1;
      }
```

```
      shutdown no;
   }
   Loopback 110 {
      ip {
         address 10.255.255.3;
      }
      shutdown no;
   }
   FastEthernet 0/0 {
   }
   FastEthernet 0/1 {
   }
   GigabitEthernet 0/0/0 {
   }
   GigabitEthernet 0/0/0.100 {
      encapsulation {
         dot1Q {
            vlan-id 100;
         }
      }
      service-policy {
         input PE-3
.Service1.PolicyMap;
      }
      xconnect {
         address 10.255.255.1;
         vcid 100;
         encapsulation mpls;
      }
   }
   GigabitEthernet 0/0/1 {
   }
   GigabitEthernet 0/0/2 {
   }
   GigabitEthernet 0/0/3 {
   }
   TenGigabitEthernet 0/0 {
   }
   TenGigabitEthernet 0/1 {
   }
}
admin@PE-3 [/]> show configuration policy
dev-sim:policy {
   policy-map PE-3.Service1.PolicyMap {
      class Service1-policy-class {
      }
      police {
         cir 8000;
         bc 8000;
         pir 9000;
         be 9000;
      }
   }
}
admin@PE-3 [/]> show configuration vlan
dev-sim:vlan {
   vlan-list 1 {
      name default;
   }
   vlan-list 100 {
      name PE-3.Service1.VLAN;
   }
}
```

j. Open the BPO UI again. This time, test the **Update** operation by editing the L2VPN service. Set a new VLAN ID for your L2VPN resource.



k. Make sure the old managed logical ports get deleted and new ones successfully created. The same applies to the managed VLAN resource.



l. Connect to the **PE-1** device. Inspect the configuration again and ensure the correct (updated) VLAN is now used for your service.

```
student@POD-XX:~$ ssh admin@PE-1
admin@pe-1's password: admin
admin@PE-1 [/]> show configuration interface
dev-sim:interface {
   Loopback 0 {
      ip {
         address 127.0.0.1;
      }
      shutdown no;
   }
   Loopback 110 {
      ip {
         address 10.255.255.1;
      }
      shutdown no;
   }
   FastEthernet 0/0 {
   }
```

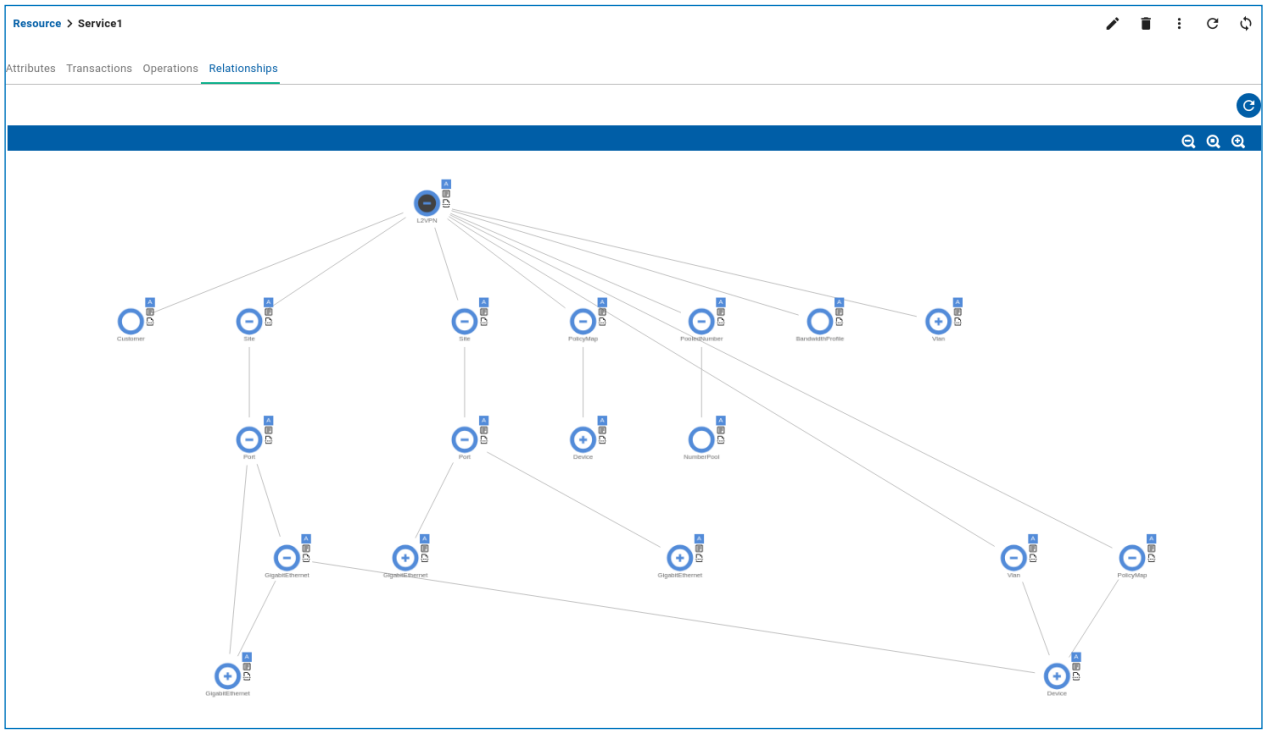
```
   FastEthernet 0/1 {
   }
   GigabitEthernet 0/0/0 {
   }
   GigabitEthernet 0/0/0.300 {
      encapsulation {
         dot1Q {
            vlan-id 300;
         }
      }
      service-policy {
         input PE-1.Service1.PolicyMap;
      }
      xconnect {
         address 10.255.255.3;
         vcid 300;
         encapsulation mpls;
      }
   }
   GigabitEthernet 0/0/1 {
   }
   GigabitEthernet 0/0/2 {
   }
   GigabitEthernet 0/0/3 {
   }
   TenGigabitEthernet 0/0 {
   }
   TenGigabitEthernet 0/1 {
   }
}
admin@PE-1 [/]> show configuration policy
dev-sim:policy {
   policy-map PE-1.Service1.PolicyMap {
      class Service1-policy-class {
      }
      police {
         cir 8000;
         bc 8000;
         pir 9000;
         be 9000;
      }
   }
}
admin@PE-1 [/]> show configuration vlan
dev-sim:vlan {
   vlan-list 1 {
      name default;
   }
   vlan-list 300 {
      name PE-1.Service1.VLAN;
   }
}
```

m. Congratulations, you have successfully implemented this p2p L2VPN service.

Task 3: Add a New Custom Operation for Checking Service Status

Complex resources and services tend to end up composed of multiple instances of different resources, forming multiple dependencies with each other. Checking each resource and its dependencies manually can be time-consuming. Fortunately, there are several ways, both graphical and programmatical, to check the state of all the resources involved in a single service.

1. Open the BPO UI and navigate to the **Network > All Resources**. Inspect the L2VPN resource and open the **Relationships** tab.
2. Expand all the resources by clicking on the **+** symbol in each circle. You see that your service now forms quite a sizeable dependency tree. This is one way to see the state of your service since active resources are marked in blue, and failed ones are marked in red. For now, all your resources should be active.



3. Another way to see the state of your resources is programmatically. You implement a custom operation to loop through all the L2VPN dependencies and sort them by active and failed.
   a. First, create a new custom operation. Open the **service_template_l2vpn.tosca** file in the **bpo-sdd/model-definitions/types/tosca/training** folder. Add a new **getStatus** custom operation. The remote script path should point to **training.l2vpn.GetStatus**.

```
"$schema"    = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN service template definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN service template."
authors     = [ "Developer (developer@bptrn.com)"  ]

serviceTemplates {

  L2VPN {
    title = "L2VPN"
    description = "A service template for L2VPN."
    implements = training.resourceTypes.L2VPN
```

```
    plans {

      activate {
        type = remote
        language = python
        path = training.l2vpn.Activate
      }

      terminate {
        type = remote
        language = python
        path = training.l2vpn.Terminate
      }

      update {
        type = remote
        language = python
        path = training.l2vpn.Update
      }

      changePort {
        type = remote
        language = python
        path = training.l2vpn.ChangePort
      }

      getStatus {
        type = remote
        language = python
        path = training.l2vpn.GetStatus
      }
    }

    validators {

      activate {
        type = remote
        language = python
        path = training.l2vpn.ValidateActivate
      }
    }
  }
}
```

b. Add a custom interface for this operation. Open the **resource_type_l2vpn.tosca** file and add a new **getStatus** interface. It should have no inputs and two string-type outputs – **active** and **failed**.

```
"$schema"   = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
title       = "L2VPN resource type definition"
package     = training
version     = "1.0"
description = "This TOSCA document defines the L2VPN resource type."
authors     = [ "Developer (developer@bptrn.com)"  ]

imports {
  Root = tosca.resourceTypes.Root
}

resourceTypes {
  L2VPN {
    derivedFrom = Root
    title = "L2VPN"
    description = "The L2VPN resource is used to create a point to point layer 2 service."

    properties {

      vlanId {
```

```
      title = "VLAN Identifier"
      description = "Specify the VLAN to be used by the L2VPN"
      type = integer
      optional = true
      updatable = true
      minimum = 2
      maximum = 4095
    }

    customerLabel {
      title = "Customer Label"
      description = "Label of the customer"
      type = string
    }

    sites {
      title = "Site list"
      description = "Array of sites to be used for the service creation"
      type = array
      minItems = 2
      maxItems = 2
    }

    bandwidthProfile {
      title = "Bandwidth Profile"
      description = "Label of the bandwidth profile"
      type = string
    }
  }

  interfaces {
    changePort {
      title = "Change port"
      description = "Change port on a specified site for l2vpn resource"

      inputs {

        portName {
          title = "Port Name"
          description = "New port for the site"
          type = string
        }

        deviceName {
          title = "Device Name"
          description = "New device for the site"
          type = string
        }

        portSpeed {
          title = "Port Speed"
          description = "Speed of the port"
          type = string
          enum = [100Mbps, 1Gbps, 10Gbps]
        }

        siteLabel {
          title = "Site Label"
          description = "Site in which to change the port"
          type = string
        }
      }
      outputs {
      }
    }
```

```
      getStatus {
        title = "Get Service Status"
        description = "Get Service status by validating all resources are in an active state"

        inputs {
        }
        outputs {
          active {
            title = "Active resources"
            description = "Active resources"
            type = string
          },
          failed {
            title = "Failed resources"
            description = "Failed resources"
            type = string
          }
        }
      }
    }
  }
}
```

c. Now implement a **GetStatus** class that will handle this custom operation. Open the **l2vpn.py** file in the **bpo-sdd/model-definitions/training** folder and add a **GetStatus** class there containing a **run()** method.

```
class GetStatus(Plan):
    """
    Get Service status by validating all resources are in an active state
    """
    def run(self):
```

d. Read the resource ID and object and add some basic logging to print these out.

```
class GetStatus(Plan):
    """
    Get Service status by validating all resources are in an active state
    """
    def run(self):
        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"GetStatus: resourceId {resource_id}")
        log.info(f"GetStatus: {resource}")
```

e. Read the dependencies for your L2VPN resource. Use the **get_dependencies_with_filters()** method and use the **Recursive** modifier. Study the PlanSDK documentation to understand how to use it.

```
from plansdk.apis.plan import Plan
from plansdk.apis.bpo import ExactTypeId, QQuery, Recursive
from .port import PortUtils
from .util import failure, success
import logging

...

class GetStatus(Plan):
    """
    Get Service status by validating all resources are in an active state
    """
    def run(self):
        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"GetStatus: resourceId {resource_id}")
        log.info(f"GetStatus: {resource}")
```

```
        log.info(f"Reading dependencies for the L2VPN service - {resource['label']}")
        dependencies = self.bpo.resources.get_dependencies_with_filters(
                    resource['id'],
                    Recursive(True)
                )
```

f. Create two new output strings that contain information about dependent resources. Loop through all dependencies and sort them by the **orchState** of the resource.

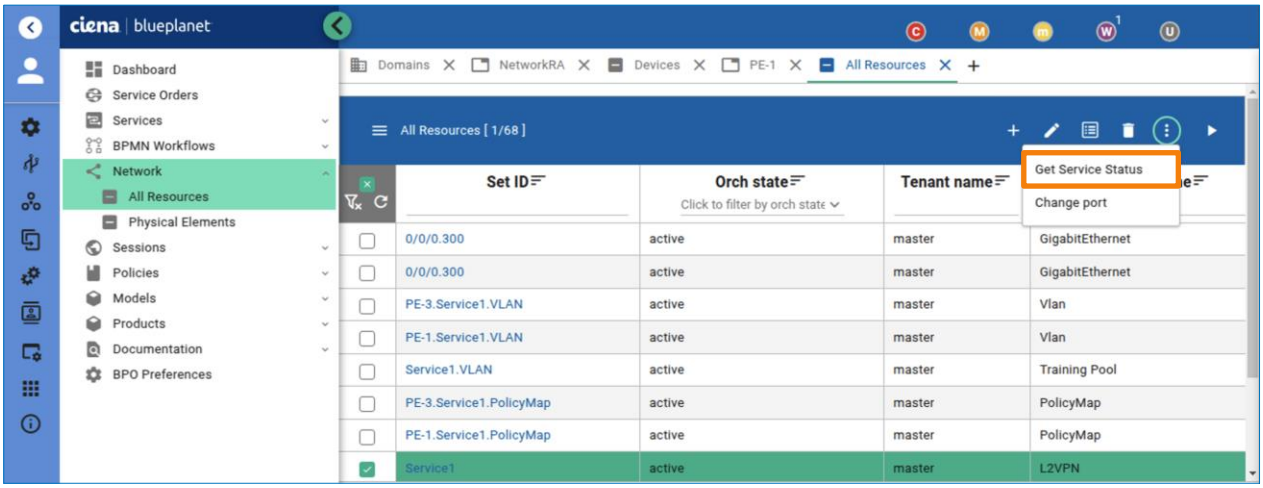
```
class GetStatus(Plan):
    """
    Get Service status by validating all resources are in an active state
    """
    def run(self):
        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"GetStatus: resourceId {resource_id}")
        log.info(f"GetStatus: {resource}")

        log.info(f"Reading dependencies for the L2VPN service - {resource['label']}")
        dependencies = self.bpo.resources.get_dependencies_with_filters(
                    resource['id'],
                    Recursive(True)
                )

        active = ""
        failed = ""

        for dependency in dependencies:
            if dependency['orchState'] == 'active':
                active += f"{dependency['label']}({dependency['resourceTypeId']})  "
            else:
                failed += f"{dependency['label']}({dependency['resourceTypeId']})  "
```

g. Finally, return a dictionary containing the **active** and **failed** keys and correlated values.

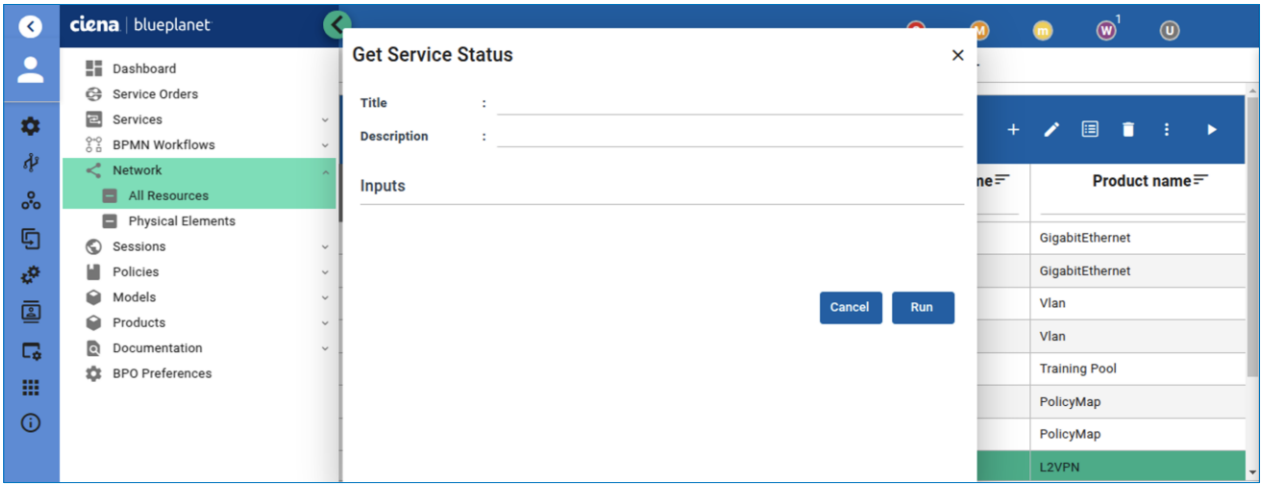
```
class GetStatus(Plan):
    """
    Get Service status by validating all resources are in an active state
    """
    def run(self):
        resource_id = self.params['resourceId']
        resource = self.bpo.resources.get(resource_id)

        log.info(f"GetStatus: resourceId {resource_id}")
        log.info(f"GetStatus: {resource}")

        log.info(f"Reading dependencies for the L2VPN service - {resource['label']}")
        dependencies = self.bpo.resources.get_dependencies_with_filters(
                    resource['id'],
                    Recursive(True)
                )

        active = ""
        failed = ""

        for dependency in dependencies:
            if dependency['orchState'] == 'active':
                active += f"{dependency['label']}({dependency['resourceTypeId']})  "
            else:
                failed += f"{dependency['label']}({dependency['resourceTypeId']})  "

        return {"active": active, "failed": failed}
```

h. **Onboard** the changes to the BPO server using the Blue Planet VS Code extension.

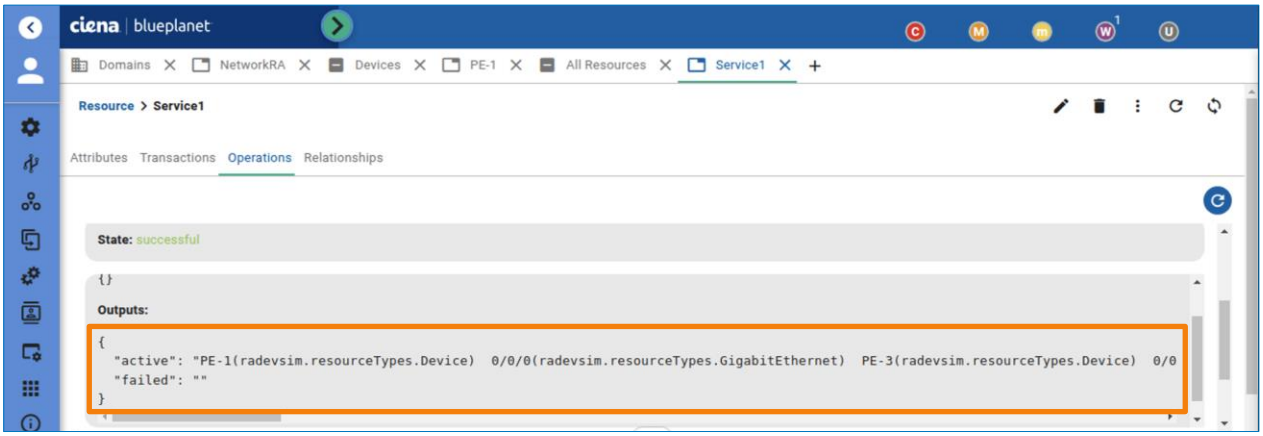
4. Open the BPO UI and navigate to the **Network > All Resources** tab. Choose your resource and then choose the **Get Service Status** custom operation.



5. Since there are no inputs for this custom operation, you can click **Run** to start the operation.



6. Wait for the operation to succeed, inspect the L2VPN resource, and open the **Operations** tab. Open the execution log of the last operation and check the output message. As you can also see here, all the resources that compose your service are in an active state.



End of Lab