Create SVI Service Using pre_modification Service Callback

Introduction

In this activity, you will use advanced YANG modeling techniques and implement a *pre_modification* service callback.

The service will need a YANG data model, XML templates, and some Python code. Your new service package will be based on the existing model, which will be provided to you in a separate folder.

Because your service is intended for switches, you will put a constraint in your data model. The YANG modeling language allows creating constraints with a *must* statement.

The *pre_modification* callback will be implemented in Python and will provide the next available *vlan-id* to the upgraded *svi* service. The callback will be executed when you commit the configuration. Therefore, your service can keep track of every vlan-id in use without the need for a separate database.

This activity will also show you how to use VS Code to debug Python code that is running inside a container.

After completing this activity, you will be able to:

- Modify an existing YANG service model to impose stricter constraints on input data
- Implement a *pre_modification* service callback using Python
- Use the *pylint* tool to analyze the code

- Use the Visual Studio Code remote debugger

Job Aids

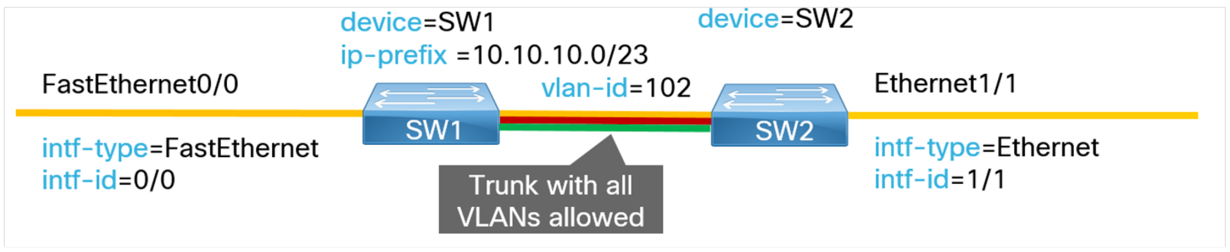
The following job aid is available to help you complete the lab activities:

- This Lab Guide

The following table contains passwords that you might need.

Device	**Username**	**Password**
Student-VM	student	1234QWer
NSO application	admin	admin

The following figure provides a visual aid for this activity.



Required Resources

The following resources and equipment are required for completing the activities in this lab guide:

- PC or laptop with a web browser
- Access to the Internet

Command Syntax Reference

This lab guide uses the following conventions for command syntax:

Formatting	**Description and Examples**
show running config	Commands in steps use this formatting.
Example	Type **show running config**

Formatting	Description and Examples
Example	Use the **name** command.
`show running config`	Commands in CLI outputs and configurations use this formatting.
highlight	CLI output that is important is highlighted.

Example	`student@student-vm:~$ ncs --version`
	`5.3.2`

Example	Save your current configuration as the default **startup config**. `Router Name# copy running startup`
brackets ([ ])	Indicates optional element. You can choose one of the options.
Example:	`(config-if)# frame-relay lmi-type {ansi|cisco|q933a}`
italics font	Arguments for which you supply values.
Example	Open file **ip tcp window-size** *bytes*
angle brackets (<>)	In contexts that do not allow italics, arguments for which you supply values are enclosed in angle brackets [<>]. Do not type the brackets when entering the command.
Example	If the command syntax is **ping** <*ip_address*>, you enter ping *192.32.10.12*
string	A non-quoted set of characters. Type the characters as-is.
Example	(config)# **hostname MyRouter**
vertical line (|)	Indicates that you enter one of the choices. The vertical line separates choices. Do not type the vertical line when entering the command.
Example	If the command syntax is **show ip route|arp**, you enter either **show ip route** or **show ip arp**, but not both.

Command List

The following are the most common commands that you will need.

Linux Shell:

Command	Comment

Command	Comment
source /opt/ncs/ncs-5.3.2/ncsrc	Source NSO environmental variable in Docker container.
ls|ll	Display contents of the current directory.
cd	Move directly to user home directory.
cd ..	Exit out of current directory.
cd test	Move into folder "test" which is a subfolder of the current directory.
cd /home/student/nso300	Move into folder "nso300" by specifying direct path to it starting from the root of directory system.
ncs_cli -C -u admin	Log in to NSO CLI directly from local server.

NSO CLI:

Command	Comment
switch cli	Change CLI style.
show ?	Display all command options for current mode.
configure	Enter configuration mode.
commit	Commit new configuration (configuration mode only command).
show configuration	Display new configuration that has not yet been committed (configuration mode only command).

Makefile commands for Docker environment:

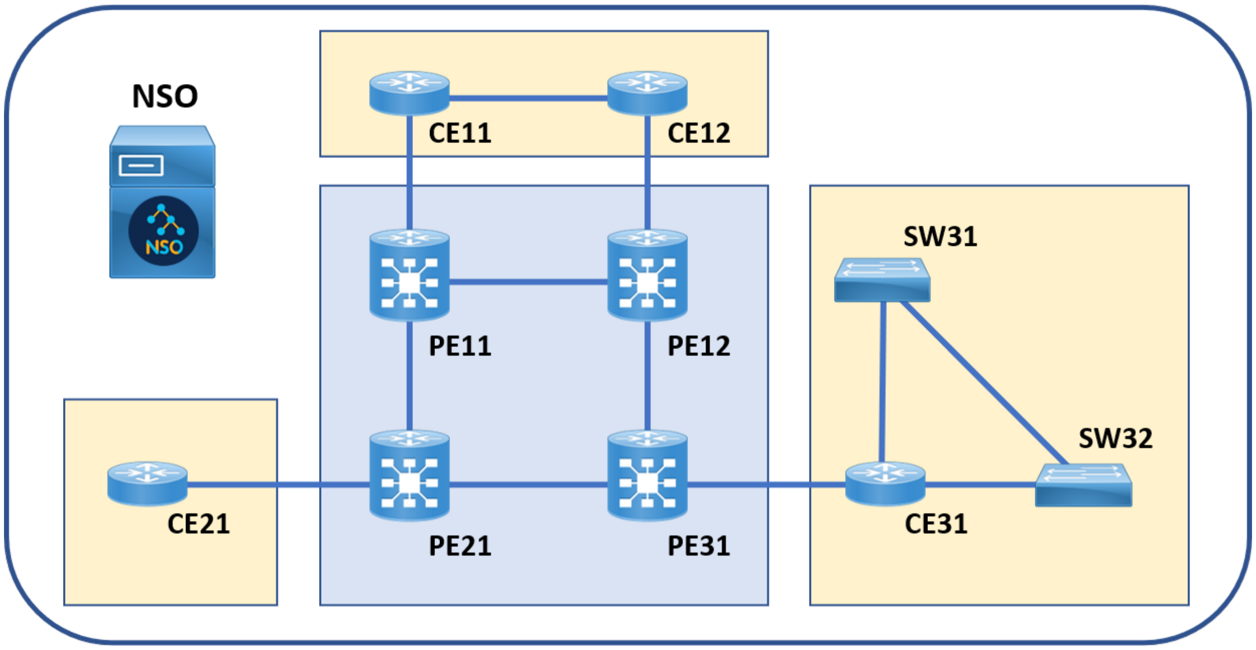
Command	Comment
make build	Builds the main NSO Docker image.
make testenv-start	Starts the NSO Docker environment.
make testenv-stop	Stops the NSO Docker environment.
make testenv-build	Recompiles and reloads the NSO packages.
make testenv-cli	Enters the NSO CLI of the NSO Docker container.
make testenv-shell	Enters the Linux shell of the NSO Docker container.
make dev-shell	Enters the Linux shell of the NSO Docker development container.

Lab Topology Information

Your lab session is your own personal sandbox. What you do in your session will not be reflected in anyone else's session. There are two topologies. The general topology is your lab environment with a Linux server (Student-VM). On the Linux server within that topology is your second topology—a Docker environment, which consists of an NSO Docker image with your NSO installation, together with numerous Docker images of NetSim routers and switches that are logically grouped into a network topology. This will be the network that you will orchestrate with your NSO.

- The network topology is designed in a way to cover both Service Provider and Enterprise use cases. It is a simulated NetSim network—devices have no Control or Data Plane. The devices will, however, accept or reject configurations sent by the NSO, just as real devices would.

Topology



Task 1: Upgrade the YANG Service Model

In this task, you will upgrade the *svi* YANG model. Files from the old service have been provided for you in the *$HOME/packages/svi* directory.

The *must* YANG statement requires an argument in the form of a string. The string must be a valid XPath expression that evaluates to true for data to be valid.

The feature *interface-vlan* must be enabled if the ned-id is *cisco-nx*. This configuration will be a part of the service template.

The final solutions for all labs, including this one, are located in the ~/solutions directory. You can use them for copying longer pieces of code and as a reference point for troubleshooting your packages.

Activity

Complete these steps:

Step 1

Connect to the Student-VM server by clicking the icon labelled NSO in the topology.

Step 2

Open the terminal window; click the Terminal icon on the bottom bar.

```
student@student-vm:~$
```

Step 3

Change directory to *nso300*.

```
student@student-vm:~$ cd nso300
student@student-vm:~/nso300$
```

Step 4

Enter the development NSO Docker container shell with the **make dev-shell** command and enter the */src/packages* directory.

```
student@student-vm:~/nso300$ make dev-shell
docker run -it -v $(pwd):/src nso300.gitlab.local/cisco-nso-dev:5.3.2
root@095671b75dd1:/# cd src/packages/
root@095671b75dd1:/src/packages#
```

Step 5

Create a new *svi* package with the **ncs-make-package** command.

```
root@095671b75dd1:/src/packages# ncs-make-package --service-skeleton
python-and-template --component-class svi.Svi svi
```

Step 6

Change the ownership of the package and exit the Docker container.

```
root@095671b75dd1:/src/packages# chown -Rv 1000:1000 svi
changed ownership of 'svi/test/internal/Makefile' from root:root to
1000:1000
changed ownership of 'svi/test/internal/lux/Makefile' from root:root to
1000:1000
changed ownership of 'svi/test/internal/lux/service/Makefile' from
root:root to 1000:1000
changed ownership of 'svi/test/internal/lux/service/dummy-service.xml'
from root:root to 1000:1000
changed ownership of 'svi/test/internal/lux/service/run.lux' from
root:root to 1000:1000
changed ownership of 'svi/test/internal/lux/service/dummy-device.xml'
from root:root to 1000:1000
changed ownership of 'svi/test/internal/lux/service/pyvm.xml' from
root:root to 1000:1000
changed ownership of 'svi/test/internal/lux/service' from root:root to
```

```
1000:1000
changed ownership of 'svi/test/internal/lux' from root:root to
1000:1000
changed ownership of 'svi/test/internal' from root:root to 1000:1000
changed ownership of 'svi/test/Makefile' from root:root to 1000:1000
changed ownership of 'svi/test' from root:root to 1000:1000
changed ownership of 'svi/python/svi/svi.py' from root:root to
1000:1000
changed ownership of 'svi/python/svi/__init__.py' from root:root to
1000:1000
changed ownership of 'svi/python/svi' from root:root to 1000:1000
changed ownership of 'svi/python' from root:root to 1000:1000
changed ownership of 'svi/README' from root:root to 1000:1000
changed ownership of 'svi/templates/svi-template.xml' from root:root to
1000:1000
changed ownership of 'svi/templates' from root:root to 1000:1000
changed ownership of 'svi/src/Makefile' from root:root to 1000:1000
changed ownership of 'svi/src/yang/svi.yang' from root:root to
1000:1000
changed ownership of 'svi/src/yang' from root:root to 1000:1000
changed ownership of 'svi/src' from root:root to 1000:1000
changed ownership of 'svi/package-meta-data.xml' from root:root to
1000:1000
changed ownership of 'svi' from root:root to 1000:1000
root@095671b75dd1:/src/packages# exit
logout
student@student-vm:~/nso300$
```

Step 7

List the contents of the package.

The package you created exists on your Student-VM machine and is mounted as a volume to the development Docker container.

```
student@student-vm:~/nso300$ ls packages/svi/
package-meta-data.xml  python  README  src  templates  test
```

Step 8

Open and study the *package-meta-data.xml* of your new service package.

This is how the package-meta-data.xml file appears initially.

```
student@student-vm:~/nso300$ vim packages/svi/package-meta-data.xml
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>svi</name>
  <package-version>1.0</package-version>
  <description>Generated Python package</description>
  <ncs-min-version>5.3</ncs-min-version>

  <component>
    <name>svi</name>
    <application>
      <python-class-name>svi.svi.Svi</python-class-name>
    </application>
  </component>
</ncs-package>
```

You can also use an IDE tool or text editor of your choice. For example, to view the file in Microsoft Visual Studio Code, enter the command **code packages/svi/package-meta-data.xml**.

Step 9

Change the package description and component name to better represent service purpose.

```
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>svi</name>
  <package-version>1.0</package-version>
  <description>SVI Python and Template Service</description>
  <ncs-min-version>5.3</ncs-min-version>

  <component>
    <name>Switch Virtual Interface</name>
    <application>
      <python-class-name>svi.svi.Svi</python-class-name>
    </application>
  </component>
</ncs-package>
```

Step 10

Save the file and exit the file editor.

Step 11

Delete the generated XML template file.

```
student@student-vm:~/nso300$ rm packages/svi/templates/svi-template.xml
```

Step 12

Copy the existing templates from *~/packages/svi/templates* and verify that you have the correct files.

The files are separated by functionality. One template is for interface addressing, the other for vlan database and switch port configuration.

```
student@student-vm:~/nso300$ cp ~/packages/svi/templates/svi-*
packages/svi/templates
student@student-vm:~/nso300$ ls packages/svi/templates
svi-intf-template.xml  svi-vlan-template.xml
```

Step 13

Open the *svi-intf-template.xml* template and add the configuration for enabling the feature *interface-vlan*.

```
student@student-vm:~/nso300$ vim packages/svi/templates/svi-intf-
template.xml
```

Step 14

Save the file.

Step 15

Your *svi-intf-template-xml* should now appear as follows:

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">

    <device>
      <name>{/device/name}</name>
      <config>
        <!-- IOS -->
        <interface xmlns="urn:ios">
          <?if {string(name)=$SVI-DEVICE}?>
          <Vlan>
              <name>{$VLAN-ID}</name>
              <ip>
                <address>
                  <primary>
                    <address>{$IP-ADDR}</address>
                    <mask>{$NETMASK}</mask>
                  </primary>
                </address>
              </ip>
            </Vlan>
            <?end?>
        </interface>
        <!-- NX-OS -->
        <feature xmlns="http://tail-f.com/ned/cisco-nx">
            <interface-vlan/>
        </feature>
        <interface xmlns="http://tail-f.com/ned/cisco-nx">
          <?if {string(name)=$SVI-DEVICE}?>
          <Vlan>
            <name>{$VLAN-ID}</name>
            <ip>
              <address>
                <ipaddr>{$IP-PREFIX}</ipaddr>
              </address>
            </ip>
          </Vlan>
          <?end?>
        </interface>
      </config>
    </device>
  </devices>
</config-template>
```

This feature is already enabled if you use an older version of the *cisco-nx* NED.

Step 16

Copy the existing service YANG model to your new service package from the *~/packages/svi* folder.

This data model has already been created and serves you as base for your new service.

```
student@student-vm:~/nso300$ cp ~/packages/svi/src/yang/svi.yang
packages/svi/src/yang/svi.yang
```

Step 17

Open the service YANG model.

This is how the YANG module should initially appear.

```
student@student-vm:~/nso300$ vim packages/svi/src/yang/svi.yang
module svi {
  namespace "http://example.com/svi";
  prefix svi;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-common {
    prefix tailf;
  }
  import tailf-ncs {
    prefix ncs;
  }

  description
    "YANG model for SVI service";

  revision 2020-07-21 {
    description
      "Initial revision.";
  }

  augment "/ncs:services" {

    list svi {
      key "name";

      leaf name {
        tailf:info "Unique service id";
        tailf:cli-allow-range;
        type string;
      }
      uses ncs:service-data;
      ncs:servicepoint "svi-servicepoint";

      leaf vlan-id {
        tailf:info "Unique VLAN ID";
        mandatory true;
        type uint32 {
          range "1..4096";
        }
      }

      list device {
        tailf:info "L3 switch";
        key "name";

        leaf name {
          tailf:info "Device name";
          type leafref {
            path "/ncs:devices/ncs:device/ncs:name";
          }
        }

        leaf ip-prefix {
          tailf:info "Unique IPv4 prefix for VLAN";
          type inet:ip-prefix;
        }

        list interface {
          tailf:info "Ethernet interface";
          key "intf-type intf-id";

          leaf intf-type {
            tailf:info "Ethernet interface type";
            type enumeration {
              enum Ethernet;
              enum FastEthernet;
              enum GigabitEthernet;
            }
          }

          leaf intf-id {
            tailf:info "Ethernet interface ID";
            type string;
          }
        }
      }
    }
  }
}
```

Try to reuse old packages as much as you can. Reusing saves you time, but it also provides you with a well-tested and production-proven source.

Step 18

Restrict device selection to switches only.

This constraint makes sure that your service can only be used on devices with specific names. The device name comes from the device as registered in NSO and is not to be confused with the device *hostname* configuration setting.

```
module svi {
  namespace "http://example.com/svi";
  prefix svi;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-common {
    prefix tailf;
  }
  import tailf-ncs {
    prefix ncs;
  }

  description
    "YANG model for SVI service";

  revision 2020-07-21 {
    description
      "Initial revision.";
  }

  augment "/ncs:services" {

    list svi {
      key "name";

      leaf name {
        tailf:info "Unique service id";
        tailf:cli-allow-range;
        type string;
      }
      uses ncs:service-data;
      ncs:servicepoint "svi-servicepoint";

      leaf vlan-id {
        tailf:info "Unique VLAN ID";
        mandatory true;
        type uint32 {
          range "1..4096";
        }
      }

      list device {
        tailf:info "L3 switch";
        key "name";

        leaf name {
          tailf:info "Device name";
          type leafref {
            path "/ncs:devices/ncs:device/ncs:name";
          }
          must "starts-with(current(),'SW')" {
            error-message "Only SW devices can be selected.";
          }
        }

        leaf ip-prefix {
          tailf:info "Unique IPv4 prefix for VLAN";
          type inet:ip-prefix;
        }

        list interface {
          tailf:info "Ethernet interface";
          key "intf-type intf-id";

          leaf intf-type {
            tailf:info "Ethernet interface type";
            type enumeration {
              enum Ethernet;
              enum FastEthernet;
              enum GigabitEthernet;
            }
          }

          leaf intf-id {
            tailf:info "Ethernet interface ID";
            type string;
          }
        }
      }
    }
  }
}
```

Step 19

Create a *vlan-id-cnt* leaf at the end of the YANG module.

This leaf serves as a counter and is updated only by your Python code inside the *pre_modification* callback. It always holds the *vlan-id* number for your next service instance.

```
module svi {
  namespace "http://example.com/svi";
  prefix svi;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-common {
    prefix tailf;
  }
  import tailf-ncs {
    prefix ncs;
  }

  description
    "YANG model for SVI service";

  revision 2020-07-21 {
    description
      "Initial revision.";
  }

  augment "/ncs:services" {

    list svi {
      key "name";

      leaf name {
        tailf:info "Unique service id";
        tailf:cli-allow-range;
        type string;
      }
      uses ncs:service-data;
      ncs:servicepoint "svi-servicepoint";

      leaf vlan-id {
        tailf:info "Unique VLAN ID";
        mandatory true;
        type uint32 {
          range "1..4096";
        }
      }

      list device {
        tailf:info "L3 switch";
        key "name";

        leaf name {
          tailf:info "Device name";
          type leafref {
            path "/ncs:devices/ncs:device/ncs:name";
          }
          must "starts-with(current(),'SW')" {
            error-message "Only SW devices can be selected.";
          }
        }

        leaf ip-prefix {
          tailf:info "Unique IPv4 prefix for VLAN";
          type inet:ip-prefix;
        }

        list interface {
          tailf:info "Ethernet interface";
          key "intf-type intf-id";

          leaf intf-type {
            tailf:info "Ethernet interface type";
            type enumeration {
              enum Ethernet;
              enum FastEthernet;
              enum GigabitEthernet;
            }
          }

          leaf intf-id {
            tailf:info "Ethernet interface ID";
            type string;
          }
        }
      }
    }
  }
  augment "/ncs:services" {
    leaf vlan-id-cnt {
      description
        "Provides a unique number used as VLAN identifier";
      tailf:hidden "Counter";
      type uint32 {
        range "2..4096";
      }
      default "2";
    }
  }
}
```

Step 20

Look for the leaf node vlan-id and delete it.

This leaf is no longer needed since the vlan-id-cnt counter is used as source.

```
    leaf vlan-id {
      tailf:info "Unique VLAN ID";
      mandatory true;
      type uint32 {
        range "1..4096";
      }
    }
```

Step 21

Restrict the leaf *ip-prefix* to be configured on one of the devices only.

```
module svi {

  namespace "http://Example.com/svi";
  prefix svi;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-common {
    prefix tailf;
  }
  import tailf-ncs {
    prefix ncs;
  }

  description
    "YANG model for SVI service";

  revision 2020-07-21 {
    description
      "Initial revision.";
  }

  augment "/ncs:services" {

    list svi {
      key "name";

      leaf name {
        tailf:info "Unique service id";
        tailf:cli-allow-range;
        type string;
      }
      uses ncs:service-data;
      ncs:servicepoint "svi-servicepoint";

      list device {
        tailf:info "L3 switch";
        key "name";

        leaf name {
          tailf:info "Device name";
          type leafref {
            path "/ncs:devices/ncs:device/ncs:name";
          }
          must "starts-with(current(),'SW')" {
            error-message "Only SW devices can be selected.";
          }
        }

        leaf ip-prefix {
          tailf:info "Unique IPv4 prefix for VLAN. Device with ip-
prefix configured will serve as gateway.";
          type inet:ipv4-prefix;
          // Only one device can have ip-prefix configured
          when "count(../../device[name != current()/../name]/ip-
prefix)=0";
        }

        list interface {
          tailf:info "Ethernet interface";
          key "intf-type intf-id";

          leaf intf-type {
            tailf:info "Ethernet interface type";
            type enumeration {
              enum Ethernet;
              enum FastEthernet;
              enum GigabitEthernet;
            }
          }

          leaf intf-id {
            tailf:info "Ethernet interface ID";
            type string;
          }
        }
      }
    }
  }

  augment "/ncs:services" {
    leaf vlan-id-cnt {
      description
        "Provides a unique number used as VLAN identifier";
      tailf:hidden "Counter";
      type uint32 {
        range "2..4096";
      }
      default "2";
    }
  }
}
```

Step 22

Save the file when finished.

Activity Verification

You have completed this task when you attain this result:

- You successfully modified the YANG service model.

Task 2: Implement the Python Mapping Code

In this task, you will implement some Python code for configuration mapping. Files from an existing service have been provided for you in the *$HOME/packages/svi* directory.

NSO will call your *pre_modification* method before it calls the *cb_create* method. Therefore, you can prepare and format data.

When the *pre_modification* returns, your data can be passed to *cb_create* through the *proplist* object. *proplist* is a Python list of tuples, and each tuple consists of two members—a name and a value.

The old service package used the *netaddr* Python module, which is not a part of the standard library and therefore is not present in any of the containers. You will replace this module with the *ipaddress* module and make minor adjustments.

Activity

Complete these steps:

Step 1

Copy the existing Python-mapping code.

```
student@student-vm:~/nso300$ cp ~/packages/svi/python/svi/svi.py
packages/svi/python/svi/svi.py
```

Step 2

Open Python mapping code for the svi service.

This is how the Python code should initially appear:

```
student@student-vm:~/nso300$ vim packages/svi/python/svi/svi.py
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import netaddr


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    # The create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Service create(service={service._path})')
        svi = {'vlan-id': "",
               'svi-device': "",
               'ip-prefix': "",
               'ip-addr': "",
               'netmask': ""}

        svi['vlan-id'] = service.vlan_id

        for device in service.device:
            self.log.info(f'Entering /device list = {device.name}')
            if device.ip_prefix:
                self.log.info(f'SVI device = {device.name}')

                ip_net = netaddr.IPNetwork(device.ip_prefix)
                svi['svi-device'] = device.name
                svi['ip-prefix'] = f'{ip_net[2]}/{ip_net.prefixlen}'
                svi['ip-addr'] = str(ip_net[1])
                svi['netmask'] = str(ip_net.netmask)

                svi_tvars = ncs.template.Variables()
                svi_tvars.add('VLAN-ID', svi['vlan-id'])
                svi_tvars.add('SVI-DEVICE', svi['svi-device'])
                svi_tvars.add('IP-PREFIX', svi['ip-prefix'])
                svi_tvars.add('IP-ADDR', svi['ip-addr'])
                svi_tvars.add('NETMASK', svi['netmask'])
                svi_template = ncs.template.Template(service)
                svi_template.apply('svi-intf-template', svi_tvars)

        vlan_tvars = ncs.template.Variables()
        vlan_tvars.add('VLAN-ID', svi['vlan-id'])
        vlan_template = ncs.template.Template(service)
        vlan_template.apply('svi-vlan-template', vlan_tvars)

    # The pre_modification() and post_modification() callbacks are
optional,
    # and are invoked outside FASTMAP. pre_modification() is invoked
before
    # create, update, or delete of the service, as indicated by the
enum
    # ncs_service_operation op parameter. Conversely
    # post_modification() is invoked after create, update, or delete
    # of the service. These functions can be useful e.g. for
    # allocations that should be stored and existing also when the
    # service instance is removed.

    # @Service.pre_lock_create
    # def cb_pre_lock_create(self, tctx, root, service, proplist):
    #     self.log.info('Service plcreate(service=', service._path,
')')

    # @Service.pre_modification
    # def cb_pre_modification(self, tctx, op, kp, root, proplist):
    #     self.log.info('Service premod(service=', kp, ')')

    # @Service.post_modification
    # def cb_post_modification(self, tctx, op, kp, root, proplist):
    #     self.log.info('Service premod(service=', kp, ')')


# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Svi(ncs.application.Application):
    def setup(self):
        # The application class sets up logging for us. It is
accessible
        # through 'self.log' and is a ncs.log.Log instance.
        self.log.info('Main RUNNING')

        # Service callbacks require a registration for a 'service
point',
        # as specified in the corresponding data model.
        #
        self.register_service('svi-servicepoint', ServiceCallbacks)

        # If we registered any callback(s) above, the Application class
        # took care of creating a daemon (related to the service/action
point).

        # When this setup method is finished, all registrations are
        # considered done and the application is 'started'.

    def teardown(self):
        # When the application is finished (which would happen if NCS
went
        # down, packages were reloaded or some error occurred) this
teardown
        # method will be called.

        self.log.info('Main FINISHED')
```

You can also use an IDE tool or text editor of your choice.

Step 3

Delete all the comments to improve readability.

You can keep the *pre_modification* definition and decorator.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import netaddr


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Service create(service={service._path})')
        svi = {'vlan-id': "",
               'svi-device': "",
               'ip-prefix': "",
               'ip-addr': "",
               'netmask': ""}

        svi['vlan-id'] = service.vlan_id

        for device in service.device:
            self.log.info(f'Entering /device list = {device.name}')
            if device.ip_prefix:
                self.log.info(f'SVI device = {device.name}')

                ip_net = netaddr.IPNetwork(device.ip_prefix)
                svi['svi-device'] = device.name
                svi['ip-prefix'] = f'{ip_net[2]}/{ip_net.prefixlen}'
                svi['ip-addr'] = str(ip_net[1])
                svi['netmask'] = str(ip_net.netmask)

                svi_tvars = ncs.template.Variables()
                svi_tvars.add('VLAN-ID', svi['vlan-id'])
                svi_tvars.add('SVI-DEVICE', svi['svi-device'])
                svi_tvars.add('IP-PREFIX', svi['ip-prefix'])
                svi_tvars.add('IP-ADDR', svi['ip-addr'])
                svi_tvars.add('NETMASK', svi['netmask'])
                svi_template = ncs.template.Template(service)
                svi_template.apply('svi-intf-template', svi_tvars)

        vlan_tvars = ncs.template.Variables()
        vlan_tvars.add('VLAN-ID', svi['vlan-id'])
        vlan_template = ncs.template.Template(service)
        vlan_template.apply('svi-vlan-template', vlan_tvars)

    # @Service.pre_modification
    # def cb_pre_modification(self, tctx, op, kp, root, proplist):
    #     self.log.info('Service premod(service=', kp, ')')

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Svi(ncs.application.Application):
    def setup(self):
        self.log.info('Main RUNNING')
        self.register_service('svi-servicepoint', ServiceCallbacks)
    def teardown(self):
        self.log.info('Main FINISHED')
```

Step 4

Implement the *pre_modification* callback method.

If this is a service creation, then assign the value *vlan-id-cnt* to a local variable *vlan_id*, increment the counter by one, and append it to the *proplist* variable before returning.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import netaddr


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Service create(service={service._path})')
        svi = {'vlan-id': "",
               'svi-device': "",
               'ip-prefix': "",
               'ip-addr': "",
               'netmask': ""}

        svi['vlan-id'] = service.vlan_id

        for device in service.device:
            self.log.info(f'Entering /device list = {device.name}')
            if device.ip_prefix:
                self.log.info(f'SVI device = {device.name}')
                ip_net = netaddr.IPNetwork(device.ip_prefix)
                svi['svi-device'] = device.name
                svi['ip-prefix'] = f'{ip_net[2]}/{ip_net.prefixlen}'
                svi['ip-addr'] = str(ip_net[1])
                svi['netmask'] = str(ip_net.netmask)

                svi_tvars = ncs.template.Variables()
                svi_tvars.add('VLAN-ID', svi['vlan-id'])
                svi_tvars.add('SVI-DEVICE', svi['svi-device'])
                svi_tvars.add('IP-PREFIX', svi['ip-prefix'])
                svi_tvars.add('IP-ADDR', svi['ip-addr'])
                svi_tvars.add('NETMASK', svi['netmask'])
                svi_template = ncs.template.Template(service)
                svi_template.apply('svi-intf-template', svi_tvars)

        vlan_tvars = ncs.template.Variables()
        vlan_tvars.add('VLAN-ID', svi['vlan-id'])
        vlan_template = ncs.template.Template(service)
        vlan_template.apply('svi-vlan-template', vlan_tvars)

    @Service.pre_modification
    def cb_pre_modification(self, tctx, op, kp, root, proplist):
        self.log.info(f'Service premod(service={kp})')
        if op == ncs.dp.NCS_SERVICE_CREATE:
            self.log.info('Service premod(operation=NCS_SERVICE_CREATE,
allocate)')
            vlan_id = root.services.vlan_id_cnt
            proplist.append(('vlan-id', str(vlan_id)))
            self.log.info(f'Service premod(allocated vlan-id:
{vlan_id})')
            root.services.vlan_id_cnt = vlan_id + 1

        elif op == ncs.dp.NCS_SERVICE_DELETE:
            self.log.info('Service premod(operation=NCS_SERVICE_DELETE,
skip)')

        return proplist

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Svi(ncs.application.Application):
    def setup(self):
        self.log.info('Main RUNNING')
        self.register_service('svi-servicepoint', ServiceCallbacks)
    def teardown(self):
        self.log.info('Main FINISHED')
```

Step 5

Remove the following line from the *cb_create* method because the vlan-id will be passed through the *proplist* object.

```
svi['vlan-id'] = service.vlan_id
```

Step 6

Update the *cb_create* code. Retrieve the *vlan-id* as returned from the *pre_modification* callback inside the *proplist* object.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import netaddr


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Service create(service={service._path})')
        svi = {'vlan-id': "",
               'svi-device': "",
               'ip-prefix': "",
               'ip-addr': "",
               'netmask': ""}

        # proplist object list(tuple(str,str)) to pass information
between invocations. We set
        # value for vlan_id in cb_pre_modification and use it here in
cb_create.
        svi['vlan-id'] = [x[1] for x in proplist if x[0] == 'vlan-id']
[0]


        for device in service.device:
            self.log.info(f'Entering /device list = {device.name}')
            if device.ip_prefix:
                self.log.info(f'SVI device = {device.name}')

                ip_net = netaddr.IPNetwork(device.ip_prefix)
                svi['svi-device'] = device.name
                svi['ip-prefix'] = f'{ip_net[2]}/{ip_net.prefixlen}'
                svi['ip-addr'] = str(ip_net[1])
                svi['netmask'] = str(ip_net.netmask)

                svi_tvars = ncs.template.Variables()
                svi_tvars.add('VLAN-ID', svi['vlan-id'])
                svi_tvars.add('SVI-DEVICE', svi['svi-device'])
                svi_tvars.add('IP-PREFIX', svi['ip-prefix'])
                svi_tvars.add('IP-ADDR', svi['ip-addr'])
                svi_tvars.add('NETMASK', svi['netmask'])
                svi_template = ncs.template.Template(service)
                svi_template.apply('svi-intf-template', svi_tvars)

        vlan_tvars = ncs.template.Variables()
        vlan_tvars.add('VLAN-ID', svi['vlan-id'])
        vlan_template = ncs.template.Template(service)
        vlan_template.apply('svi-vlan-template', vlan_tvars)
    @Service.pre_modification
    def cb_pre_modification(self, tctx, op, kp, root, proplist):
        self.log.info(f'Service premod(service={kp})')
        if op == ncs.dp.NCS_SERVICE_CREATE:
            self.log.info('Service premod(operation=NCS_SERVICE_CREATE,
allocate)')
            vlan_id = root.services.vlan_id_cnt
            proplist.append(('vlan-id', str(vlan_id)))
            self.log.info(f'Service premod(allocated vlan-id:
{vlan_id})')
            root.services.vlan_id_cnt = vlan_id + 1

        elif op == ncs.dp.NCS_SERVICE_DELETE:
            self.log.info('Service premod(operation=NCS_SERVICE_DELETE,
skip)')

        return proplist

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Svi(ncs.application.Application):
    def setup(self):
        self.log.info('Main RUNNING')
        self.register_service('svi-servicepoint', ServiceCallbacks)
    def teardown(self):
        self.log.info('Main FINISHED')
```

Step 7

Replace the Python *netaddr* module with the *ipaddress* module, and *IPNetwork* call with ip_*network*.

The Python *netaddr* module is not a part of Python's standard library and does not exist inside the container.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import ipaddress


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Service create(service={service._path})')
        svi = {'vlan-id': "",
               'svi-device': "",
               'ip-prefix': "",
               'ip-addr': "",
               'netmask': ""}
        # proplist object list(tuple(str,str)) to pass information
between invocations. We set
        # value for vlan_id in cb_pre_modification and use it here in
cb_create.
        svi['vlan-id'] = [x[1] for x in proplist if x[0] == 'vlan-id']
[0]


        for device in service.device:
            self.log.info(f'Entering /device list = {device.name}')
            if device.ip_prefix:
                self.log.info(f'SVI device = {device.name}')

                ip_net = ipaddress.ip_network(device.ip_prefix)
                svi['svi-device'] = device.name
                svi['ip-prefix'] = f'{ip_net[2]}/{ip_net.prefixlen}'
                svi['ip-addr'] = str(ip_net[1])
                svi['netmask'] = str(ip_net.netmask)

                svi_tvars = ncs.template.Variables()
                svi_tvars.add('VLAN-ID', svi['vlan-id'])
                svi_tvars.add('SVI-DEVICE', svi['svi-device'])
                svi_tvars.add('IP-PREFIX', svi['ip-prefix'])
                svi_tvars.add('IP-ADDR', svi['ip-addr'])
                svi_tvars.add('NETMASK', svi['netmask'])
                svi_template = ncs.template.Template(service)
                svi_template.apply('svi-intf-template', svi_tvars)

        vlan_tvars = ncs.template.Variables()
        vlan_tvars.add('VLAN-ID', svi['vlan-id'])
        vlan_template = ncs.template.Template(service)
        vlan_template.apply('svi-vlan-template', vlan_tvars)

    @Service.pre_modification
    def cb_pre_modification(self, tctx, op, kp, root, proplist):
        self.log.info(f'Service premod(service={kp})')
        if op == ncs.dp.NCS_SERVICE_CREATE:
            self.log.info('Service premod(operation=NCS_SERVICE_CREATE,
allocate)')
            vlan_id = root.services.vlan_id_cnt
            proplist.append(('vlan-id', str(vlan_id)))
            self.log.info(f'Service premod(allocated vlan-id:
{vlan_id})')
            root.services.vlan_id_cnt = vlan_id + 1

        elif op == ncs.dp.NCS_SERVICE_DELETE:
            self.log.info('Service premod(operation=NCS_SERVICE_DELETE,
skip)')

        return proplist

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Svi(ncs.application.Application):
    def setup(self):
        self.log.info('Main RUNNING')
        self.register_service('svi-servicepoint', ServiceCallbacks)
    def teardown(self):
        self.log.info('Main FINISHED')
```

Step 8

Save the file when finished.

Activity Verification

You have completed this task when you attain this result:

- You have successfully modified the Python mapping code.

Task 3: Compile and Deploy the Service

In this task, you will compile and deploy the service package created in the previous task. But before doing so, you will create a new target for *pylint*.

Before data model compilation, your Python code should be checked for errors. You do this action by running *pylint* on every *.py* file found in your package.

After a successful *pylint* run, your data model must be checked and compiled into binary form. This is achieved by running the NSO compiler. For every *.yang* file found in your package, the compiler will output a corresponding *.fxs* file. These files are compiled versions of your text data models.

All this work is abstracted from you in the form of Makefile targets. Running the **make testenv-build** command will progress to the next target only if there are no errors in the current target.

Only after a successful compilation will the package be complete and ready for deployment. This step is also the last *Makefile* target.

If there is an error in any of the targets, simply identify the cause, make corrections, and try to build again.

Activity

Complete these steps:

Step 1

Open the Makefile with a text editor of your choice.

```
student@student-vm:~/nso300$ vim packages/svi/src/Makefile
```

Step 2

Edit the first line at the very top and add *pylint* at the end.

```
all: fxs pylint
```

```
.PHONY: all
```

Step 3

Scroll down and add a *pylint* target declaration. Make sure you use *TAB* for indentation.

```
< ... Output Omitted ... >

NCSCPATH   = $(YANGPATH:%=--yangpath %)
YANGERPATH = $(YANGPATH:%=--path %)

pylint:
	pylint --disable=R,C --reports=n ../python/svi/*.py  || (test $
$? -ge 4)

fxs: $(DIRS) $(FXS)

< ... Output Omitted ... >
```

Step 4

Save changes and exit the text editor.

Step 5

Compile the package. Use the **make testenv-build** command. This command recompiles and reloads or redeploys the packages, used by the Docker NSO containers.

Inspect the output and make sure that no errors are present.

Notice the pylint lines in the output. Linter checks your code for errors and provides a score, which at this point should be of no concern.

```
student@student-vm:~/nso300$ make testenv-build
for NSO in $(docker ps --format '{{.Names}}' --filter label=testenv-
nso300-5.3.2-student --filter label=nidtype=nso); do \
    echo "-- Rebuilding for NSO: ${NSO}"; \
    docker run -it --rm -v /home/student/nso300:/src --volumes-from
${NSO} --network=container:${NSO} -e NSO=${NSO} -e PACKAGE_RELOAD= -e
SKIP_LINT= -e PKG_FILE=nso300.gitlab.local/nso300/package:5.3.2-student
nso300.gitlab.local/cisco-nso-dev:5.3.2 /src/nid/testenv-build; \
done
-- Rebuilding for NSO: testenv-nso300-5.3.2-student-nso
(package-meta-data.xml|\.cli1|\.yang1)
make: Entering directory '/var/opt/ncs/packages/svi/src'
pylint --disable=R,C --reports=n ../python/svi/svi.py || (test $? -ge
4)
************* Module svi.svi
/var/opt/ncs/packages/svi/python/svi/svi.py:16:48: W0212: Access to a
protected member _path of a client class (protected-access)
```

```
-----------------------------------
Your code has been rated at 9.78/10

pylint --disable=R,C --reports=n ../python/svi/__init__.py || (test $?
-ge 4)
mkdir -p ../load-dir
mkdir -p java/src//
/opt/ncs/ncs-5.3.2/bin/ncsc  `ls svi-ann.yang  > /dev/null 2>&1 && echo
"-a svi-ann.yang"` \
              -c -o ../load-dir/svi.fxs yang/svi.yang
make: Leaving directory '/var/opt/ncs/packages/svi/src'
make: Entering directory '/src/packages/svi'
if [ ! -f build-meta-data.xml ]; then \
    export PKG_NAME=$(xmlstarlet sel -N x=http://tail-f.com/ns/ncs-
packages -t -v '/x:ncs-package/x:name' $(ls package-meta-data.xml src/
package-meta-data.xml.in 2>/dev/null | head -n 1)); \
    export PKG_VERSION=$(xmlstarlet sel -N x=http://tail-f.com/ns/ncs-
packages -t -v '/x:ncs-package/x:package-version' $(ls package-meta-
data.xml src/package-meta-data.xml.in 2>/dev/null | head -n 1)); \
    eval "cat <<< \"$(</src/nid/build-meta-data.xml)\"" > /var/opt/ncs/
packages/svi/build-meta-data.xml; fi
make: Leaving directory '/src/packages/svi'
-- Reloading packages for NSO testenv-nso300-5.3.2-student-nso

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has
completed.
>>> System upgrade has completed successfully.
reload-result {
    package cisco-asa-cli-6.10
    result true
}
reload-result {
    package cisco-ios-cli-6.54
    result true
}
reload-result {
    package cisco-iosxr-cli-7.26
    result true
}
reload-result {
    package cisco-nx-cli-5.15
    result true
}
reload-result {
    package svi
    result true
}
```

Step 6

Connect to the NSO CLI and switch to the *Cisco* mode.

```
student@student-vm:~/nso300$ make testenv-cli
docker exec -it testenv-nso300-5.3.2-student-nso bash -lc 'ncs_cli -u
```

```
admin'

admin connected from 127.0.0.1 using console on c79dc518a316
admin@ncs> switch cli
admin@ncs#
```

Step 7

Configure the service instance.

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services svi ACME
admin@ncs(config-svi-ACME)# device SW31
admin@ncs(config-device-SW31)# ip-prefix 10.10.0.0/16
admin@ncs(config-device-SW31)# interface GigabitEthernet 0/1
admin@ncs(config-interface-GigabitEthernet/0/1)# exit
admin@ncs(config-device-SW31)# exit
admin@ncs(config-svi-ACME)# device SW32
admin@ncs(config-device-SW32)# interface Ethernet 0/2
admin@ncs(config-interface-Ethernet/0/2)# top
admin@ncs(config)# show configuration
services svi ACME
 device SW31
  ip-prefix 10.10.0.0/16
  interface GigabitEthernet 0/1
  !
 !
 device SW32
  interface Ethernet 0/2
  !
 !
!
admin@ncs(config)#
```

Step 8

Review the configuration that will be created using the **commit dry-run** command.

```
admin@ncs(config)# commit dry-run
cli {
    local-node {
        data  devices {
                  device SW31 {
                      config {
                          vlan {
             +                vlan-list 2 {
             +                }
                          }
                          interface {
                              GigabitEthernet 0/1 {
             +                    switchport {
             +                        mode {
             +                            access {
```

```
             +                            }
             +                        }
             +                        access {
             +                            vlan 2;
             +                        }
             +                    }
                                  ip {
                                      no-address {
             -                            address false;
                                      }
                                  }
                              }
             +                Vlan 2 {
             +                    ip {
             +                        address {
             +                            primary {
             +                                address 10.10.0.1;
             +                                mask 255.255.0.0;
             +                            }
             +                        }
             +                    }
             +                }
                          }
                      }
                  }
                  device SW32 {
                      config {
                          vlan {
             +                vlan-list 2 {
             +                }
                          }
                          interface {
             +                Ethernet 0/2 {
             +                    switchport {
             +                        mode access;
             +                        access {
             +                            vlan 2;
             +                        }
             +                    }
             +                }
                          }
                      }
                  }
              }
              services {
             +    svi ACME {
             +        device SW31 {
             +            ip-prefix 10.10.0.0/16;
             +            interface GigabitEthernet 0/1;
             +        }
             +        device SW32 {
             +            interface Ethernet 0/2;
             +        }
             +    }
              }
    }
}
```

Step 9

Commit the configuration to deploy the service.

Your service should be configured with *vlan-id 2*.

```
admin@ncs(config)# commit
Commit complete.
```

Step 10

Exit the privileged mode and exit the NSO CLI.

```
admin@ncs(config)# exit
admin@ncs# exit
student@student-vm:~/nso300$
```

Activity Verification

You have completed this task when you attain this result:

- You have successfully deployed a service instance.

Task 4: Use the VS Code Remote Debugger

In this task, you will use Microsoft Visual Studio Code to debug your Python code that is executing inside a container.

You will also put a breakpoint in place. Breakpoints inform the debugger to pause execution and pass the control over to you. This enables you to pause execution and inspect the run time environment. You can then step through the lines slowly and see what is happening in real time

Microsoft Visual Studio Code will act as a client that connects to a running container where the *debugpy* server will be listening on port 5678.

Activity

Complete these steps:

Step 1

From the Terminal window, run the **make testenv-debug-vscode** command. This command generates a *launch.json* file.

```
student@student-vm:~/nso300$ make testenv-debug-vscode
== Created .vscode/launch.json
== Added "Python: NID Remote Attach" debug configuration
student@student-vm:~/nso300$
```

Step 2

View the contents of the newly created file.

Notice the TCP port on the host machine which is mapped to container port 5678 where the debugger will be listening.

```
student@student-vm:~/nso300$ cat .vscode/launch.json
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: NID Remote Attach",
      "type": "python",
      "request": "attach",
      "port": 32792,
      "host": "localhost",
      "pathMappings": [
        {
          "localRoot": "${workspaceFolder}/packages",
          "remoteRoot": "/nso/run/state/packages-in-use.cur/1"
        }
      ]
    }
  ]
}
student@student-vm:~/nso300$
```

Port mapping for debugger is automatically assigned by the Docker and might differ from the one show here.

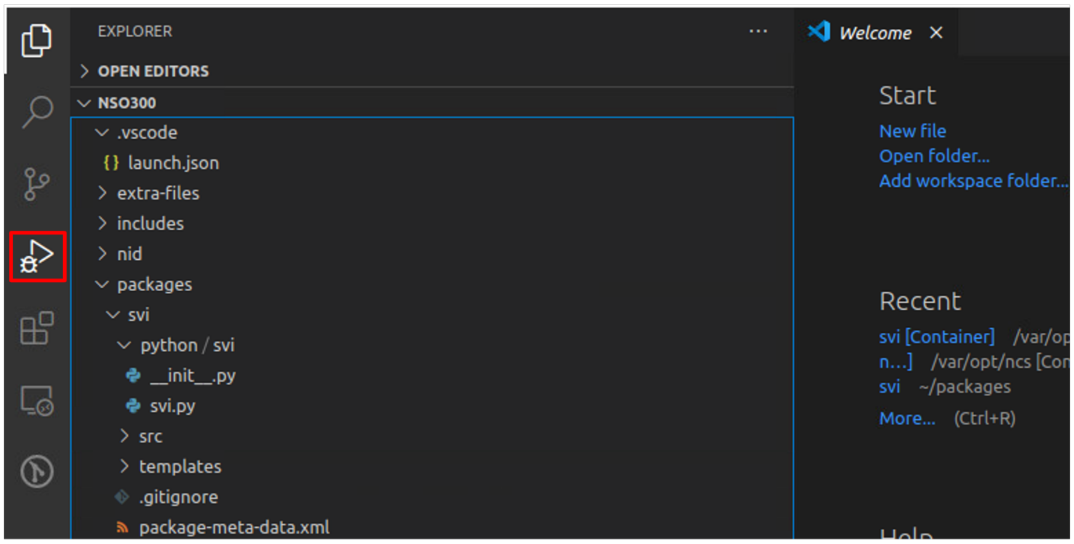
Step 3

Open the current directory with Visual Studio Code editor.

```
student@student-vm:~/nso300$ code .
student@student-vm:~/nso300$
```

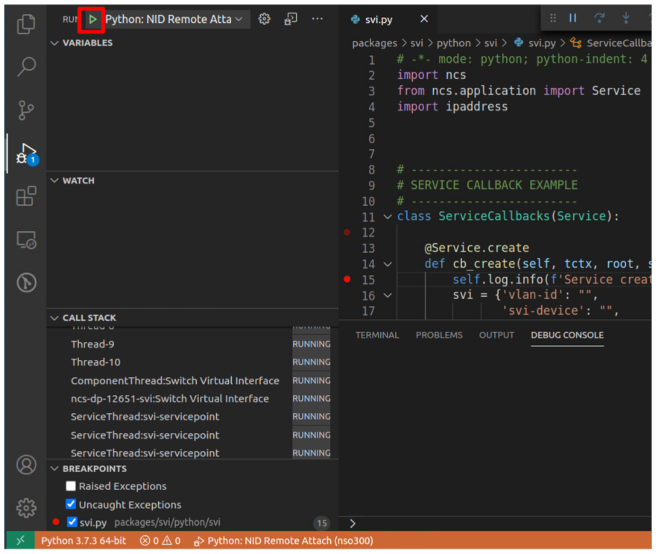
Step 4

On the activity bar on the left of the interface, click **Run**.



Step 5

Press *F5* or click the play icon at the top to attach to the debugger running inside the container.


RUN
Python: NID Remote Atta
VARIABLES
WATCH
CALL STACK
Thread-9 RUNNING
Thread-10 RUNNING
ComponentThread:Switch Virtual Interface RUNNING
ncs-dp-12651-svi:Switch Virtual Interface RUNNING
ServiceThread:svi-servicepoint RUNNING
ServiceThread:svi-servicepoint RUNNING
ServiceThread:svi-servicepoint RUNNING
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
svi.py packages/svi/python/svi 15
svi.py
packages > svi > python > svi > svi.py > ServiceCallba
TERMINAL
PROBLEMS
OUTPUT
DEBUG CONSOLE
Python 3.7.3 64-bit
Python: NID Remote Attach (nso300)

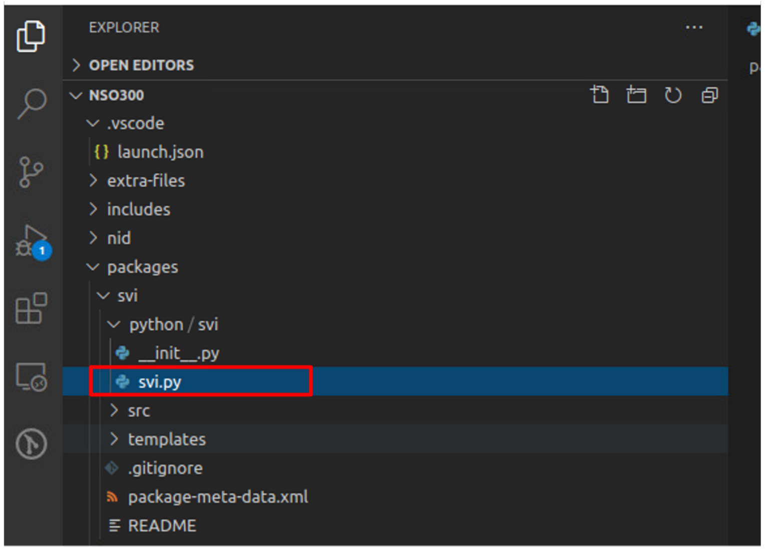
```
# -*- mode: python; python-indent: 4
import ncs
from ncs.application import Service
import ipaddress



# -----------------------
# SERVICE CALLBACK EXAMPLE
# -----------------------
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, s
        self.log.info(f'Service creat
        svi = {'vlan-id': "",
               'svi-device': "",
```

Notice how the bottom bar changes its color to orange, indicating that debugger is now attached.

Step 6

From the top menu, choose **View** > **Explorer** to open the Explorer window. Expand the **packages** > **svi** > **python** > **svi** folder and click the **svi.py** file.

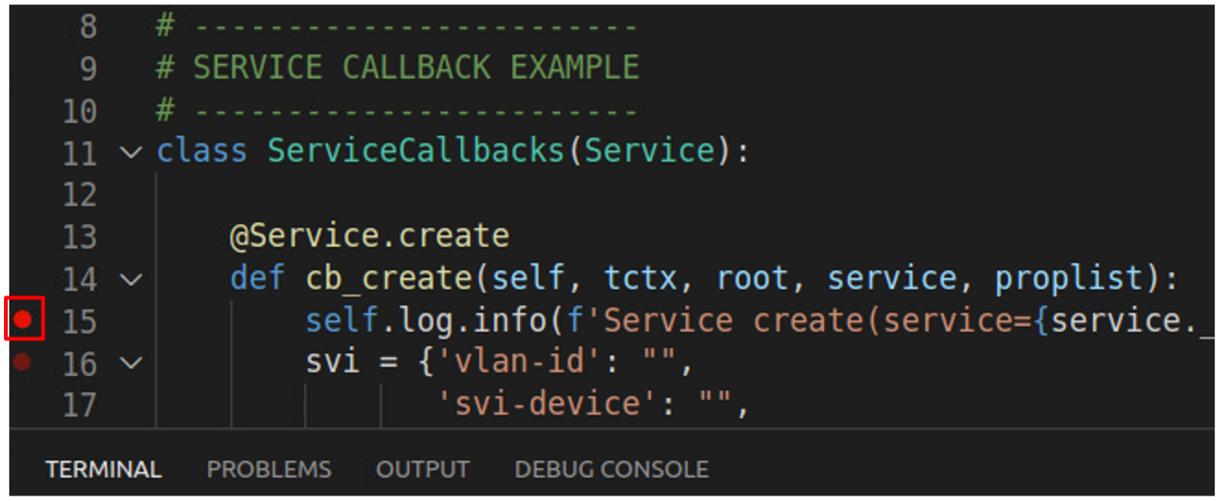
This action opens the file for editing in the right pane.



Step 7

Create a breakpoint for the line that contains the logging statement (or any other lines you wish to debug) inside the *cb_create* function. To do this action, click just left of the line number. A red dot is shown to represent the breakpoint.

This makes the Python interpreter pause, giving you time to inspect the *runtime* environment.

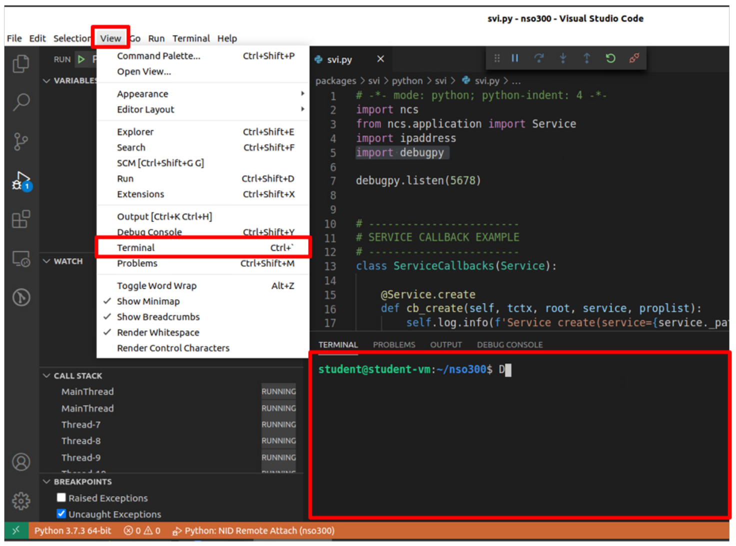


```
# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Service create(service={service._
        svi = {'vlan-id': "",
               'svi-device': "",
```

Step 8

Open the VS Code integrated terminal by clicking on the **View** > **Terminal** on the top menu bar.

The terminal appears at the bottom of the window, just under the currently open files.

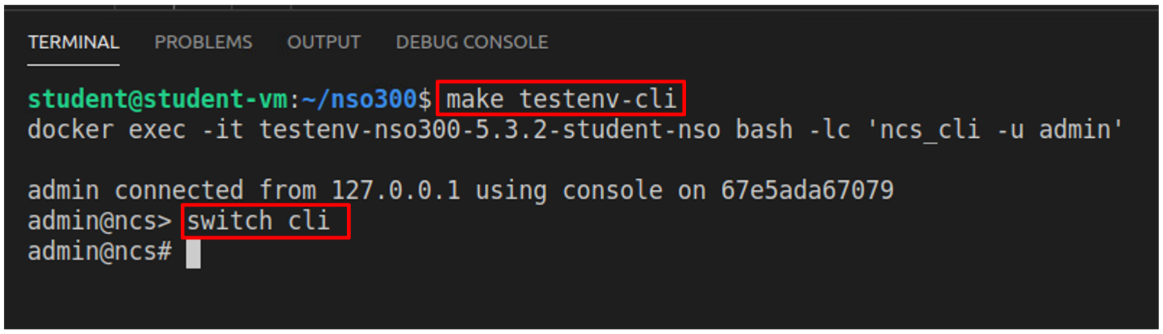


```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import ipaddress
import debugpy

debugpy.listen(5678)


# -----------------------
# SERVICE CALLBACK EXAMPLE
# -----------------------
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Service create(service={service._pa
```

```
student@student-vm:~/nso300$ D
```

Step 9

From the integrated terminal, enter the NSO CLI and switch to the Cisco style CLI.



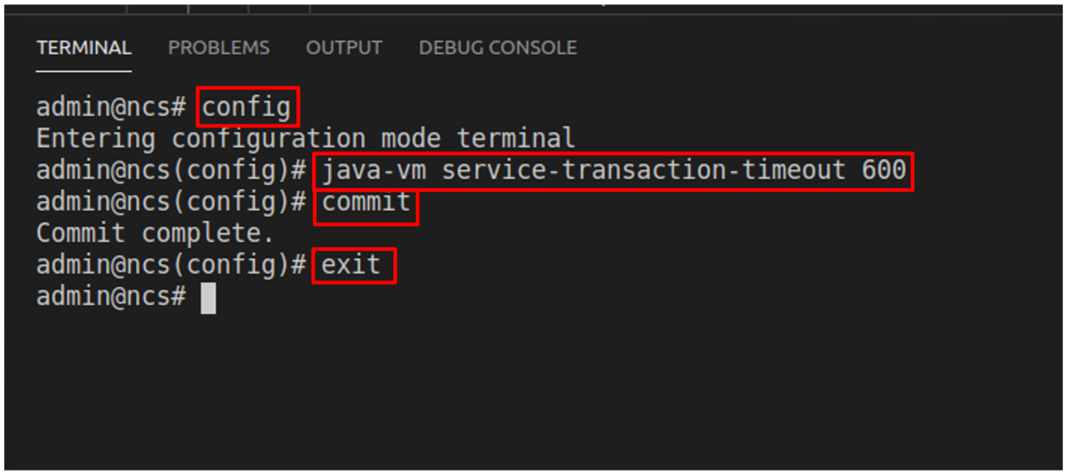
```
student@student-vm:~/nso300$ make testenv-cli
docker exec -it testenv-nso300-5.3.2-student-nso bash -lc 'ncs_cli -u admin'

admin connected from 127.0.0.1 using console on 67e5ada67079
admin@ncs> switch cli
admin@ncs#
```

Step 10

Enter the config mode and increase the service transaction timeout to 10 minutes.

This action enables you to pause execution for more than 2 minutes, which is the default.

Note that time is specified in seconds.



```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# java-vm service-transaction-timeout 600
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# exit
admin@ncs#
```

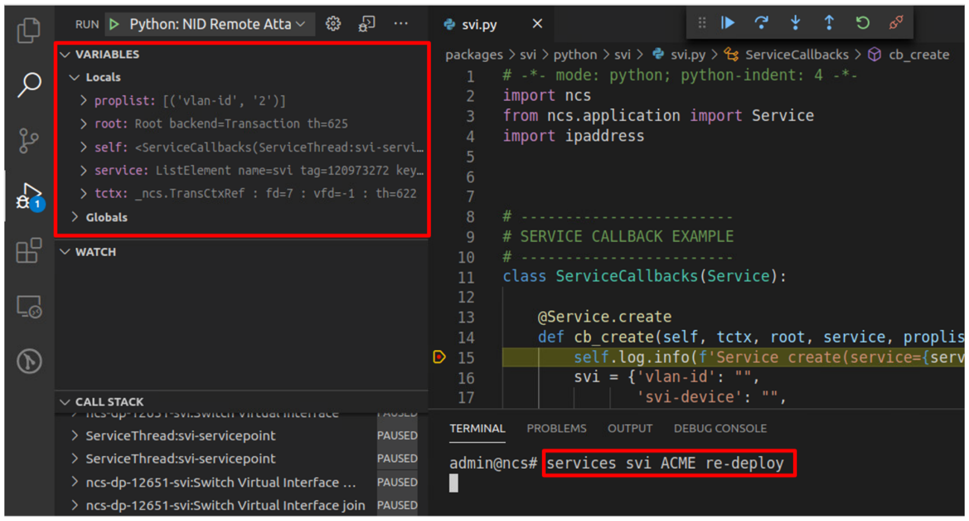
Step 11

Ensure that the run pane is open. To do this, choose the **View > Run** from the top menu.

Step 12

Redeploy your package code by running the command **services svi ACME re-deploy**.

The system will appear unresponsive, but for the next 10 minutes, you are controlling the execution. Call stack and run time variables will appear on the left side where you can view and modify them.



```
VARIABLES
  Locals
    > proplist: [('vlan-id', '2')]
    > root: Root backend=Transaction th=625
    > self: <ServiceCallbacks(ServiceThread:svi-servi…
    > service: ListElement name=svi tag=120973272 key…
    > tctx: _ncs.TransCtxRef : fd=7 : vfd=-1 : th=622
  > Globals
```

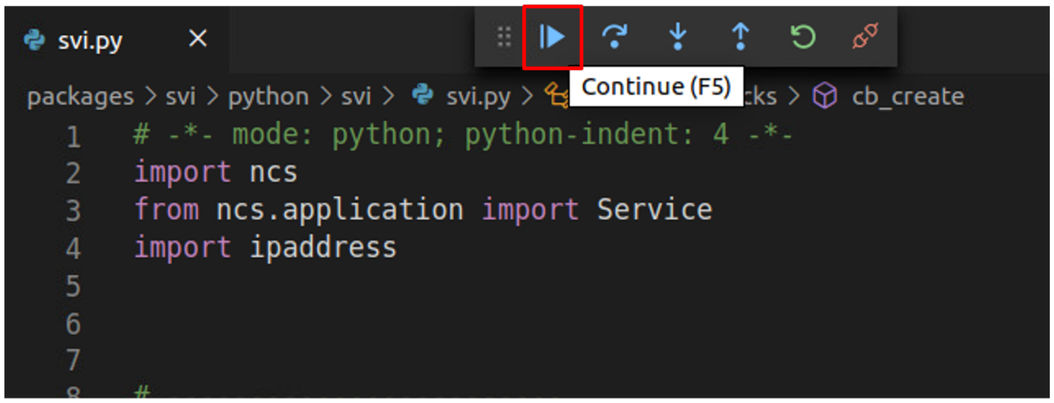
```
CALL STACK
> ServiceThread:svi-servicepoint                          PAUSED
> ServiceThread:svi-servicepoint                          PAUSED
> ncs-dp-12651-svi:Switch Virtual Interface ...           PAUSED
> ncs-dp-12651-svi:Switch Virtual Interface join          PAUSED
```

```python
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import ipaddress



# -----------------------
# SERVICE CALLBACK EXAMPLE
# -----------------------
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplis
        self.log.info(f'Service create(service={serv
        svi = {'vlan-id': "",
               'svi-device': "",
```

```
admin@ncs# services svi ACME re-deploy
```

Step 13

Press *F5* again or click the icon at the top to continue execution.



```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import ipaddress



#
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully configured the debugger.
- You have successfully started the debugger.
- You understand how breakpoints work and how to use them.
- You know where to find the values of run-time variables.
- You know how to increase the service timeout for longer debugging sessions.