Implement Self-Test Action

Introduction

In this activity, you will implement a self-test action available on the static route package. The action will then be used by operators to ping any IP address configured with static route action to verify if the next-hop connectivity exists.

You will add the appropriate changes to the service model and implement the action in Python. This action will use another action defined in the Cisco IOS NED that allows executing a **ping** command directly on the device.

The action will examine the output of the **ping** command and set success.

After completing this activity, you will be able to:

- Design, implement, and call a service action

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

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**
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*

Formatting	Description and Examples
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
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.

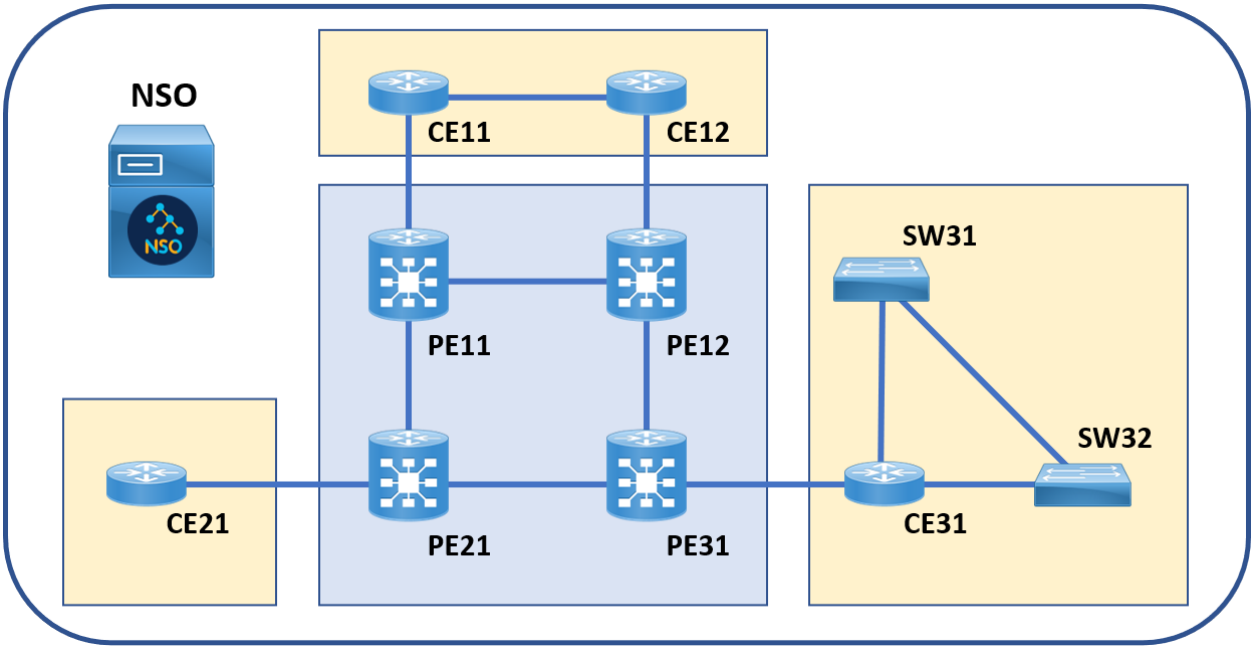
Command	Comment
make testenv-stop	Stops the NSO Docker environment.
make testenv-build	Recompiles and reloads the NSO packages.
make testenv-cli	Enters the NSO CLI of the NSO Docker container.
make testenv-shell	Enters the Linux shell of the NSO Docker container.
make dev-shell	Enters the Linux shell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

Lab Topology Information

Your lab session is your own personal sandbox. Whatever 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 System container with your NSO installation, together with numerous Docker containers 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.

- 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. Devices will, however, accept or reject configurations sent by the NSO, just as real devices would.

Topology



Task 1: Design a Service Action

In this task, you will design a service action in the YANG model for an existing service and implement it using Python code.

The final solutions for all labs, including this lab, are in the ~/solutions directory. You can use them for copy-pasting 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

Go to the **nso300** folder.

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

Step 4

Copy the existing *static-route* package from the *~/packages/static-route* folder to the *packages* folder of the *nso300* project.

```
student@student-vm:~/nso300$ cp -r ~/packages/static-route packages/
student@student-vm:~/nso300$
```

Step 5

Enter the Linux shell of the NSO System container, using the **make testenv-shell** command.

```
student@student-vm:~/nso300$ make testenv-shell
docker exec -it testenv-nso300-5.3.2-student-nso bash -l
root@e7035af9b837:/#
```

Step 6

Examine the **exec** action definition in the Cisco IOS NED. The file **tailf-ned-cisco-ios-stats.yang** is located in the **/var/opt/ncs/packages/cisco-ios-cli-6.54/src/yang** folder.

```
root@e7035af9b837:/# cat /var/opt/ncs/packages/cisco-ios-cli-6.54/src/
yang/tailf-ned-cisco-ios-stats.yang
```

The model for **ping** command should look like this:

```
< … output omitted … >

    // ping [arg 1] .. [arg N]
    tailf:action ping {
      tailf:info "Send echo messages";
      tailf:actionpoint ncsinternal {
        tailf:internal;
      }
      input {
        list auto-prompts {
          tailf:cli-suppress-mode;
          key question;
          leaf question {
            type string;
          }
          leaf answer {
            type string;
          }
        }
        leaf-list args {
          tailf:cli-drop-node-name;
          tailf:cli-flat-list-syntax;
          type string {
            tailf:info "ping argument(s)";
          }
        }
      }
      output {
        leaf result {
          type string;
        }
      }
    }

< … output omitted … >
```

Step 7

Exit the NSO System container.

```
root@e7035af9b837:/# exit
logout
student@student-vm:~/nso300$
```

Step 8

Enter NSO CLI by using the **make testenv-cli** command.

```
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 e7035af9b837
admin@ncs>
```

Step 9

Verify that the ping action works on the NetSim device when invoked through the Cisco NSO CLI, by pinging an IP address from the CE11 device.

Due to how the **ping** command works on NetSim devices, the result is a success.

```
admin@ncs> switch cli
admin@ncs# devices device CE11 live-status exec ping 10.10.0.1
result
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.10.0.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/20/21 ms
dev#
admin@ncs#
```


Netsim devices simulate the **ping** command with a script that runs behind the scenes and mocks the output of the command. In this way, every **ping** command execution is successful. However, the *Cisco IOS XR* NetSim device does not support the **ping** command execution. For this reason, you will only implement self-test action for the *Cisco IOS device*. You can, on the other hand, execute the **ping** command on a normal Cisco IOS XR device.

Step 10

Exit the NSO CLI.

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

Step 11

Open the YANG service model with a text editor.

```
student@student-vm:~/nso300$ cd packages/static-route/
student@student-vm:~/nso300/packages/static-route$ vi src/yang/static-
route.yang
```

The model should be as shown when you open it for the first time.

```
module static-route {

  namespace "http://example.com/static-route";
  prefix static-route;

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

  augment /ncs:services {
    list static-route {
      description "Static route service.";

      key device;

      uses ncs:service-data;
      ncs:servicepoint static-route-servicepoint;

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

      list route {
        description "Device routes.";

        key ip-prefix;

        leaf ip-prefix {
          tailf:info "IP Prefix";
          type inet:ip-prefix;
          mandatory true;
        }

        leaf next-hop {
          tailf:info "Next Hop";
          type inet:ipv4-address;
          mandatory true;
        }
      }
    }
  }
}
```

Step 12

Define an **action** container that contains a **self-test** action. Place the container as a

child of the **route** element, since the action must exist for each route in the service.

```
module static-route {

  namespace "http://example.com/static-route";
  prefix static-route;

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

  augment /ncs:services {
    list static-route {
      description "Static route service.";

      key device;

      uses ncs:service-data;
      ncs:servicepoint static-route-servicepoint;

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

      list route {
        description "Device routes.";

        key ip-prefix;

        leaf ip-prefix {
          tailf:info "IP Prefix";
          type inet:ip-prefix;
          mandatory true;
        }

        leaf next-hop {
          tailf:info "Next Hop";
          type inet:ipv4-address;
          mandatory true;
        }

        container action {
          tailf:action self-test {
            tailf:actionpoint self-test;
            tailf:info "Ping next-hop address.";
          }
        }
      }
    }
```

```
    }
  }
```

Step 13

Add an output element to the action. This element can be used to notify the NSO administrator about the action results and other customizable information. In this case, return the success of the action as a custom output message.

```
module static-route {

  namespace "http://example.com/static-route";
  prefix static-route;

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

  augment /ncs:services {
    list static-route {
      description "Static route service.";

      key device;

      uses ncs:service-data;
      ncs:servicepoint static-route-servicepoint;

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

      list route {
        description "Device routes.";

        key ip-prefix;

        leaf ip-prefix {
          tailf:info "IP Prefix";
          type inet:ip-prefix;
          mandatory true;
        }

        leaf next-hop {
          tailf:info "Next Hop";
          type inet:ipv4-address;
          mandatory true;
        }
```

```
        container action {
          tailf:action self-test {
            tailf:actionpoint self-test;
            tailf:info "Ping next-hop address.";

            output {
              leaf output {
                type string;
              }
            }
          }
        }
      }
    }
  }
}
```


An action can also have some input parameters. However, in this example, all the required action input is read from the static route service.

Step 14

Save the file and exit the file editor.

Step 15

Open the **static-route.py** file, located in the **python/static_route** folder. You now need to implement the self-test action with some Python code.

```
student@student-vm:~/nso300/packages/static-route$ vi python/
static_route/static_route.py
```

Step 16

Import the Action module from the **ncs.dp** library.

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

class ServiceCallbacks(Service):

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

        for route in service.route:

            # Convert CIDR to network address and netmask
            net = ipaddress.ip_network(route.ip_prefix)
```

```
            network_address = str(net.network_address)
            netmask = str(net.netmask)

            tvars = ncs.template.Variables()
            tvars.add('DEVICE', service.device)
            tvars.add('NETWORK-ADDRESS', network_address)
            tvars.add('NETMASK', netmask)
            tvars.add('NEXT-HOP', route.next_hop)
            tvars.add('IP-PREFIX', route.ip_prefix)

            template = ncs.template.Template(service)
            template.apply('static-route-template', tvars)


# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class StaticRoute(ncs.application.Application):
    def setup(self):
        self.log.info('StaticRoute RUNNING')
        self.register_service('static-route-servicepoint',
ServiceCallbacks)

    def teardown(self):
        self.log.info('StaticRoute FINISHED')
```

Step 17

Create a **SelfTest** class and register it to the *self*-test action point.

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

class ServiceCallbacks(Service):

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

        for route in service.route:

            # Convert CIDR to network address and netmask
            net = ipaddress.ip_network(route.ip_prefix)
            network_address = str(net.network_address)
            netmask = str(net.netmask)

            tvars = ncs.template.Variables()
            tvars.add('DEVICE', service.device)
            tvars.add('NETWORK-ADDRESS', network_address)
            tvars.add('NETMASK', netmask)
            tvars.add('NEXT-HOP', route.next_hop)
            tvars.add('IP-PREFIX', route.ip_prefix)
```

```
            template = ncs.template.Template(service)
            template.apply('static-route-template', tvars)

class SelfTest(Action):

    @Action.action
    def cb_action(self, uinfo, name, kp, action_input, action_output,
trans):
        self.log.info(f"Action called: {name}")


# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class StaticRoute(ncs.application.Application):
    def setup(self):
        self.log.info('StaticRoute RUNNING')
        self.register_service('static-route-servicepoint',
ServiceCallbacks)
        self.register_action('self-test', SelfTest)

    def teardown(self):
        self.log.info('StaticRoute FINISHED')
```

Step 18

Use the **ncs.maapi.single_read_trans()** method to open a read session toward ncs. Save the **service** and **route** nodes to variables.

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

class ServiceCallbacks(Service):

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

        for route in service.route:

            # Convert CIDR to network address and netmask
            net = ipaddress.ip_network(route.ip_prefix)
            network_address = str(net.network_address)
            netmask = str(net.netmask)

            tvars = ncs.template.Variables()
            tvars.add('DEVICE', service.device)
            tvars.add('NETWORK-ADDRESS', network_address)
            tvars.add('NETMASK', netmask)
            tvars.add('NEXT-HOP', route.next_hop)
            tvars.add('IP-PREFIX', route.ip_prefix)

            template = ncs.template.Template(service)
```

```
            template.apply('static-route-template', tvars)

class SelfTest(Action):

    @Action.action
    def cb_action(self, uinfo, name, kp, action_input, action_output,
trans):
        self.log.info(f"Action called: {name}")

        with ncs.maapi.single_read_trans('admin', 'python',
db=ncs.OPERATIONAL) as t:
            root = ncs.maagic.get_root(t)
            route = ncs.maagic.get_node(t, kp)._parent
            service = route._parent._parent


# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class StaticRoute(ncs.application.Application):
    def setup(self):
        self.log.info('StaticRoute RUNNING')
        self.register_service('static-route-servicepoint',
ServiceCallbacks)
        self.register_action('self-test', SelfTest)

    def teardown(self):
        self.log.info('StaticRoute FINISHED')
```

Step 19

Execute the **ping** command on the service device, parse, save, and return the output and the command success.

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

class ServiceCallbacks(Service):

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

        for route in service.route:

            # Convert CIDR to network address and netmask
            net = ipaddress.ip_network(route.ip_prefix)
            network_address = str(net.network_address)
            netmask = str(net.netmask)

            tvars = ncs.template.Variables()
            tvars.add('DEVICE', service.device)
            tvars.add('NETWORK-ADDRESS', network_address)
```

```
            tvars.add('NETMASK', netmask)
            tvars.add('NEXT-HOP', route.next_hop)
            tvars.add('IP-PREFIX', route.ip_prefix)

            template = ncs.template.Template(service)
            template.apply('static-route-template', tvars)

class SelfTest(Action):

    @Action.action
    def cb_action(self, uinfo, name, kp, action_input, action_output,
trans):
        self.log.info(f"Action called: {name}")

        with ncs.maapi.single_read_trans('admin', 'python',
db=ncs.OPERATIONAL) as t:
            root = ncs.maagic.get_root(t)
            route = ncs.maagic.get_node(t, kp)._parent
            service = route._parent._parent

            ping =
root.devices.device[service.device].live_status.ios_stats__exec.ping
            ping_input = ping.get_input()
            ping_input.args = [route.next_hop]
            output = ping(ping_input)
            success = '!!!!!!' in output.result
            if success:
                action_output.output = f"Next hop for route
{route.ip_prefix} reachable."
            else:
                action_output.output = f"Next hop for route
{route.ip_prefix} NOT reachable."

# ------------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ------------------------------------------------
class StaticRoute(ncs.application.Application):
    def setup(self):
        self.log.info('StaticRoute RUNNING')
        self.register_service('static-route-servicepoint',
ServiceCallbacks)
        self.register_action('self-test', SelfTest)

    def teardown(self):
        self.log.info('StaticRoute FINISHED')
```

Step 20

Save the file and exit the file editor.

Step 21

Return to the **~/nso300** directory and reload the packages by using the **make testenv-build** command.

```
student@student-vm:~/nso300/packages/static-route$ cd ../..
```

```
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=${NSO} -e PACKAGE_RELOAD= -e
SKIP_LINT= -e PKG_FILE=nso300.gitlab.local/nso300/package:5.3.2-student
nso300.gitlab.local/cisco-nso-dev:5.3.2 /src/nid/testenv-build; \
done
-- Rebuilding for NSO: testenv-nso300-5.3.2-student-nso
(package-meta-data.xml|\.cli1|\.yang1)
make: Entering directory '/var/opt/ncs/packages/static-route/src'
mkdir -p java/src//
/opt/ncs/ncs-5.3.2/bin/ncsc  `ls static-route-ann.yang  > /dev/null
2>&1 && echo "-a static-route-ann.yang"` \
              -c -o ../load-dir/static-route.fxs yang/static-route.yang
make: Leaving directory '/var/opt/ncs/packages/static-route/src'
make: Entering directory '/src/packages/static-route'
if [ ! -f build-meta-data.xml ]; then \
        export PKG_NAME=$(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/static-route/build-meta-data.xml; fi
make: Leaving directory '/src/packages/static-route'
-- 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 static-route
    result true
}
student@student-vm:~/nso300$
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully designed and implemented a self-test action.

Task 2: Add and Test a Static Route

In this task, you will add a static route to a device and ping the next-hop address using the service action you created.

Activity

Complete these steps:

Step 1

Enter the NSO CLI inside NSO System container using the **make testenv-cli** command.

```
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 e7035af9b837
admin@ncs>
```

Step 2

Switch the CLI mode and enter config mode.

```
admin@ncs> switch cli
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)#
```

Step 3

Add a static route to the PE11 device, commit the transaction, and exit the configuration mode.

```
admin@ncs(config)# services static-route PE11
admin@ncs(config-static-route-PE11)# route 10.100.0.0/16 next-hop
10.0.0.1
admin@ncs(config-route-10.100.0.0/16)# top
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# exit
```

Step 4

Call the self-test action for the new route.

```
admin@ncs# services static-route PE11 route 10.100.0.0/16 action self-
test
admin@ncs#
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully executed the service action.