Implement Stacked Services

Introduction

In this activity, you will use an advanced service design approach called service stacking. You will design and implement a high-level DMZ service that includes the router and firewall services.

After completing this activity, you will be able to:

- Design and implement a service that uses service stacking.

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*
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.

Formatting	Description and Examples
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).

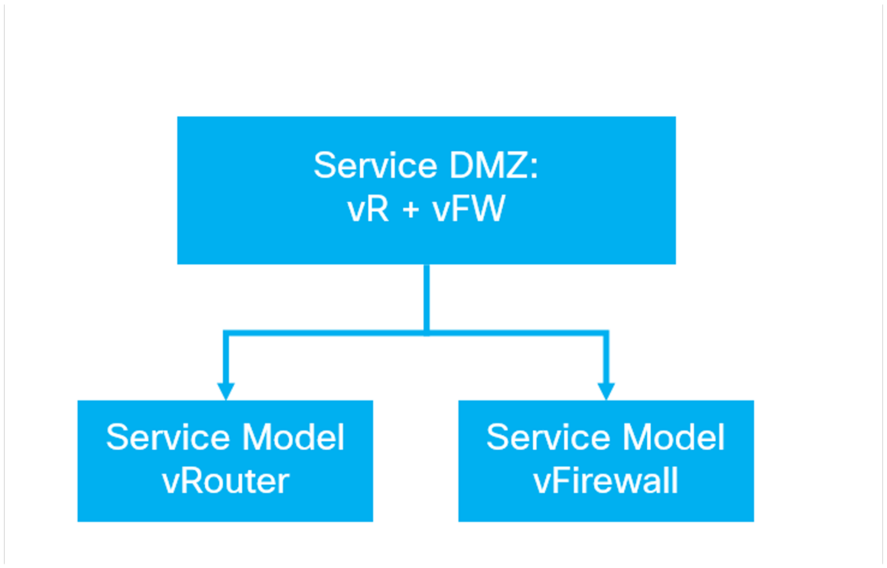
Makefile commands for Docker environment:

Command	Comment
make build	Builds the main NSO Docker image.
make testenv-start	Starts the NSO Docker environment.
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.

Command	Comment
make dev-shell	Enters the Linux shell of the NSO Docker development container.

Visual Aids

The figure below provides a visual aid for this activity.

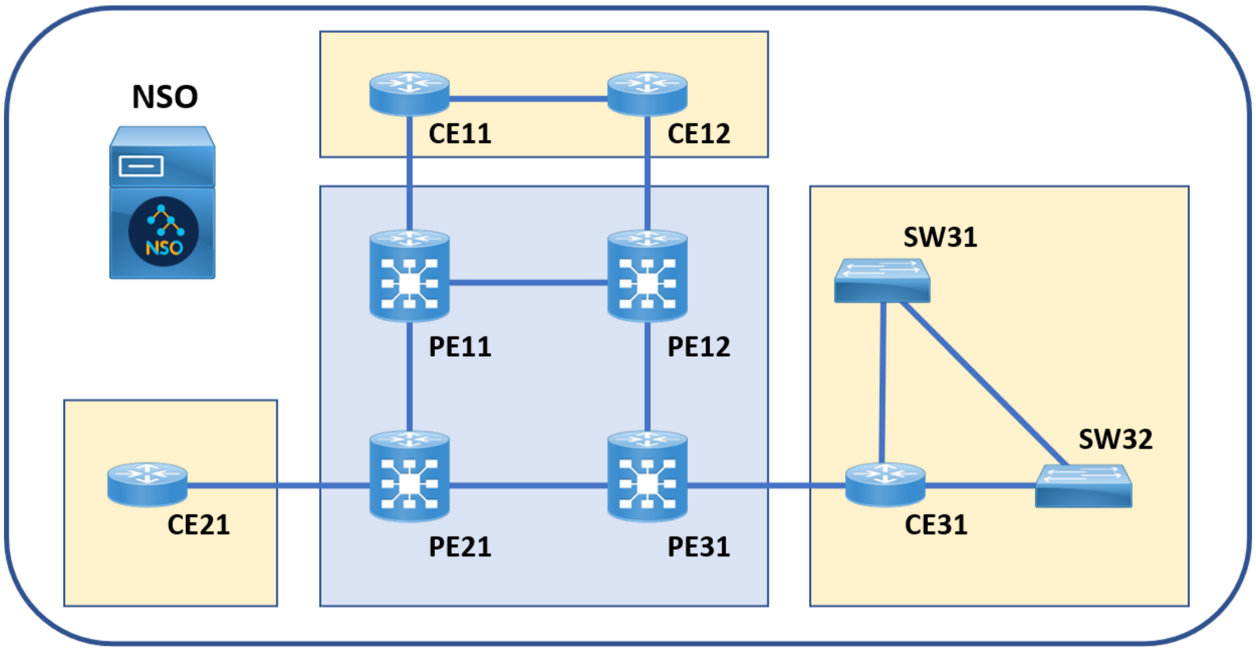


Lab Topology Information

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 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.

- 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: Create a Router Service

In this task, you will create a router service. A basic service bundle consists of a virtual router and virtual firewall service instance. A virtual router is used to provide client connectivity between the customer and internet networks. A virtual firewall is used to block or allow certain types of traffic between the two networks.

Before you can create a bundle service, you need to implement the virtual router service. When it has been created and tested, you can create a high-level service called DMZ, which will instantiate both the firewall and router services.

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 using 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

Enter the development NSO Docker container shell with the command **make dev-shell** 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@4b68acc5198d:/# cd src/packages/
root@4b68acc5198d:/src/packages#
```

Step 5

Use the **ncs-make-package** tool to create a new service skeleton called **router**. Use the **template-based** service skeleton type.

```
root@4b68acc5198d:/src/packages# ncs-make-package --service-skeleton
template router
```

Step 6

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

```
root@4b68acc5198d:/src/packages# chown -Rv 1000:1000 router
changed ownership of 'router/test/internal/Makefile' from root:root to
1000:1000
changed ownership of 'router/test/internal/lux/basic/Makefile' from
root:root to 1000:1000
changed ownership of 'router/test/internal/lux/basic/run.lux' from
root:root to 1000:1000
changed ownership of 'router/test/internal/lux/basic' from root:root to
1000:1000
changed ownership of 'router/test/internal/lux/Makefile' from root:root
to 1000:1000
changed ownership of 'router/test/internal/lux' from root:root to
1000:1000
changed ownership of 'router/test/internal' from root:root to 1000:1000
changed ownership of 'router/test/Makefile' from root:root to 1000:1000
changed ownership of 'router/test' from root:root to 1000:1000
changed ownership of 'router/templates/router-template.xml' from
root:root to 1000:1000
changed ownership of 'router/templates' from root:root to 1000:1000
changed ownership of 'router/src/Makefile' from root:root to 1000:1000
changed ownership of 'router/src/yang/router.yang' from root:root to
1000:1000
changed ownership of 'router/src/yang' from root:root to 1000:1000
changed ownership of 'router/src' from root:root to 1000:1000
changed ownership of 'router/package-meta-data.xml' from root:root to
```

```
1000:1000
changed ownership of 'router' from root:root to 1000:1000
root@4b68acc5198d:/src/packages# logout
student@student-vm:~/nso300$
```

Step 7

Edit the **router.yang** file, to create the service model. Create a grouping *router*, containing a list of *routes*. Each route will use leaves *network, mask,* and *gateway* for storing the route information. Groupings are used to define a reusable block of nodes, which may be used locally in the module or submodule, and by other modules that import from it. They are instantiated with the *uses* statement.

This is how the YANG model should initially appear:

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

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

  grouping router {
    list routes {
      key "network mask";
      min-elements 1;

      leaf network {
        type inet:ipv4-address;
      }
      leaf mask {
        type inet:ipv4-address;
      }
      leaf gateway {
        mandatory true;
        type inet:ipv4-address;
      }
    }
  }

  augment /ncs:services {

    list router {
      description "This is a router service";

      key name;

      uses ncs:service-data;
      ncs:servicepoint "router";

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

      uses router;
    }
  }
}
```

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

Step 8

Save the file when finished.

Step 9

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

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

Step 10

Compile the package. Use the **make testenv-build** command, which recompiles and reloads/redeploys the packages, used by the Docker NSO containers. Make sure that no errors are present.

```
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=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

<... output omitted ...>

-- 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 router
    result true
}
student@student-vm:~/nso300$
```

Step 11

Connect to the NSO Docker 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 19c16f78fbfd
admin@ncs>
```

Step 12

Switch the CLI type.

```
admin@ncs> switch cli
admin@ncs#
```

Step 13

Use the Cisco NSO CLI and the test device CE31 to obtain the configuration in XML format.

```
admin@ncs# config
```

```
Entering configuration mode terminal
admin@ncs(config)# devices device CE31 config
admin@ncs(config-config)# ip route 192.168.1.0 255.255.255.0 10.0.0.1
admin@ncs(config-config)# commit dry-run outformat xml
result-xml {
    local-node {
        data <devices xmlns="http://tail-f.com/ns/ncs">
               <device>
                 <name>CE31</name>
                 <config>
                   <ip xmlns="urn:ios">
                     <route>
                       <ip-route-forwarding-list>
                         <prefix>192.168.1.0</prefix>
                         <mask>255.255.255.0</mask>
                         <forwarding-address>10.0.0.1</forwarding-
address>
                       </ip-route-forwarding-list>
                     </route>
                   </ip>
                 </config>
               </device>
             </devices>
    }
}
admin@ncs(config-config)# abort
admin@ncs# exit
```

Step 14

Create the **router-template.xml** XML mapping template for the static route configuration.

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

Use the XML configuration to build the template file and parametrize it. Replace the placeholder values with XPath queries that will use the service instance input data:

- 192.168.1.0: {network}
- 255.255.255.0: {mask}
- 10.0.0.1: {gateway}

Step 15

In addition, add a **foreach** tag to the **ip-route-forwarding-list** element to repeat the configuration for every entry in the routes list.

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="router">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{/device}</name>
      <config>
        <ip xmlns="urn:ios">
          <route>
            <ip-route-forwarding-list foreach = "{routes}">
              <prefix>{network}</prefix>
              <mask>{mask}</mask>
              <forwarding-address>{gateway}</forwarding-address>
            </ip-route-forwarding-list>
          </route>
        </ip>
      </config>
    </device>
  </devices>
</config-template>
```

Step 16

Save the file when finished.

Step 17

Compile the package.

```
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

<... output omitted ...>

-- Redeploying package router for NSO testenv-nso300-5.3.2-student-nso
result true
student@student-vm:~/nso300$
```

Step 18

Create a test instance of a *router* service.

```
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 f6a87450ba68
admin@ncs> switch cli
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services router first-route device CE11 routes
10.0.10.0 255.255.255.128 gateway 10.0.20.1
```

```
admin@ncs(config-routes-10.255.255.0/255.255.255.128)# commit dry-run
cli {
    local-node {
        data  devices {
                  device CE11 {
                      config {
                          ip {
                              route {
             +                    ip-route-forwarding-list 10.0.10.0
255.255.255.128 10.0.20.1 {
             +                    }
                              }
                          }
                      }
                  }
              }
              services {
             +    router first-route {
             +        device CE11;
             +        routes 10.0.10.0 255.255.255.128 {
             +            gateway 10.0.20.1;
             +        }
             +    }
              }
    }
}
admin@ncs(config-routes-10.0.10.0/255.255.255.128)# abort
admin@ncs# exit
student@student-vm:~/nso300$
```

Activity Verification

You have completed this task when you attain the following result:

- You have finished the router service.
- You have successfully created a test instance of the router service.

Task 2: Create a Firewall Service

In this task, you will create a firewall service. Another second-level service is a virtual firewall service. This service package will include some Python code because the mapping logic requires a transformation of the input parameters before they can be used in the device configuration.

In the example, you will work with Cisco ASA, which has a specific way of configuring an ACL entry. The network masks are bitwise inverted—wildcard masks. Specifying ports is only valid for a TCP or UDP protocol filter.

Activity

Complete these steps:

Step 1

Enter the development NSO Docker container shell with the command **make dev-**

shell 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@ee53066ed2ec:/# cd src/packages
root@ee53066ed2ec:/src/packages#
```

Step 2

Use the **ncs-make-package** tool to create a new service skeleton called **firewall**. Use the **python-and-template** based service skeleton type.

```
root@ee53066ed2ec:/src/packages# ncs-make-package --service-skeleton
python-and-template --component-class firewall.Firewall firewall
```

Step 3

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

```
root@ee53066ed2ec:/src/packages# chown -Rv 1000:1000 firewall
changed ownership of 'firewall/test/internal/Makefile' from root:root
to 1000:1000
changed ownership of 'firewall/test/internal/lux/Makefile' from
root:root to 1000:1000
changed ownership of 'firewall/test/internal/lux/service/Makefile' from
root:root to 1000:1000
changed ownership of 'firewall/test/internal/lux/service/dummy-
service.xml' from root:root to 1000:1000
changed ownership of 'firewall/test/internal/lux/service/run.lux' from
root:root to 1000:1000
changed ownership of 'firewall/test/internal/lux/service/dummy-
device.xml' from root:root to 1000:1000
changed ownership of 'firewall/test/internal/lux/service/pyvm.xml' from
root:root to 1000:1000
changed ownership of 'firewall/test/internal/lux/service' from
root:root to 1000:1000
changed ownership of 'firewall/test/internal/lux' from root:root to
1000:1000
changed ownership of 'firewall/test/internal' from root:root to
1000:1000
changed ownership of 'firewall/test/Makefile' from root:root to
1000:1000
changed ownership of 'firewall/test' from root:root to 1000:1000
changed ownership of 'firewall/python/firewall/firewall.py' from
root:root to 1000:1000
changed ownership of 'firewall/python/firewall/__init__.py' from
root:root to 1000:1000
changed ownership of 'firewall/python/firewall' from root:root to
1000:1000
changed ownership of 'firewall/python' from root:root to 1000:1000
changed ownership of 'firewall/README' from root:root to 1000:1000
changed ownership of 'firewall/templates/firewall-template.xml' from
```

```
root:root to 1000:1000
changed ownership of 'firewall/templates' from root:root to 1000:1000
changed ownership of 'firewall/src/Makefile' from root:root to
1000:1000
changed ownership of 'firewall/src/yang/firewall.yang' from root:root
to 1000:1000
changed ownership of 'firewall/src/yang' from root:root to 1000:1000
changed ownership of 'firewall/src' from root:root to 1000:1000
changed ownership of 'firewall/package-meta-data.xml' from root:root to
1000:1000
changed ownership of 'firewall' from root:root to 1000:1000
root@ee53066ed2ec:/src/packages# logout
student@student-vm:~/nso300$
```

Step 4

Edit the **firewall.yang** file and create the service model. Create a grouping **firewall**, containing a list of **access-list-rules**. Each access list rule will use leaves listed in the table below for storing the rule information.

Name	Type	Description
name	string	Name of the ACL rule
action	enum: permit, deny	Action for the ACL rule
protocol	enum: ip, icmp, udp, tcp	Layer 3 protocol for the ACL rule
direction	enum: in, out	Direction on which ACL rule is applied
interface	enum: inside, outside, management	ASA interface for the ACL
src-ip	inet:ipv4-address	Source IP address
src-mask	inet:ipv4-address	Source IP address mask
src-port	union: int16 or any	Source port (if protocol UDP or TCP)
dest-ip	inet:ipv4-address	Destination IP address
dest-mask	inet:ipv4-address	Destination IP address mask
dest-port	union: int16 or any	Destination port (if protocol UDP or TCP)

```
student@student-vm:~/nso300$ vim packages/firewall/src/yang/
firewall.yang
```

Step 5

Make sure that after making the changes to the skeleton model, your *firewall.yang* service model should looks like this.

```
module firewall {

  namespace "http://example.com/firewall";
  prefix firewall;

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

  grouping firewall {
    list access-list-rules {
      ordered-by user;
      key name;

      leaf name {
        type string;
      }
      leaf action {
        mandatory true;
        type enumeration {
          enum permit;
          enum deny;
        }
      }
      leaf protocol {
        default ip;
        type enumeration {
          enum ip;
          enum tcp;
          enum udp;
          enum icmp;
        }
      }
      leaf direction {
        mandatory true;
        type enumeration {
          enum in;
          enum out;
        }
      }
      leaf interface {
        mandatory true;
        type enumeration {
          enum inside;
          enum outside;
          enum management;
        }
      }
      leaf src-ip {
        mandatory true;
        type inet:ipv4-address;
      }
      leaf src-mask {
        mandatory true;
        type inet:ipv4-address;
      }
      leaf src-port {
        when "../protocol = 'tcp' or ../protocol = 'udp'";
        default any;
        type union {
          type uint16;
          type enumeration {
            enum any;
          }
        }
      }
      leaf dest-ip {
        mandatory true;
        type inet:ipv4-address;
      }
      leaf dest-mask {
        mandatory true;
        type inet:ipv4-address;
      }
      leaf dest-port {
        when "../protocol = 'tcp' or ../protocol = 'udp'";
        default any;
        type union {
          type uint16;
          type enumeration {
            enum any;
          }
        }
      }
    }
  }

  augment /ncs:services {

    list firewall {
      description "Firewall service";
      key name;

      leaf name {
        type string;
      }

      uses ncs:service-data;
      ncs:servicepoint firewall-servicepoint;

      leaf device {
        mandatory true;
        type leafref {
          path "/ncs:devices/ncs:device/ncs:name";
        }
      }
      uses firewall;
    }
  }
}
```

At nodes **src-port** and **dest-port** a *when* method is being used to specify a condition. In this case one can choose a port number only when TCP or UDP protocol is being used.

Step 6

Save the file when finished.

Step 7

Connect to the NSO Docker 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 774da66c63d7
admin@ncs>
```

Step 8

Switch the CLI type.

```
admin@ncs> switch cli
admin@ncs#
```

Step 9

Use the Cisco NSO CLI and the test device ASA41 to obtain the configuration in XML format. Assign the ACL on the internet-facing interface (GigabitEthernet 3), direction in.

The Cisco IOS NED used in the lab models ACL rules (list entries) as strings. It is possible to construct a rule that is accepted by Cisco NSO, but will later be rejected by the device, making the service instance configuration fail.

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device ASA41 config
admin@ncs(config-config)# access-list TEST "extended permit tcp
10.0.0.0 255.255.255.0 eq 1337 20.0.0.0 255.255.255.0 eq 80"
```

```
admin@ncs(config-config)# access-group TEST in interface inside
admin@ncs(config-config)# commit dry-run outformat xml
result-xml {
    local-node {
        data <devices xmlns="http://tail-f.com/ns/ncs">
               <device>
                 <name>ASA41</name>
                 <config>
                   <access-list xmlns="http://cisco.com/ned/asa">
                     <access-list-id>
                       <id>TEST</id>
                       <rule>
                         <id>extended permit tc 10.0.0.0 255.255.255.0
eq 1337 20.0.0.0 255.255.255.0 eq 80</id>
                       </rule>
                     </access-list-id>
                   </access-list>
                   <access-group xmlns="http://cisco.com/ned/asa">
                     <interface-list>
                       <direction>in</direction>
                       <interface>inside</interface>
                       <access-list>TEST</access-list>
                     </interface-list>
                   </access-group>
                 </config>
               </device>
             </devices>
    }
}
admin@ncs(config-config)# abort
admin@ncs# exit
student@student-vm:~/nso300$
```

Step 10

Edit a **firewall-template.xml** XML template in the **templates** directory.

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

Step 11

Parametrize the static XML template with XPath variables. You will need the following variables in place of the static values:

Name	Description
$DEVICE	Device Name
$INTERFACE-ID	Access-group Interface ID
$ACCESS-LIST-NAME	Access List Name

Name	Description
$ACCESS-LIST-DIRECTION	Access List Direction on Interface
$ACCESS-LIST-RULE	Access List Rule (as a string)

The final parametrized template should look like the listing below.

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{$DEVICE}</name>
      <config>
        <access-list xmlns="http://cisco.com/ned/asa">
          <access-list-id>
            <id>{$ACCESS-LIST-NAME}</id>
            <rule>
              <id>{$ACCESS-LIST-RULE}</id>
            </rule>
          </access-list-id>
        </access-list>
        <access-group xmlns="http://cisco.com/ned/asa">
          <interface-list>
            <direction>{$ACCESS-LIST-DIRECTION}</direction>
            <interface>{$INTERFACE-ID}</interface>
            <access-list>{$ACCESS-LIST-NAME}</access-list>
          </interface-list>
        </access-group>
      </config>
    </device>
  </devices>
</config-template>
```

Step 12

Save the file when finished.

Step 13

Open the **firewall.py** Python code

Step 14

Remove the comments and dummy code to improve readability.

Step 15

Implement the mapping logic that will build the correct ACL rule from the input parameters and apply the template.

```
student@student-vm:~/nso300$ vim packages/firewall/python/firewall/
firewall.py
```

Step 16

The rules for building an ACL rule state that the TCP or UDP port number must be prepended with the "**eq**" string. Create the helper method -***_stringify_port*** in the **ServiceCallbacks** class.

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


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})')


    def _stringify_port(self, port):
        if isinstance(port, int):
            return f'eq {port}'
        else:
            return ''

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

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

Step 17

The second helper method **_build_acl_rule** will be used to create the ACL rule string from the input parameters.

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

```
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})')

    def _build_acl_rule(self, rule):
        action = rule.action.string
        protocol = rule.protocol
        src_ip = rule.src_ip
        src_mask = rule.src_mask
        src_port = self._stringify_port(rule.src_port)
        dest_ip = rule.dest_ip
        dest_mask = rule.dest_mask
        dest_port = self._stringify_port(rule.dest_port)
        acl_entry = f'extended {action} {protocol} {src_ip} {src_mask}
{src_port} {dest_ip} {dest_mask} {dest_port}'

        self.log.info(f'Generated ACL entry: {acl_entry}')

        return acl_entry.rstrip()

    def _stringify_port(self, port):
        if isinstance(port, int):
            return f'eq {port}'
        else:
            return ''

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

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

In this example implementation, all helper methods are part of the **ServiceCallbacks** class – make sure to use the correct indentation level! You could also implement these methods as functions in the **firewall** module, or in a separate supporting module.

Step 18

Finally, you can implement the **cb_create** method to provide device configuration. In the beginning of the method, set up some global service variables. For this example, you hardcode the interface and ACL direction, but these could also be service input parameters.

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


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})')

        device = service.device

        for rule in service.access_list_rules:
            template = ncs.template.Template(service)
            tvars = ncs.template.Variables()

            tvars.add('DEVICE', device)
            tvars.add('ACCESS-LIST-NAME', rule.name)
            tvars.add('INTERFACE-ID', rule.interface.string)
            tvars.add('ACCESS-LIST-DIRECTION', rule.direction.string)

            tvars.add('ACCESS-LIST-RULE', self._build_acl_rule(rule))
            template.apply('firewall-template', tvars)

    def _build_acl_rule(self, rule):
        action = rule.action.string
        protocol = rule.protocol
        src_ip = rule.src_ip
        src_mask = rule.src_mask
        src_port = self._stringify_port(rule.src_port)
        dest_ip = rule.dest_ip
        dest_mask = rule.dest_mask
        dest_port = self._stringify_port(rule.dest_port)
        acl_entry = f'extended {action} {protocol} {src_ip} {src_mask}
{src_port} {dest_ip} {dest_mask} {dest_port}'

        self.log.info(f'Generated ACL entry: {acl_entry}')

        return acl_entry.rstrip()

    def _stringify_port(self, port):
        if isinstance(port, int):
            return f'eq {port}'
        else:
            return ''

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

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

Step 19

Save the file when finished.

Step 20

Add some additional instruction to the **Makefile** that will allow you to check the syntax and style of your Python code. This is accomplished by running the **pylint** tool on all Python source files. The tool has already been installed in the lab environment.

```
student@student-vm:~/nso300$ vim packages/firewall/src/Makefile
all: fxs pylint
.PHONY: all

# Include standard NCS examples build definitions and rules
include $(NCS_DIR)/src/ncs/build/include.ncs.mk

SRC = $(wildcard yang/*.yang)
DIRS = ../load-dir java/src/$(JDIR)/$(NS)
FXS = $(SRC:yang/%.yang=../load-dir/%.fxs)

## Uncomment and patch the line below if you have a dependency to a NED
## or to other YANG files
# YANGPATH += ../../<ned-name>/src/ncsc-out/modules/yang \
# ../../<pkt-name>/src/yang

NCSCPATH   = $(YANGPATH:%=--yangpath %)
YANGERPATH = $(YANGPATH:%=--path %)
PYLINT = pylint
PYLINTFLAGS = --disable=R,C
PYDIR = ../python/firewall
PYTHONFILES = $(wildcard $(PYDIR)/*.py)

pylint: $(patsubst %.py, %.pylint, $(PYTHONFILES))

%.pylint:
$(PYLINT) $(PYLINTFLAGS) $*.py || (test $$? -ge 4)

fxs: $(DIRS) $(FXS)

$(DIRS):
mkdir -p $@

../load-dir/%.fxs: yang/%.yang
$(NCSC)  `ls $*-ann.yang  > /dev/null 2>&1 && echo "-a $*-ann.yang"` \
             $(NCSCPATH) -c -o $@ $<

clean:
rm -rf $(DIRS)
.PHONY: clean
```

Step 21

Save the file when finished.

Step 22

Compile the package.

```
student@student-vm:~/nso300$ make testenv-build
<... output omitted ...>
-- Rebuilding for NSO: testenv-nso300-5.3.2-student-nso
(package-meta-data.xml|\.cli$|\.yang$)
make: Entering directory '/var/opt/ncs/packages/router/src'
make: Nothing to be done for 'all'.
```

```
make: Leaving directory '/var/opt/ncs/packages/router/src'
make: Entering directory '/src/packages/router'
<... output omitted ...>
pylint --disable=R,C ../python/firewall/firewall.py || (test $? -ge 4)
<... output omitted ...>
-- 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 firewall
    result true
}
reload-result {
    package router
    result true
}
```

Step 23

Create a test instance of the **firewall** service with the parameters shown below. You do not necessarily have to commit the configuration at the end.

```
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 0a18b5d7699b
admin@ncs> switch cli
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services firewall fw-first-test device ASA41
admin@ncs(config-firewall-fw-first-test)# access-list-rules test
Value for 'action' [deny,permit]: permit
Value for 'direction' [in,out]: in
Value for 'interface' [inside,management,outside]: inside
Value for 'src-ip' (<IPv4 address>): 10.0.10.0
Value for 'src-mask' (<IPv4 address>): 255.255.255.0
Value for 'dest-ip' (<IPv4 address>): 10.0.20.1
Value for 'dest-mask' (<IPv4 address>): 255.255.255.255
admin@ncs(config-access-list-rules-test)# protocol tcp
admin@ncs(config-access-list-rules-test)# dest-port 80
admin@ncs(config-access-list-rules-test)# commit dry-run outformat
native
native {
    device {
        name ASA41
        data access-list test "extended permit tcp 10.0.10.0
255.255.255.0  10.0.20.1 255.255.255.255 eq 80"
             access-group test in interface inside
    }
}
admin@ncs(config-access-list-rules-test)# abort
admin@ncs# logout
student@student-vm:~/nso300$
```

Activity Verification

You have completed this task when you attain the following result:

- You have implemented the firewall service.
- You have created a test instance of the firewall service.

Task 3: Create a DMZ Service

In this task, you will create a DMZ service.

The top-level service DMZ will create instances of the firewall and router services. The service model will contain all necessary parameters to create a router and apply the firewall rules.

The DMZ service will depend on several different packages:

- **Firewall:** The firewall service, implementing the virtual firewall functionality.
- **Router:** The router service, implementing the virtual router functionality.

Activity

Step 1

Enter the development NSO Docker container shell with the command **make dev-shell** 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@6b89acc0b592:/# cd src/packages
root@6b89acc0b592:/src/packages#
```

Step 2

Use the **ncs-make-package** tool to create a new service skeleton called *router.* Use the *template-based* service skeleton type.

```
root@6b89acc0b592:/src/packages# ncs-make-package --service-skeleton
python-and-template --component-class dmz.DMZ dmz
```

Step 3

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

```
root@6b89acc0b592:/src/packages# chown -Rv 1000:1000 dmz
changed ownership of 'dmz/test/internal/Makefile' from root:root to
1000:1000
changed ownership of 'dmz/test/internal/lux/Makefile' from root:root to
1000:1000
changed ownership of 'dmz/test/internal/lux/service/Makefile' from
root:root to 1000:1000
changed ownership of 'dmz/test/internal/lux/service/dummy-service.xml'
from root:root to 1000:1000
changed ownership of 'dmz/test/internal/lux/service/run.lux' from
root:root to 1000:1000
changed ownership of 'dmz/test/internal/lux/service/dummy-device.xml'
from root:root to 1000:1000
changed ownership of 'dmz/test/internal/lux/service/pyvm.xml' from
root:root to 1000:1000
changed ownership of 'dmz/test/internal/lux/service' from root:root to
1000:1000
changed ownership of 'dmz/test/internal/lux' from root:root to
1000:1000
changed ownership of 'dmz/test/internal' from root:root to 1000:1000
changed ownership of 'dmz/test/Makefile' from root:root to 1000:1000
changed ownership of 'dmz/test' from root:root to 1000:1000
changed ownership of 'dmz/python/dmz/dmz.py' from root:root to
1000:1000
changed ownership of 'dmz/python/dmz/__init__.py' from root:root to
1000:1000
changed ownership of 'dmz/python/dmz' from root:root to 1000:1000
changed ownership of 'dmz/python' from root:root to 1000:1000
changed ownership of 'dmz/README' from root:root to 1000:1000
changed ownership of 'dmz/templates/dmz-template.xml' from root:root to
1000:1000
changed ownership of 'dmz/templates' from root:root to 1000:1000
changed ownership of 'dmz/src/Makefile' from root:root to 1000:1000
changed ownership of 'dmz/src/yang/dmz.yang' from root:root to
1000:1000
changed ownership of 'dmz/src/yang' from root:root to 1000:1000
changed ownership of 'dmz/src' from root:root to 1000:1000
changed ownership of 'dmz/package-meta-data.xml' from root:root to
1000:1000
changed ownership of 'dmz' from root:root to 1000:1000
root@6b89acc0b592:/src/packages# logout
student@student-vm:~/nso300$
```

Step 4

Edit the *dmz.yang* file, containing the service model. Add a leaf *csr-name* and *asa-name* for specifying VNF devices where this service will be provisioned. Add two containers for the *router* and *firewall* configuration. To avoid repetition when modeling the router and firewall parameters, use a *uses* statement, to reference the *router* and *firewall* service model.

```
student@student-vm:~/nso300$ vim packages/dmz/src/yang/dmz.yang
module dmz {

  namespace "http://example.com/dmz";
  prefix dmz;

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

  augment /ncs:services {

    list dmz {
      description "This is a DMZ service";

      key name;
      leaf name {
        type string;
      }

      uses ncs:service-data;
      ncs:servicepoint dmz-servicepoint;

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

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

      container router {
        presence "router";
        uses router:router;
      }

      container firewall {
        presence "firewall";
        uses firewall:firewall;
      }
    }
  }
}
```

Note the use of the *presence* statements in the containers. This is important, as you are going to check for the existence of the container in the mapping logic to determine if the Router or Firewall functionality is required in the DMZ service. The parameter for the *presence* statement is a string describing its purpose.

Step 5

Save the file when finished.

Step 6

Because you referenced statements defined in another module (the *import router* and *firewall* statements), you also need to inform the compiler about where to find the *router* and *firewall* module. Add the YANGPATH to the service *Makefile*.

```
student@student-vm:~/nso300$ vim packages/dmz/src/Makefile
all: fxs
.PHONY: all

# Include standard NCS examples build definitions and rules
include $(NCS_DIR)/src/ncs/build/include.ncs.mk

SRC = $(wildcard yang/*.yang)
DIRS = ../load-dir java/src/$(JDIR)/$(NS)
FXS = $(SRC:yang/%.yang=../load-dir/%.fxs)

NCSCPATH   = $(YANGPATH:%=--yangpath %)
YANGERPATH = $(YANGPATH:%=--path %)
YANGPATH += ../../router/src/yang
YANGPATH += ../../firewall/src/yang

fxs: $(DIRS) $(FXS)

$(DIRS):
mkdir -p $@

../load-dir/%.fxs: yang/%.yang
$(NCSC)  `ls $*-ann.yang  > /dev/null 2>&1 && echo "-a $*-ann.yang"` \
             $(NCSCPATH) -c -o $@ $<

clean:
rm -rf $(DIRS)
.PHONY: clean
```

Step 7

Add some additional instruction to the *Makefile* that will allow you to check the syntax and style of your Python code. This is accomplished by running the *pylint* tool on all Python source files. The tool has already been installed in the lab environment.

```
all: fxs pylint
.PHONY: all

# Include standard NCS examples build definitions and rules
include $(NCS_DIR)/src/ncs/build/include.ncs.mk

SRC = $(wildcard yang/*.yang)
DIRS = ../load-dir java/src/$(JDIR)/$(NS)
FXS = $(SRC:yang/%.yang=../load-dir/%.fxs)

NCSCPATH   = $(YANGPATH:%=--yangpath %)
YANGERPATH = $(YANGPATH:%=--path %)
YANGPATH += ../../router/src/yang
YANGPATH += ../../firewall/src/yang

PYLINT = pylint
PYLINTFLAGS = --disable=R,C --reports=n
PYDIR = ../python/dmz
PYTHONFILES = $(wildcard $(PYDIR)/*.py)
```

```
pylint: $(patsubst %.py, %.pylint, $(PYTHONFILES))

%.pylint:
$(PYLINT) $(PYLINTFLAGS) $*.py  || (test $$? -ge 4)

fxs: $(DIRS) $(FXS)

$(DIRS):
mkdir -p $@

../load-dir/%.fxs: yang/%.yang
$(NCSC)  `ls $*-ann.yang  > /dev/null 2>&1 && echo "-a $*-ann.yang"` \
             $(NCSCPATH) -c -o $@ $<

clean:
rm -rf $(DIRS)
.PHONY: clean
```

Step 8

Save the file when finished.

Step 9

Recall the input parameters for the *router* service. You will need to construct an XML template to create a service instance.

Use the show running-configuration services router <instance> | display xml command to get the service instance configuration in XML.

```
student@student-vm:~/nso300$ vim packages/dmz/templates/router-service-template.xml
```

Parametrize the template. Use *$NAME* and *$DEVICE* variables for the *name* and *device* input parameters for the *router* service. Use the *foreach* tag to copy the input parameters containing the list of static routes directly from the *dmz* service instance data. Save the XML template in *router-service-template.xml* template in the *dmz* service package under templates. The final solution is shown below:

```
<config xmlns="http://tail-f.com/ns/config/1.0">
  <services xmlns="http://tail-f.com/ns/ncs">
    <router xmlns="http://com/example/router">
      <name>{$NAME}</name>
      <device>{$DEVICE}</device>
      <?foreach {/routes}?>
      <routes>
        <network>{network}</network>
        <mask>{mask}</mask>
        <gateway>{gateway}</gateway>
      </routes>
      <?end?>
    </router>
  </services>
</config>
```

Step 10

Save the file when finished.

Step 11

The skeleton *dmz.py* class already contains the *cb_create()* method with some sample code. Remove the commented-out code, and only keep the method declaration.

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

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})')

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class DMZ(ncs.application.Application):

    def setup(self):
        self.log.info('DMZ RUNNING')
        self.register_service('dmz-servicepoint', ServiceCallbacks)

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

Step 12

Prepare the variables for storing the "global service input parameters". Store the value of the *csr-name* and *asa-name* leaf in a corresponding variable.

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

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})')

        csr_name = service.csr_name
        asa_name = service.asa_name


# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class DMZ(ncs.application.Application):

    def setup(self):
        self.log.info('DMZ RUNNING')
        self.register_service('dmz-servicepoint', ServiceCallbacks)

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

Step 13

Apply the *router-service-template.xml* service template in the *cb_create()* method. In the beginning, check for the existence of the *router* container in the *dmz* service input parameters with the MAAGIC *exists()* method. If the container exists, you need to apply the template, otherwise ignore it.

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

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})')

        csr_name = service.csr_name
        asa_name = service.asa_name

        if service.router.exists():
            self.log.info('Router config exists, will create Router
instance')
            template = ncs.template.Template(service.router)
            tvars = ncs.template.Variables()
            tvars.add('NAME', service.name)
            tvars.add('DEVICE', csr_name)
            template.apply('router-service-template', tvars)


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

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

Step 14

Save the file when finished.

Step 15

Perform the same steps for the *firewall* service. Create an XML template for the service input parameters. The final template is shown below:

```
student@student-vm:~/nso300$ vim packages/dmz/templates/firewall-
service-template.xml
<config xmlns="http://tail-f.com/ns/config/1.0">
  <services xmlns="http://tail-f.com/ns/ncs">
    <firewall xmlns="http://example.com/firewall">
      <name>{$NAME}</name>
      <device>{$DEVICE}</device>
      <?foreach {/access-list-rules}?>
      <access-list-rules>
        <name>{name}</name>
        <direction>{direction}</direction>
        <interface>{interface}</interface>
        <action>{action}</action>
        <protocol>{protocol}</protocol>
        <src-ip>{src-ip}</src-ip>
        <src-mask>{src-mask}</src-mask>
        <src-port>{src-port}</src-port>
        <dest-ip>{dest-ip}</dest-ip>
        <dest-mask>{dest-mask}</dest-mask>
        <dest-port>{dest-port}</dest-port>
      </access-list-rules>
      <?end?>
    </firewall>
  </services>
</config>
```

Step 16

Save the file when finished.

Step 17

Conditionally apply the template from the *dmz* service, based on the existence of the *firewall* presence container.

```
student@student-vm:~/nso300$ vim packages/dmz/python/dmz/dmz.py
# -*- mode: python; python-indent: 4 -*-
```

```
import ncs
from ncs.application import Service

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})')

        csr_name = service.csr_name
        asa_name = service.asa_name

        if service.router.exists():
            self.log.info('Router config exists, will create Router
instance')
            template = ncs.template.Template(service.router)
            tvars = ncs.template.Variables()
            tvars.add('NAME', service.name)
            tvars.add('DEVICE', csr_name)
            template.apply('router-service-template', tvars)

        if service.firewall.exists():
            self.log.info('Firewall service exists, will create
Firewall instance')
            template = ncs.template.Template(service.firewall)
            tvars = ncs.template.Variables()
            tvars.add('NAME', service.name)
            tvars.add('DEVICE', asa_name)
            template.apply('firewall-service-template', tvars)

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class DMZ(ncs.application.Application):

    def setup(self):
        self.log.info('DMZ RUNNING')
        self.register_service('dmz-servicepoint', ServiceCallbacks)

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

Step 18

Save the file when finished.

Step 19

Compile the package.

```
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)

<... output omitted ...>
```

Step 20

Create a test instance of the *dmz* service.

```
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 0a18b5d7699b
admin@ncs> switch cli
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services dmz webservers csr-name CE31 asa-name ASA41
admin@ncs(config-dmz-webservers)# commit
Commit complete.
admin@ncs(config-dmz-webservers)# top
admin@ncs(config)#
```

Step 21

Configure a static route on the web servers dmz service instance, to test the router functionality.

```
admin@ncs(config)# services dmz webservers
admin@ncs(config-dmz-webservers)# router routes 10.100.100.128
255.255.255.128 gateway 10.30.50.1
admin@ncs(config-routes-10.100.100.128/255.255.255.128)# commit dry-run
outformat native
native {
    device {
        name CE31
        data ip route 10.100.100.128 255.255.255.128 10.30.50.1
    }
}
admin@ncs(config-routes-10.100.100.128/255.255.255.128)# commit
Commit complete.
admin@ncs(config-routes-10.100.100.128/255.255.255.128)# top
admin@ncs(config)#admin@ncs(config)#
```

Step 22

Configure a firewall rule todatabase on the webservers dmz service instance, to test the Firewall functionality.

```
admin@ncs(config)# services dmz webservers firewall access-list-rules
todatabase
Value for 'action' [deny,permit]: permit
Value for 'direction' [in,out]: in
Value for 'interface' [inside,management,outside]: out
Value for 'src-ip' (<IPv4 address>): 10.100.100.128
Value for 'src-mask' (<IPv4 address>): 255.255.255.128
Value for 'dest-ip' (<IPv4 address>): 10.250.200.128
Value for 'dest-mask' (<IPv4 address>): 255.255.255.128
admin@ncs(config-access-list-rules-todatabase)# protocol tcp
admin@ncs(config-access-list-rules-todatabase)# dest-port 1521
admin@ncs(config-access-list-rules-todatabase)# commit dry-run
outformat native
native {
    device {
        name ASA41
        data access-list todatabase "extended permit tcp 10.100.100.128
255.255.255.128  10.250.200.128 255.255.255.128 eq 1521"
             access-group todatabase in interface outside
    }
}
admin@ncs(config-access-list-rules-todatabase)# commit
Commit complete.
admin@ncs(config-access-list-rules-todatabase)# top
admin@ncs(config)#
```

Step 23

The commit operation must complete. Examine the output of the *services dmz web servers get-modifications* and observe the changed device-modifications, containing the static route and a firewall rule.

```
admin@ncs(config)# services dmz webservers get-modifications
cli {
    local-node {
        data  devices {
                   device ASA41 {
                       config {
                           access-list {
              +                access-list-id todatabase {
              +                    rule "extended permit tcp
10.100.100.128 255.255.255.128  10.250.200.128 255.255.255.128 eq
1521";
              +                }
                           }
                           access-group {
              +                interface-list in outside {
              +                    access-list todatabase;
              +                }
                           }
                       }
                   }
```

```
                    device CE31 {
                        config {
                            ip {
                                route {
              +                     ip-route-forwarding-list
10.100.100.128 255.255.255.128 10.30.50.1 {
              +                     }
                                }
                            }
                        }
                    }
                }
                services {
              +     router webservers {
              +         device CE31;
              +         routes 10.100.100.128 255.255.255.128 {
              +             gateway 10.30.50.1;
              +         }
              +     }
              +     firewall webservers {
              +         device ASA41;
              +         access-list-rules todatabase {
              +             action permit;
              +             protocol tcp;
              +             direction in;
              +             interface outside;
              +             src-ip 10.100.100.128;
              +             src-mask 255.255.255.128;
              +             src-port any;
              +             dest-ip 10.250.200.128;
              +             dest-mask 255.255.255.128;
              +             dest-port 1521;
              +         }
              +     }
                }

    }
}
admin@ncs(config)#
```

Step 24

Perform the suggested steps (services global-settings collect-forward-diff true and services dmz web servers re-deploy) in case you get the following error:

```
admin@ncs(config)# services dmz webservers get-modifications
Error: No forward diff found for this service. Either /services/global-
settings/collect-forward-diff is false, or the forward diff has become
invalid. A re-deploy of the service will correct the latter.
admin@ncs(config)# services global-settings collect-forward-diff true
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# services dmz webservers re-deploy
```

Activity Verification

You have completed this task when you attain the following result:

- You have finished the dmz service.
- You are able to create a test instance of the DMZ service and configure the router and firewall functionality on the CE31 and ASA41 device.