Deploy NFV for DMZ Service

Introduction

In this activity, you will learn how to modify an existing NSO service to deploy NFVs on ESC. You will transform the *dmz* service into a nanoservice and deploy an NFV, which consists of a virtual router and a virtual firewall.

After completing this activity, you will be able to:

- Implement NFV with ESC.
- Implement self-test action.

Job Aids

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

- This Lab Guide

The following table contains credentials and addresses that you might need.

Device	Username	Password	Address
Student-VM	student	1234QWer	10.0.0.102
NSO application	admin	admin	10.0.0.102
ESC	admin	admin	10.0.0.104
OpenStack	admin	admin	10.0.0.103
OpenStack SSH	root	1234QWer	10.0.0.103

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.
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:

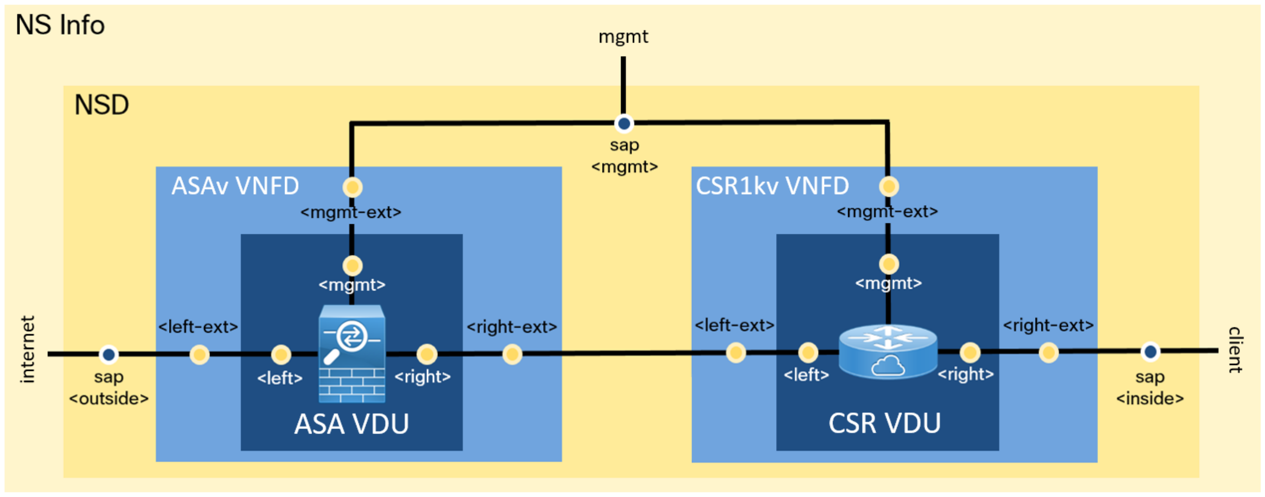
Command	Comment
switch cli	Change CLI style.
show ?	Display all command options for current mode.
configure	Enter configuration mode.
commit	Commit new configuration (configuration mode only command).
show configuration	Display new configuration that has not yet been committed (configuration mode only 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.
make dev-shell	Enters the Linux shell of the NSO Docker development container.

Visual Objective

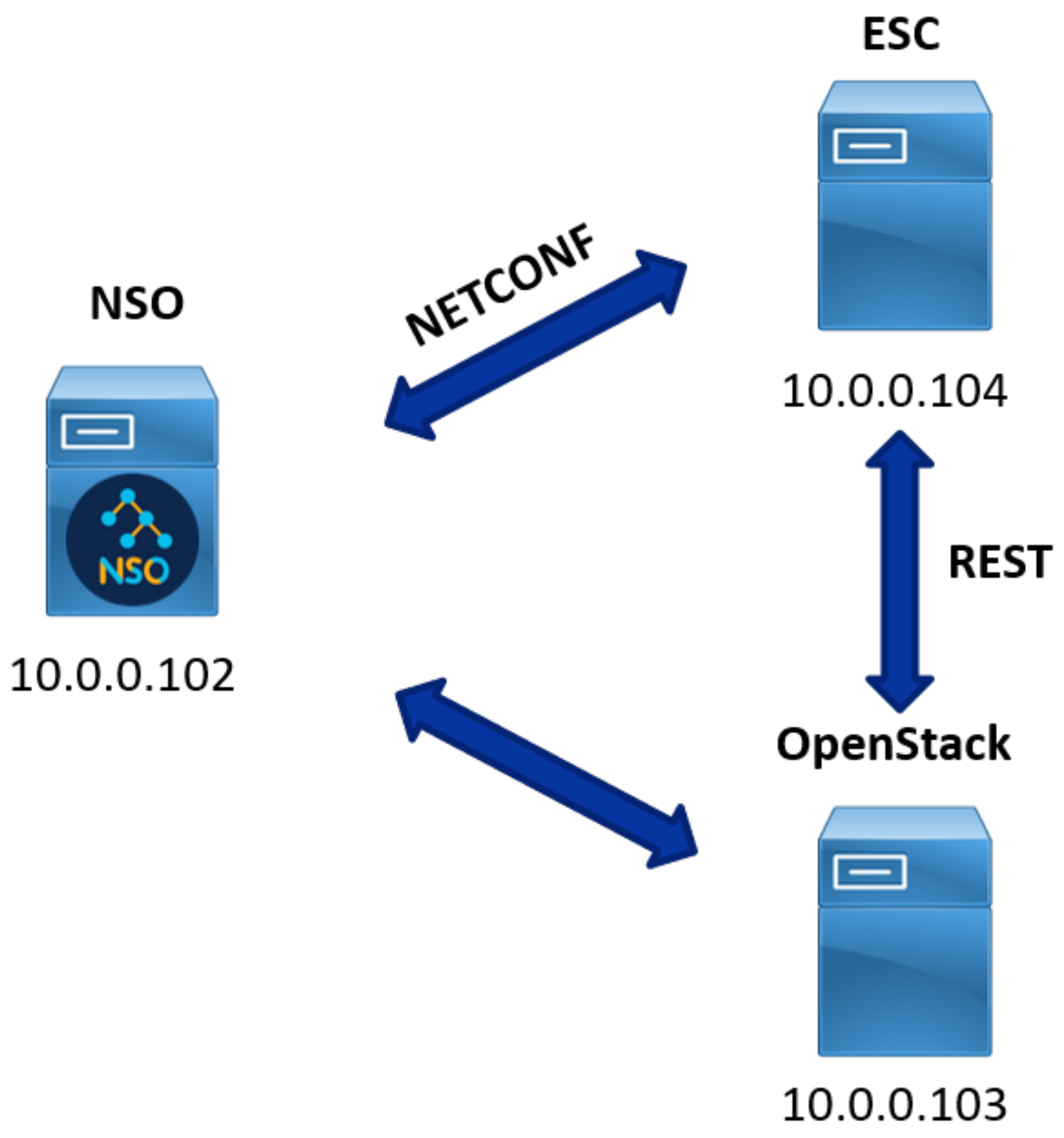
The following figure provides a visual aid for this activity.



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), an OpenStack server instance and an ESC server instance. On the Linux server within that topology is your second topology—a Docker environment, which in this activity contains only an NSO Docker image with your NSO installation.

Topology



Task 1: Modify the DMZ Service Model

In this task, you will modify the *dmz* service model to accommodate NFV management on OpenStack. You will add leaves for specifying the OpenStack tenant and networks. Also, the leaves for *csr-name* and *asa-name* need to have the config option set to false, since they will be set from Python code when NFVs are deployed.

The final solution for this lab is located in the ~solutions/packages/dmz/nfv directory. You can use it for copy-pasting longer pieces of code and as a reference point for troubleshooting your packages. The result of this lab is an NFV containing a virtual router and a virtual firewall, deployed on the ESC.

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

Enter the nso300 folder.

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

Step 4

Copy the *dmz*, *firewall*, and *router packages* from *~/packages* directory to the nso300 project *packages* folder. Also, copy the IOS and ASA in the same folder because you need them to manage VNF devices.

```
student@student-vm:~/nso300$ cp -r ../packages/dmz packages/
student@student-vm:~/nso300$ cp -r ../packages/router packages/
student@student-vm:~/nso300$ cp -r ../packages/firewall packages/
student@student-vm:~/nso300$ cp -r ../packages/cisco-ios-cli-6.54 packages/
student@student-vm:~/nso300$ cp -r ../packages/cisco-asa-cli-6.10 packages/
student@student-vm:~/nso300$ ls packages
cisco-asa-cli-6.10  cisco-ios-cli-6.54  dmz  esc  firewall  router  tailf-etsi-
rel2-nfvo
student@student-vm:~/nso300$
```

Step 5

Display the running Docker containers with the **docker ps -a** command. The NSO container exposes the 8080 port (on which the file server is running), on a random host's port. Note that port number.

```
student@student-vm:~/nso300$ docker ps -a
CONTAINER ID        IMAGE                                        COMMAND
CREATED              STATUS                        PORTS
NAMES
ccb73064494d        nso300.gitlab.local/nso300/nso:5.3.2-student   "/run-
nso.sh"       About a minute ago   Up About a minute (healthy)   0.0.0.0:32774-
>22/tcp, 0.0.0.0:32773->80/tcp, 0.0.0.0:32772->443/tcp, 0.0.0.0:32771->830/tcp,
0.0.0.0:32770->4334/tcp, 0.0.0.0:32769->5678/tcp, 0.0.0.0:32768->8080/tcp
testenv-nso300-5.3.2-student-nso
student@student-vm:~/nso300$
```

Step 6

Open and edit the *deployment-request.xml* from the *dmz* service. The template is now located at *~/nso300/packges/dmz/templates* and represents an NFV deployment request. Change the port within the URL for both day0 configuration locations to the port that you observed in the previous step.

Observe the template parameters that you will have to supply through the service model.

```
student@student-vm:~/nso300$ cat packages/dmz/templates/deployment-request.xml
<config xmlns="http://tail-f.com/ns/config/1.0">
  <nfvo xmlns="http://tail-f.com/pkg/tailf-etsi-rel2-nfvo">
    <ns-info>
      <esc xmlns="http://tail-f.com/pkg/tailf-etsi-rel2-nfvo-esc">
        <ns-info>
          <id>dmz-service-{/name}</id>
          <tenant>{/tenant}</tenant>
          <deployment-name>{/name}</deployment-name>
          <esc>esc0</esc>
          <username>{$USERNAME}</username>
          <nsd>dmz</nsd>
          <flavor>dmz-flavor</flavor>
          <instantiation-level>small</instantiation-level>
          <vnf-info>
            <vnf-profile>ASAv</vnf-profile>
            <vnfd>ASAv</vnfd>
            <vdu>
              <id>ASA</id>
              <managed/>
              <image-name>asav-9_9_2</image-name>
              <flavor-name>asav.basic</flavor-name>
              <bootup-time>1800</bootup-time>
              <recovery-wait-time>900</recovery-wait-time>
              <day0>
                <destination>day0-config</destination>
                <url>http://10.0.0.102:32768/day0/asa_config.txt</url>
                <variable>
                  <name>ADMIN_PWD</name>
                  <value>{$ASA_PWD}</value>
                </variable>
              </day0>
              <authgroup>{/asa-authgroup}</authgroup>
            </vdu>
          </vnf-info>
          <vnf-info>
            <vnf-profile>CSR1kv</vnf-profile>
            <vnfd>CSR1kv</vnfd>
            <vdu>
              <id>CSR</id>
              <managed/>
              <image-name>csrv-9_16_9</image-name>
              <flavor-name>csrv.basic</flavor-name>
              <bootup-time>1800</bootup-time>
              <recovery-wait-time>900</recovery-wait-time>
              <day0>
                <destination>iosxe_config.txt</destination>
                <url>http://10.0.0.102:32768/day0/csr_config.txt</url>
                <variable>
                  <name>ADMIN_PWD</name>
                  <value>{$CSR_PWD}</value>
                </variable>
              </day0>
              <authgroup>{/csr-authgroup}</authgroup>
            </vdu>
          </vnf-info>
          <sap-info>
            <sapd>inside</sapd>
            <network-name>{/inside-net}</network-name>
          </sap-info>
          <sap-info>
            <sapd>mgmt</sapd>
```

```
              <network-name>{/mgmt-net}</network-name>
            </sap-info>
            <sap-info>
              <sapd>outside</sapd>
              <network-name>{/outside-net}</network-name>
            </sap-info>
            <state>instantiated</state>
          </ns-info>
        </esc>
      </ns-info>
    </nfvo>
  </config>
```

Step 7

Save the file and exit the file editor.

Step 8

Open the YANG service model of the dmz package.

```
student@student-vm:~/nso300$ vi packages/dmz/src/yang/dmz.yang
```

This is how the file should appear when you open it for the first time:

```
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;
      }
    }
  }
}
```

Step 9

Set config for leaves *csr-name* and *asa-name* to false and make them part of operational data. The names will be set from Python when devices are deployed.

```
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 {
        config false;
        tailf:cdb-oper {
          tailf:persistent true;
        }
        type leafref {
          path "/ncs:devices/ncs:device/ncs:name";
        }
      }

      leaf asa-name {
        config false;
        tailf:cdb-oper {
          tailf:persistent true;
        }
        type leafref {
          path "/ncs:devices/ncs:device/ncs:name";
        }
      }

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

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

Step 10

Add new leaves to the service model, which are needed for Open stack to deploy your VNF. All of them must be mandatory.

These leaves are:

- **csr-authgroup** (must reference an existing authgroup)
- **asa-authgroup** (must reference an existing authgroup)
- **tenant**
- **mgmt-net**
- **inside-net**
- **outside-net**

```
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 {
        config false;
        tailf:cdb-oper {
          tailf:persistent true;
        }
        type leafref {
          path "/ncs:devices/ncs:device/ncs:name";
        }
      }

      leaf asa-name {
        config false;
        tailf:cdb-oper {
          tailf:persistent true;
        }
        type leafref {
          path "/ncs:devices/ncs:device/ncs:name";
        }
      }

            leaf csr-authgroup {
        mandatory true;
        type leafref {
          path "/ncs:devices/ncs:authgroups/ncs:group/ncs:name";
        }
      }

      leaf asa-authgroup {
        mandatory true;
        type leafref {
          path "/ncs:devices/ncs:authgroups/ncs:group/ncs:name";
        }
      }

      leaf tenant {
        mandatory true;
        type string;
      }

      leaf outside-net {
        mandatory true;
        type string;
      }

      leaf mgmt-net {
        mandatory true;
        type string;
      }

      leaf inside-net {
        mandatory true;
        type string;
      }

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

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

Step 11

Save the file and exit the file editor.

Activity Verification

You have completed this task when you attain these results:

- **dmz**, **router**, and **firewall** packages are included in the nso300 project.
- The **dmz** service model has been modified with new leaves being added.

Task 2: Transform the DMZ Service into a Nano Service

In this task, you will transform the existing *dmz* service into a Nano service. The NFVs are deployed on another server, which means you will need to break the existing service into two parts—a part where you request a new deployment on ESC and a part where you handle a freshly deployed NFV.

Activity

Complete these steps:

Step 1

Open the YANG service model of the **dmz** package again.

```
student@student-vm:~/nso300$ vi packages/dmz/src/yang/dmz.yang
```

Step 2

Make the **dmz** service use the *ncs:nano-plan-data* and define the two different component states – *vnf-requested and vnf-created*.

```
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;
  }

  identity dmz {
    base ncs:plan-component-type;
  }

  identity vnf-requested {
    base ncs:plan-state;
  }

  identity vnf-created {
    base ncs:plan-state;
  }

  augment /ncs:services {

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

      key name;
      leaf name {
        type string;
      }

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

      < … Output Omitted … >
```

Step 3

Create a plan outline and define create callbacks for the two states you defined in the previous step.

```
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;
  }

  identity dmz {
    base ncs:plan-component-type;
  }

  identity vnf-requested {
    base ncs:plan-state;
  }

  identity vnf-created {
    base ncs:plan-state;
  }

  ncs:plan-outline dmz-plan {
    description "DMZ Plan";

    ncs:component-type "ncs:self" {
      ncs:state "ncs:init";
      ncs:state "ncs:ready";
    }
    ncs:component-type "dmz:dmz" {
      ncs:state "ncs:init";
      ncs:state "dmz:vnf-requested" {
        ncs:create {
          ncs:nano-callback;
        }
      }
      ncs:state "dmz:vnf-created" {
        ncs:create {
          ncs:nano-callback;
        }
      }
    }
  }

  augment /ncs:services {

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

      key name;
      leaf name {
        type string;
      }

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

      < … Output Omitted … >
```

Step 4

Add a service behavior tree, where you initialize the default **self** component and your **dmz** component.

```
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;
  }

  identity dmz {
    base ncs:plan-component-type;
  }

  identity vnf-requested {
    base ncs:plan-state;
  }

  identity vnf-created {
    base ncs:plan-state;
  }

  ncs:plan-outline dmz-plan {
    description "DMZ Plan";

    ncs:component-type "ncs:self" {
      ncs:state "ncs:init";
      ncs:state "ncs:ready";
    }
    ncs:component-type "dmz:dmz" {
      ncs:state "ncs:init";
      ncs:state "dmz:vnf-requested" {
        ncs:create {
          ncs:nano-callback;
        }
      }
      ncs:state "dmz:vnf-created" {
        ncs:create {
          ncs:nano-callback;
        }
      }
    }
  }

  ncs:service-behavior-tree dmz-servicepoint {
    description "DMZ behavior tree";
    ncs:plan-outline-ref "dmz:dmz-plan";
    ncs:selector {
      ncs:create-component "'self'" {
        ncs:component-type-ref "ncs:self";
      }
      ncs:create-component "'dmz'" {
        ncs:component-type-ref "dmz:dmz";
      }
    }
  }

  augment /ncs:services {

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

      key name;
      leaf name {
        type string;
      }

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

      < … Output Omitted … >
```

Step 5

Define a pre-condition for the *vnf-created* state. Since you are using the tailf-etsi-rel2-nfvo package to deploy the VNF, you can monitor a specific state of its deployment nano plan to see when the VNFs have been successfully deployed. As noted in NFVO documentation (The PDF is located in the **~/nso300/packages/tailf-etsi-rel2-nfvo** folder), the deployment nano plan is located under XPath **"/nfvo/vnf-info/esc/vnf-deployment[tenant=admin][deployment-name=<deployment>][esc=<esc>]/plan"**. The deployment is ready when the **self** component reaches the *ncs:ready* state.

```

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;
  }

  identity dmz {
    base ncs:plan-component-type;
  }
  identity vnf-requested {
    base ncs:plan-state;
  }

  identity vnf-created {
    base ncs:plan-state;
  }

  ncs:plan-outline dmz-plan {
    description "DMZ Plan";

    ncs:component-type "ncs:self" {
      ncs:state "ncs:init";
      ncs:state "ncs:ready";
    }
    ncs:component-type "dmz:dmz" {
      ncs:state "ncs:init";
      ncs:state "dmz:vnf-requested" {
        ncs:create {
          ncs:nano-callback;
        }
      }
      ncs:state "dmz:vnf-created" {
        ncs:create {
          ncs:pre-condition {
            ncs:monitor "/nfvo/vnf-info/esc/vnf-deployment[tenant='admin']
[deployment-name=$SERVICE/name][esc='esc0']/plan/component[name='self']/
state[name='ncs:ready']" {
              ncs:trigger-expr "status = 'reached'";
            }
          }
          ncs:nano-callback;
        }
      }
    }
  }

  ncs:service-behavior-tree dmz-servicepoint {
    description "DMZ behavior tree";
    ncs:plan-outline-ref "dmz:dmz-plan";
    ncs:selector {
      ncs:create-component "'self'" {
        ncs:component-type-ref "ncs:self";
      }
      ncs:create-component "'dmz'" {
        ncs:component-type-ref "dmz:dmz";
      }
    }
  }

  augment /ncs:services {

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

      key name;
      leaf name {
        type string;
      }

      uses ncs:service-data;
      uses ncs:nano-plan-data;
      ncs:servicepoint dmz-servicepoint;
```

```
     leaf csr-name {
      config false;
      tailf:cdb-oper {
        tailf:persistent true;
      }
      type leafref {
        path "/ncs:devices/ncs:device/ncs:name";
      }
    }

    leaf asa-name {
      config false;
      tailf:cdb-oper {
        tailf:persistent true;
      }
      type leafref {
        path "/ncs:devices/ncs:device/ncs:name";
      }
    }

          leaf csr-authgroup {
      mandatory true;
      type leafref {
        path "/ncs:devices/ncs:authgroups/ncs:group/ncs:name";
      }
    }

    leaf asa-authgroup {
      mandatory true;
      type leafref {
        path "/ncs:devices/ncs:authgroups/ncs:group/ncs:name";
      }
    }

    leaf tenant {
      mandatory true;
      type string;
    }

    leaf outside-net {
      mandatory true;
      type string;
    }

    leaf mgmt-net {
      mandatory true;
      type string;
    }

    leaf inside-net {
      mandatory true;
      type string;
    }

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

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

```
        }
      }
    }
```

Step 6

Save the file and exit the file editor.

Step 7

Reload the packages using the **make testenv-build** command. Make sure all packages reload successfully.

```
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 … >

-- 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 dmz
    result true
}
reload-result {
    package esc-nc-1.0
    result true
}
reload-result {
    package firewall
    result true
}
reload-result {
    package router
    result true
}
reload-result {
    package tailf-etsi-rel2-nfvo
    result true
}
student@student-vm:~/nso300$
```

Activity Verification

You have completed this task when you attain these results:

- Your dmz service model is a nano service.

Task 3: Modify the DMZ Service-Mapping Logic

In this task, you will include the *tailf-etsi-rel2-nfvo* package with the *dmz* package and modify the *dmz* package Python-mapping logic for the service to interact with ESC in each nano callback.

Activity

Complete these steps:

Step 1

Open the *dmz* package metadata file and add *tailf-etsi-rel2-nfvo* as a required package.

```
student@student-vm:~/nso300$ vi packages/dmz/package-meta-data.xml
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>dmz</name>
  <package-version>1.0</package-version>
  <description>Generated Python package</description>
  <ncs-min-version>4.6</ncs-min-version>

  <required-package>
    <name>tailf-etsi-rel2-nfvo</name>
  </required-package>

  <required-package>
    <name>router</name>
  </required-package>

  <required-package>
    <name>firewall</name>
  </required-package>

  <component>
    <name>dmz</name>
    <application>
      <python-class-name>dmz.dmz.DMZ</python-class-name>
    </application>
  </component>
</ncs-package>
```

Step 2

Save the file and exit the file editor.

Step 3

Open the dmz.py Python file, located in the *~/nso300/packages/dmz/python/dmz* folder.

```
student@student-vm:~/nso300$ vi packages/dmz/python/dmz/dmz.py
```

The file should look as shown when you open it for the first time:

```
# -*- 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})')

        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', service.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', service.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 4

Import the supplied modules and functions from the same folder that will help you with the implementation.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
from . import nfvo_helpers
from .common import setup_crypto, authgroup_password

< … output omitted … >
```

Step 5

Define and register the nano callback functions for both custom states that your dmz service uses (vnf-requested and vnf-deployed).

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
from . import nfvo_helpers
from .common import setup_crypto, authgroup_password

class VnfRequested(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)

class VnfCreated(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)

< … Output Omitted … >

# ---------------------------------------------
# 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)
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
requested', VnfRequested)
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
created', VnfCreated)

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

Step 6

Remove the service imports, callback, and registration since you do not need them anymore. Move the service code block that applies the service configuration to the NFV devices to the VnfCreated nano callback. You can also remove the if clause that checks if the device exists, since the state pre-condition takes care of that requirement.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from . import nfvo_helpers
from .common import setup_crypto, authgroup_password

class VnfRequested(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)

class VnfCreated(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)
        template = ncs.template.Template(service.router)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.csr_name)
        template.apply('router-service-template', tvars)

        template = ncs.template.Template(service.firewall)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.asa_name)
        template.apply('firewall-service-template', tvars)

    def _set_operational_device_leaf(self, tctx, service, device_name, csr):
        with ncs.maapi.single_write_trans(tctx.username, tctx.context,
db=ncs.OPERATIONAL) as oper_th:
            oper_service = ncs.maagic.get_node(oper_th, service._path)
            self.log.info(f"Writing device name leaf for
{oper_service.csr_name}")
            if csr:
                oper_service.csr_name = device_name
            else:
                oper_service.asa_name = device_name
            oper_th.apply()

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

    def setup(self):
        self.log.info('DMZ RUNNING')
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
requested', VnfRequested)
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
created', VnfCreated)

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

Step 7

Create a helper method *_set_operational_device_leaf()* within *VnfCreated* class. It should open a new transaction and write to service configuration leaves *csr-name* and *asa-name* once you get those names from the NFVO function pack code.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from . import nfvo_helpers
from .common import setup_crypto, authgroup_password

class VnfRequested(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)

class VnfCreated(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)
        template = ncs.template.Template(service.router)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.csr_name)
        template.apply('router-service-template', tvars)

        template = ncs.template.Template(service.firewall)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.asa_name)
        template.apply('firewall-service-template', tvars)

    def _set_operational_device_leaf(self, tctx, service, device_name, csr):
        with ncs.maapi.single_write_trans(tctx.username, tctx.context,
db=ncs.OPERATIONAL) as oper_th:
            oper_service = ncs.maagic.get_node(oper_th, service._path)
            self.log.info(f"Writing device name leaf for
{oper_service.csr_name}")
            if csr:
                oper_service.csr_name = device_name
            else:
                oper_service.asa_name = device_name
            oper_th.apply()

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

    def setup(self):
        self.log.info('DMZ RUNNING')
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
requested', VnfRequested)
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
created', VnfCreated)

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

Step 8

Create a helper method *_set_operational_device_leaf()* below *cb_create()* method. It should open a new transaction and write to service configuration leaves *csr-name* and *asa-name* once you get those names from the NFVO function pack code.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from . import nfvo_helpers
from .common import setup_crypto, authgroup_password

class VnfRequested(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)

class VnfCreated(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)
        template = ncs.template.Template(service.router)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.csr_name)
        template.apply('router-service-template', tvars)

        template = ncs.template.Template(service.firewall)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.asa_name)
        template.apply('firewall-service-template', tvars)

    def _set_operational_device_leaf(self, tctx, service, device_name, csr):
        with ncs.maapi.single_write_trans(tctx.username, tctx.context,
db=ncs.OPERATIONAL) as oper_th:
            oper_service = ncs.maagic.get_node(oper_th, service._path)
            self.log.info(f"Writing device name leaf for
{oper_service.csr_name}")
            if csr:
                oper_service.csr_name = device_name
            else:
                oper_service.asa_name = device_name
            oper_th.apply()

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

    def setup(self):
        self.log.info('DMZ RUNNING')
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
requested', VnfRequested)
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
created', VnfCreated)

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

Step 9

Use the methods from *nfvo_helpers* to get a list of devices within a deployment. Use the newly created *_set_operational_device_leaf()* method to set the operational NFV device names. You can also add some logging for easier troubleshooting of callbacks.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from . import nfvo_helpers
from .common import setup_crypto, authgroup_password

class VnfRequested(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)

class VnfCreated(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)
        ns_info_name = f"dmz-service-{service.name}"
        devices = nfvo_helpers.ns_devices(tctx, ns_info_name)
        self.log.info(f"devices = {devices}")

        device_name = devices[('CSR1kv', 'CSR')][0]
        self._set_operational_device_leaf(tctx, service, device_name, True)
        self.log.info(f"Router: CSR name {device_name}")
        template = ncs.template.Template(service.router)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.csr_name)
        template.apply('router-service-template', tvars)

        device_name = devices[('ASAv', 'ASA')][0]
        self._set_operational_device_leaf(tctx, service, device_name, False)
        self.log.info(f"FW: ASA name {device_name}")
        template = ncs.template.Template(service.firewall)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.asa_name)
        template.apply('firewall-service-template', tvars)

        self.log.info('Successfully configured DMZ service instance')

    def _set_operational_device_leaf(self, tctx, service, device_name, csr):
        with ncs.maapi.single_write_trans(tctx.username, tctx.context,
db=ncs.OPERATIONAL) as oper_th:
            oper_service = ncs.maagic.get_node(oper_th, service._path)
            self.log.info(f"Writing device name leaf for
{oper_service.csr_name}")
            if csr:
                oper_service.csr_name = device_name
            else:
                oper_service.asa_name = device_name
            oper_th.apply()

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

    def setup(self):
        self.log.info('DMZ RUNNING')
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
requested', VnfRequested)
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
created', VnfCreated)

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

Step 10

Add the code to request a VNF within the nano create callback for VnfRequested class, using the *deployment-request* template. Since NSO encrypts the passwords, stored in the */devices/ authgroups* subtree, use *setup_crypto* and *authgroup_passwords* methods to help you decrypt it and apply it to the template.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from . import nfvo_helpers
from .common import setup_crypto, authgroup_password

class VnfRequested(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)

        self.template = ncs.template.Template(service)
        vars = ncs.template.Variables()
        vars.add('USERNAME', tctx.username)

        setup_crypto(tctx)
        csr_pwd = authgroup_password(tctx, root, service.csr_authgroup)
        vars.add('CSR_PWD', csr_pwd)
        asa_pwd = authgroup_password(tctx, root, service.asa_authgroup)
        vars.add('ASA_PWD', asa_pwd)

        self.template.apply('deployment-request', vars)

class VnfCreated(ncs.application.NanoService):
    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
proplist, component_proplist):
        self.log.debug("NanoService create ", state)

        ns_info_name = f"dmz-service-{service.name}"
        devices = nfvo_helpers.ns_devices(tctx, ns_info_name)
        self.log.info(f"devices = {devices}")

        device_name = devices[('CSR1kv', 'CSR')][0]
        self._set_operational_device_leaf(tctx, service, device_name, True)
        self.log.info(f"Router: CSR name {device_name}")
        template = ncs.template.Template(service.router)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.csr_name)
        template.apply('router-service-template', tvars)

        device_name = devices[('ASAv', 'ASA')][0]
        self._set_operational_device_leaf(tctx, service, device_name, False)
        self.log.info(f"FW: ASA name {device_name}")
        template = ncs.template.Template(service.firewall)
        tvars = ncs.template.Variables()
        tvars.add('NAME', service.name)
        tvars.add('DEVICE', service.asa_name)
        template.apply('firewall-service-template', tvars)

        self.log.info('Successfully configured DMZ service instance')

    def _set_operational_device_leaf(self, tctx, service, device_name, csr):
        with ncs.maapi.single_write_trans(tctx.username, tctx.context,
db=ncs.OPERATIONAL) as oper_th:
            oper_service = ncs.maagic.get_node(oper_th, service._path)
            self.log.info(f"Writing device name leaf for
{oper_service.csr_name}")
            if csr:
                oper_service.csr_name = device_name
            else:
                oper_service.asa_name = device_name
            oper_th.apply()

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

    def setup(self):
        self.log.info('DMZ RUNNING')
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
requested', VnfRequested)
        self.register_nano_service('dmz-servicepoint', 'dmz:dmz', 'dmz:vnf-
created', VnfCreated)

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

Step 11

Save the file and exit the file editor.

Step 12

Reload the packages using the **make testenv-build** command. Make the *dmz* package is reloaded successfully.

```
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 … >

-- 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 dmz
    result true
}
reload-result {
    package esc-nc-1.0
    result true
}
reload-result {
    package firewall
    result true
}
reload-result {
    package router
    result true
}
reload-result {
    package tailf-etsi-rel2-nfvo
    result true
}
student@student-vm:~/nso300$
```

Activity Verification

You have completed this task when you attain these results:

- You have modified the *dmz* service Python-mapping code to work with nano services.
- The *dmz* package has been successfully reloaded.

Task 4: Configure NSO to Work with ESC and OpenStack

At this point, the DMZ NFV service should be completely functional. You will now deploy a new service instance.

Activity

Complete these steps:

Step 1

Connect to the NSO CLI, switch CLI mode and enter configuration 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 6e4e800fcc5d
admin@ncs> switch cli
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)#
```

Step 2

Add a secondary password for privileged mode to *asa* and *csr* authgroups. Use the password *admin* and commit the changes.

```
admin@ncs(config)# devices authgroups group asa default-map remote-secondary-
password admin
admin@ncs(config-group-asa)# top
admin@ncs(config)# devices authgroups group csr default-map remote-secondary-
password admin
admin@ncs(config-group-csr)# commit
Commit complete.
admin@ncs(config-group-csr)# top
```

Step 3

Create a new *dmz* service instance. Use the **tenant admin** and some pre-defined networks on OpenStack – **internet** for inside network, **mgmt** for management network and **client** for inside network. Add a route to the **172.10.0.0/16** network with **10.0.10.1** as the gateway.

```
admin@ncs(config)# services dmz DMZ1 asa-authgroup asa csr-authgroup csr tenant
admin inside-net client mgmt-net mgmt outside-net internet
admin@ncs(config-dmz-DMZ1)# router routes 172.10.0.0 255.255.0.0 gateway
10.0.10.1
admin@ncs(config-routes-10.0.100.0/255.255.255.0)# top
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# exit
```

Step 4

You can verify and monitor the progress and the status of the NFVO nano plan by using the **show nfvo** command. You will notice multiple commits being performed as NSO system messages. This is the NSO reacting to nanoservice state changes.

The deployment of NFVs can take some time—images need to be copied, started, the device OS needs to boot up and day0 configuration needs to be applied. The whole process takes at least 10 minutes. It is possible for the status to become "failed" temporarily, since ESC might check if the VM is alive after it has been deployed but before it has fully booted. Ready state should be reached eventually since ESC periodically retries to reach the VMs.

```
admin@ncs# show nfvo
NAME        TYPE  STATE     STATUS       WHEN                 ref
-------------------------------------------------------------------
self        self  init      reached      2020-09-11T08:38:11  -
                  ready     not-reached  -                    -
ASAv-ASA    vdu   init      reached      2020-09-11T08:38:11  -
                  deployed  not-reached  -                    -
                  ready     not-reached  -                    -
CSR1kv-CSR  vdu   init      reached      2020-09-11T08:38:11  -
                  deployed  not-reached  -                    -
                  ready     not-reached  -                    -

< … output omitted … >
```

Step 5

You can verify the progress and the status of your DMZ service's nano plan by using the **show services dmz DMZ1 plan | tab** command.

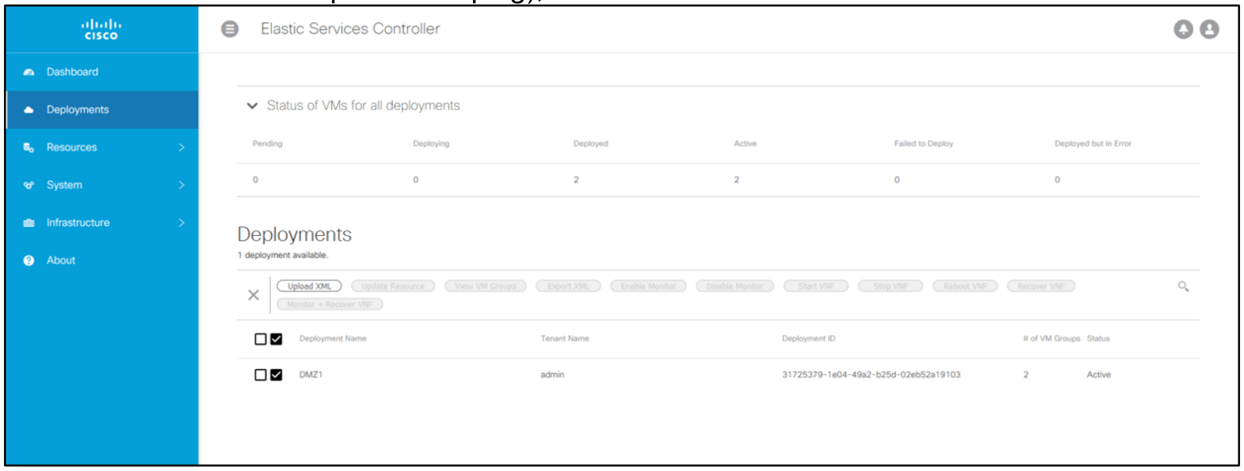
```
admin@ncs# show services dmz DMZ1 plan | tab

POST
            BACK
ACTION
TYPE  NAME  TRACK  GOAL  STATE          STATUS       WHEN                 ref
STATUS
-------------------------------------------------------------------------------------
self  self  false  -     init           reached      2020-09-11T08:38:11  -
-
                         ready          reached      2020-09-11T08:38:11  -
-
dmz   dmz   false  -     init           reached      2020-09-11T08:38:11  -
-
                         vnf-requested  reached      2020-09-11T08:38:11  -
-
                         vnf-created    not-reached  -                    -
```

```
-
```

Step 6

Open a web browser and navigate to https://10.0.0.104. Log in with credentials admin/admin and click on the Deployments on the left navigation bar. Wait until the deployment DMZ1 reaches "Active" state.

Other states that you can observe are—Deploying, Inert (deployment finished, but waiting for the device to become responsive to ping), and Error.



Status of VMs for all deployments

Pending	Deploying	Deployed	Active	Failed to Deploy	Deployed but in Error
0	0	2	2	0	0

Deployments

1 deployment available.

Deployment Name	Tenant Name	Deployment ID	# of VM Groups	Status
DMZ1	admin	31725379-1e04-49a2-b25d-02eb52a19103	2	Active

To see what is happening on devices, you can use a browser to navigate to the OpenStack Web UI (10.0.0.103) and log in with credentials admin/admin. To connect to NFV devices CLI, use the navigation bar on the left side to locate instances under Admin ▯ Compute ▯ Instances. By clicking on an instance, and then selecting Console, you open a CLI session within your browser to the device for easier troubleshooting.

Step 7

Wait for the state of your DMZ1 service instance nano plan to reach vnf-created.

```
admin@ncs# show services dmz DMZ1 plan | tab

POST
            BACK
ACTION
TYPE  NAME  TRACK  GOAL  STATE          STATUS       WHEN                 ref
STATUS
-----------------------------------------------------------------------------------------
self  self  false  -     init           reached      2020-09-11T08:38:11  -
-
                         ready          reached      2020-09-11T08:38:11  -
-
dmz   dmz   false  -     init           reached      2020-09-11T08:38:11  -
-
                         vnf-requested  reached      2020-09-11T08:38:11  -
-
                         vnf-created    reached      2020-09-11T08:55:31  -
-
```

Step 8

Return to the terminal window with the open NSO CLI session. Display the devices added to NSO with **show devices brief** command. IP addresses of NFV devices might be slightly different, since they are assigned by DHCP.

```
admin@ncs# show devices brief
NAME                          ADDRESS       DESCRIPTION  NED ID
-----------------------------------------------------------------------------
admin-DMZ1-ASAv-ASA-esc0-1    10.0.0.160    -            cisco-asa-cli-6.10
admin-DMZ1-CSR1kv-CSR-esc0-1  10.0.0.221    -            cisco-ios-cli-6.54
esc0                          10.0.0.104    -            esc-nc-1.0
admin@ncs#
```

Step 9

Verify that the custom router configuration has been applied to the NFV CSR device and exit the NSO CLI.

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# show full-configuration devices device admin-DMZ1-CSR1kv-
CSR-esc0-1 config ip route
devices device admin-DMZ1-CSR1kv-CSR-esc0-1
 config
  ip route 0.0.0.0 0.0.0.0 10.0.0.1
  ip route 172.10.0.0 255.255.0.0 10.0.10.1
 !
!
admin@ncs(config)#

admin@ncs(config)# abort
admin@ncs# exit
```

Step 10

Verify that the custom router configuration has been applied to the actual device by establishing an SSH session (use the admin/admin credentials) and displaying the configuration with the **show ip route** command. Make sure to use the correct IP of the CSR device.

```
student@student-vm:~/nso300$ ssh admin@10.0.0.221
Password:

host-10-0-0-221>enable
Password:

host-10-0-0-221#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is 10.0.0.1 to network 0.0.0.0
```

```
S*    0.0.0.0/0 [1/0] via 10.0.0.1
      10.0.0.0/8 is variably subnetted, 6 subnets, 2 masks
C        10.0.0.0/24 is directly connected, GigabitEthernet1
L        10.0.0.221/32 is directly connected, GigabitEthernet1
C        10.0.10.0/24 is directly connected, GigabitEthernet2
L        10.0.10.195/32 is directly connected, GigabitEthernet2
C        10.0.20.0/24 is directly connected, GigabitEthernet3
L        10.0.20.195/32 is directly connected, GigabitEthernet3
S     172.10.0.0/16 [1/0] via 10.0.10.1
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully deployed a dmz service instance.
- A VNF device has been successfully configured.