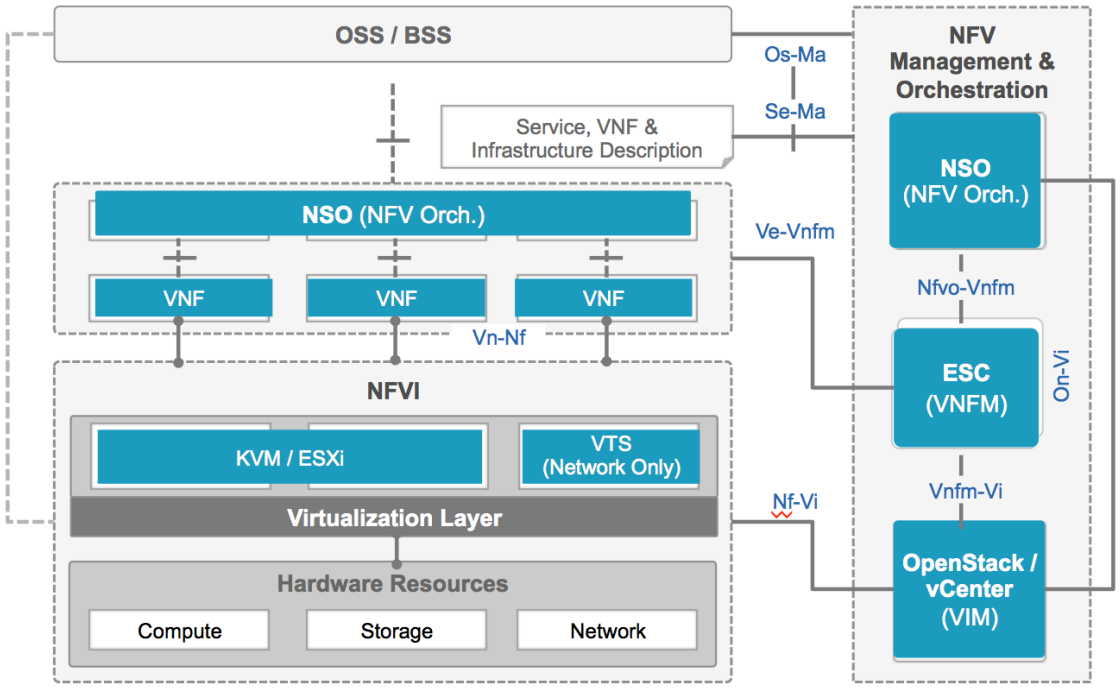
Using Elastic Services Controller with NSO

Introduction

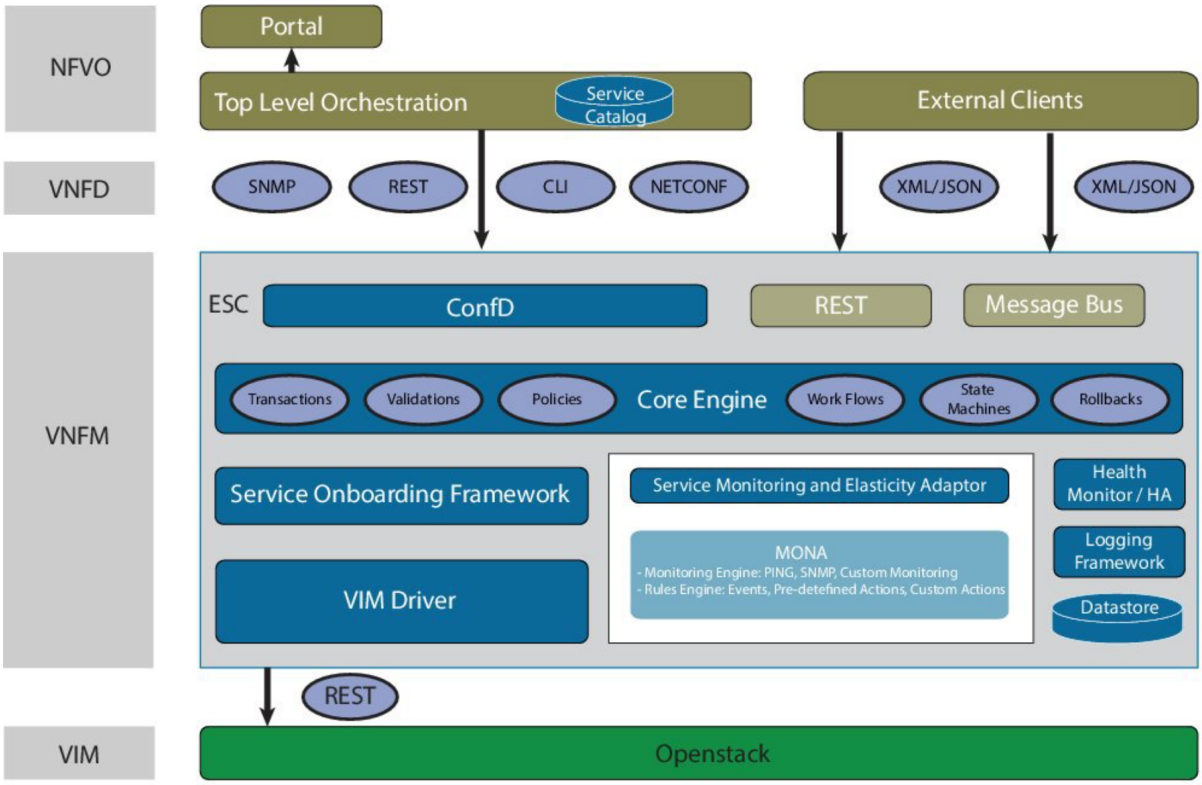
This document describes how to use NSO and Elastic Services Controller, ESC, together. Mapped to the ETSI NFV Architecture NSO performs the OSS and NFVO functionality and the ESC implements the VNF-M functionality. NSO and ESC communicates over a YANG/NETCONF interface (Nfvo-Vnfm).

Figure 1. The ETSI NFV Architecture



ESC manages on-boarding and monitoring of the VMs.

Figure 2. ESC Architecture



Overview

Cisco Elastic Services Controller (ESC) is a Virtual Network Functions Manager (VNFM), performing life cycle management of Virtual Network Functions (VNFs). ESC provides agent-less and multi vendor VNF management by provisioning the virtual services, and monitoring their health and load. ESC provides the flexibility to define rules for monitoring, and associate actions to be triggered based on the outcome of these rules. Based on the monitoring results, ESC performs scale in or scale out on the VNFs. It also supports automatic VM recovery when a VM fails.

NSO manages the end-customer specific service configuration on top of the running virtual machines. This covers both the NFVO and the OSS part. An example is that ESC manages to start a virtual PE and its initial configuration and then NSO will create a customer specific VPN over that vPE.

There is a bit of terminology confusion looking at the two roles NSO and ESC takes. ETSI MANO uses the term network service to refer to a set of started VMs. In the telco world the term service is more generally referring to end-customer specific service instances possible running over VMs.

ESC manages the ETSI definition of a service: a group of managed VMs. NSO manages the telco oriented definition of a service and also delegates the need for any ETSI MANO services to ESC. So in a Cloud VPN use case for example: the self-service portal requests NSO to create a VPN. NSO might detect that a virtual PE is

needed in a data-center, NSO will then request ESC to start that virtual PE as part of the Service Orchestration. There are several benefits of managing the complete orchestration request as one transaction via NSO:

- The complete service (VPN service and MANO vPE service) is managed in one YANG model and one transaction.
- Any life-cycle changes can impact the two service contexts: a healing event in MANO need to affect the VPN service and vice versa.

You could choose to keep the MANO part separate from the OSS part. In that case NSO/ESC would start a virtual PE and consider that part done. Then as a second step the service configuration is performed. But then you loose all the benefits above.

So the way to best utilize NSO and ESC together is to view Virtual Machines as a result from a service activation request in the OSS layer. NSO manages the complete end-customer service activation requests which covers physical devices, other support systems and often virtual network functions, all in one piece.

ESC Concepts

- Registering/Unregistering: this corresponds to VNF Package on-boarding. In order for ESC to start a VM it needs the image and OpenStack flavor information, ETSI VNFD. You can also register VNFs, a VNF with different VMs. Registration corresponds to the "service" part of the ESC YANG data-model.
- Deploying/Undeploying: a registered VNF can be deployed which makes the VMs running. Deployment includes day-0 configuration, affinity policies etc. Deployment corresponds to the "tenant" part of the ESC data-model.
- Notifications: ESC sends NETCONF notifications that NSO needs to react to and manage in the Reactive FASTMAP loop. At VNF deployment ESC sends a VM_ALIVE notification when the VM is started. There are also scaling and healing events that need to trigger Reactive FASTMAP.

Make sure you read the "Cisco Elastic Services Controller User Guide" to have a good understanding of all these concepts before continuing.

NSO Concepts

A "service" in NSO has a different meaning than in the ESC data-model and in the ETSI NFV MANO specification. In NSO a service refers to run-time configuration for a specific service instance for a specific user. This often spans physical and virtual devices. The NSO service-model defines the input parameters for the service. In the ETSI MANO specification this corresponds to the OSS layer. The mapping from the input parameters defined by the service model to the network is defined in NSO mapping logic. Mapping logic can be Java/Python or configuration templates, most often a combination of both. What is important to understand is that this mapping includes registering and deploying a VNF as a "sub-step". VMs are needed since we want to provision an end-to-end service.

NSO uses state-based convergence for the mapping logic, NSO moves the service to its final desired state, fully provisioned including started fully configured VMs. The algorithm behind the scenes is called Reactive FASTMAP, RFM. The basic modus operandi of RFM in virtual use cases is that NSO at first iteration realizes that VMs need to be started; NSO registers the VNF and requests ESC to deploy it, at this point NSO exits the first iteration in the state convergence process. When ESC later sends the VM_ALIVE notification the service is "redeployed" which

triggers the state convergence for the service again, at this point the state convergence passes the VM started state and probably continues with VNF run-time configuration. Several different events might trigger further service redeploy operations. For example scaling and healing events need to trigger redeploy so that NSO can configure, reconfigure, remove configuration etc based on the current state in the virtual infra-structure.

Make sure you read the section "Developing NCS Services" in the "NSO Development Guide". Pay special attention to subsections at the end on Reactive FASTMAP and virtual devices.

The VM Manager Package

The section above explained how NSO manages VMs using ESC. The implementation of this is simplified by a NSO Package called "VM Manager". By using this package NSO does not communicate with ESC (or any other VNF Manager), the mechanisms for notifications and Reactive FASTMAP are encapsulated into the VM Manager package. More information on the VM Manager package can be found in the NSO Developers Guide in the section on "Services that involve virtual devices, NFV ".

Putting the pieces together

The example `examples.ncs/service-provider/virtual-mpls-vpn` implements a "virtual" MPLS VPN. It uses the VM Manager Package and a NED for ESC. The example also contains a simple simulated ESC in order to get the events for state changes that trigger Reactive FASTMAP. The use case is that whenever NSO detects that a physical CE needs a virtual PE it requests ESC to start a virtual PE in the data-center. Study this example carefully to get an understanding of how to implement similar applications.

Registering and deploying VNFs (ESC Services)

The service model for the MPLS VPN takes an input parameter that picks an available VNF descriptor (esc registered services) to be used for a potential vPE. The example starts with a set of pre-registered services:

```
ncs# show full-configuration devices device esc0 config esc:esc_datamodel services
devices device esc0
 config
  esc:esc_datamodel services service_definition PE 1.1
   vm_group CSR
    config_data configuration pe-cfg
     file http://www.example.com/csr-image.iso
    !
    scaling min_active 1
    scaling max_active 1
   !
  !
  esc:esc_datamodel services service_definition PE2 0.9
   vm_group CSR
    config_data configuration pe-cfg
     file http://www.example.com/csr-old.iso
    !
    scaling min_active 1
    scaling max_active 6
   !
  !
  esc:esc_datamodel services service_definition vbranch-csr-service 1.0
   vm_group CSR
    bootup_time        600
```

```
    recovery_wait_time 0
    disk src         file://cisco/images/csr1000v-universalk9.03.13.00a.S.154-3.S0a-ext.qcow2
    disk disk_format qcow2
    disk container_format bare
    disk serial_console true
    disk e1000_net   true
    disk disk_bus    virtio
    vm_flavor vcpus   2
    vm_flavor memory_mb 4096
    vm_flavor root_disk_mb 0
    vm_flavor ephemeral_disk_mb 0
    vm_flavor swap_disk_mb 0
```

As can be seen in the CLI command above ESC is a device in NSO. Registering VNFs are done at the path: `/devices/device/esc0/config/esc:esc_datamodel/services`

The configuration of the VPN takes a registered service as input:

```
vpn l3vpn volvo
  as-number 65101
  endpoint main-office
    ce-device    ce0
      ce-interface GigabitEthernet0/1
      ip-network   10.10.1.0/24
      bandwidth    12000000
!
endpoint branch-office2
   ce-device    ce7
     ce-interface GigabitEthernet0/1
     ip-network   10.8.8.0/24
     bandwidth    300000
!
esc_service name vbranch-csr-service
esc_service version 1.0
esc_service vm_group CSR
!
```

The endpoint `branch-office2` uses CE `ce7` which requires a virtual PE to be started in the data-center. This virtual PE will be deployed using the registered ESC service. The deployed service shows up as a tenant:

```
ncs(config)# show full-configuration devices device esc0 config esc:esc_datamodel tenants
devices device esc0
 config
  esc:esc_datamodel tenants tenant volvo
   services service_definition vpn vbranch-csr-service 1.0
    vm_group CSR
     interfaces interface 0
      network    net_osmgmt
      ip_address 10.86.22.224
     !
     interfaces interface 1
      network    internet
      ip_address 10.1.4.10
     !
     interfaces interface 2
      network    dmz
```

```
      ip_address 192.168.3.10
     !
     kpi_data kpi VM_ALIVE
      metric_value 60
      metric_cond  GT
      metric_type  UINT32
      metric_collector type ICMPPing
      metric_collector nicid 0
      metric_collector poll_frequency 3
      metric_collector polling_unit seconds
      metric_collector continuous_alarm false
     !
     rules admin_rules rule VM_ALIVE
      action [ "'ALWAYS log'" "'TRUE servicebooted.sh'" ]
     !
     config_data configuration iosxe_config.txt
      file file://cisco/images/CSR.txt
     !
     scaling min_active 1
     scaling max_active 3
     scaling elastic true
     scaling static_ip_address_pool dmz
      ip_address [ 192.168.3.10 192.168.3.11 ]
     !
     scaling static_ip_address_pool internet
      ip_address [ 10.1.4.10 10.1.4.11 ]
     !
     scaling static_ip_address_pool net_osmgmt
      ip_address [ 10.86.22.224 10.86.22.225 ]
     !
    !
   !
  !
 !
!
```

Reactive FASTMAP, the VM Manager package

Reactive FASTMAP is a design pattern that is used to provision services that involves side-effects and steps that takes a long time, for example requesting ESC/OpenStack to start a VM. Read more on Reactive FASTMAP in the NSO Developers Guide.

The overall steps to create a virtual devices are:

1. Instructing ESC to start the virtual device with some input parameters (which image, cpu settings, day0 configuration etc).
2. Waiting for the virtual device to be started, the VIM/VNF-M may signal this through some event, or polling of some state might be necessary.
3. Mount the device in the NCS device tree
4. Fetch ssh-keys and perform sync-from on the newly created device

Applying reactive FASTMAP to the steps of starting a VM is outlined in pseudo code below:

```
create(Serv) {
```

```
VNFs = figure_out_which_VMs(Serv)
for N in VNFs {
   vms[N] = requestVM(N)
}
for N in VNFs {
   if (! readyVMs(VMs[N]))
     return;
}
```

According to the Reactive FASTMAP pattern the `requestVM` writes CDB configuration data to indicate the VM create request. A CDB subscriber triggers on that requests and makes the call to ESC. Once the VM is up and running (in the ESC case detected by a notification) a call is made to service redeploy which will run "another instance" of the create method above. In this way the service will converge to a state where all VMs are up and running.

There is a NSO package "VM Manager" that implements a general interface for the above. The backend of the package manages the Reactive FASTMAP implementation. There are specific backends per VNF Manager. The virtual mpls vpn example shows how to use this package together with ESC.

Building the ESC NED

ESC supports a northbound NETCONF/YANG interface. This means that NSO can talk native NETCONF towards ESC. This section walks you through the required steps to build a ESC NED.

The ESC YANG files can be found in the ESC installation directory at `esc-confd/YANGmodels-tailf`.

Procedure 1. Building the ESC NED

Step 1 Prepare a directory with the ESC YANG files

Step 2 Run ncs-make-package to build a NETCONF NED based on these YANG files.

Step 3 **make** the NED package

Step 4 Edit the metadata file for the generated NED

This steps are described below.

Prepare a directory with the ESC YANG files somewhere, say `~/esc/esc-yang`. It should look something like this:

```
$ls -l
 -rw-r--r--  1 svallin  staff  5491 Jun  9 14:31 esc.yang
-rw-r--r--  1 svallin  staff  3565 Jun  9 14:31 esc_config_data.yang
-rw-r--r--  1 svallin  staff  5411 Jun  9 14:31 esc_datamodel.yang
-rw-r--r--  1 svallin  staff  2285 Jun  9 14:31 esc_dependencies.yang
-rw-r--r--  1 svallin  staff  3088 Jun  9 14:31 esc_disk.yang
-rw-r--r--  1 svallin  staff  2581 Jun  9 14:31 esc_flavor.yang
-rw-r--r--  1 svallin  staff  3209 Jun  9 14:31 esc_interface.yang
-rw-r--r--  1 svallin  staff  6085 Jun  9 14:31 esc_kpi.yang
-rw-r--r--  1 svallin  staff  2989 Jun  9 14:31 esc_network.yang
-rw-r--r--  1 svallin  staff  4286 Jun  9 14:31 esc_notifications.yang
-rw-r--r--  1 svallin  staff   920 Jun  9 14:31 esc_opdata.yang
-rw-r--r--  1 svallin  staff  7101 Jun  9 14:31 esc_opdata_devstats.yang
```

```
-rw-r--r--  1 svallin  staff  1685 Jun  9 14:31 esc_opdata_interface.yang
-rw-r--r--  1 svallin  staff  2255 Jun  9 14:31 esc_opdata_networks.yang
-rw-r--r--  1 svallin  staff  1166 Jun  9 14:31 esc_opdata_state_machines.yang
-rw-r--r--  1 svallin  staff  3627 Jun  9 14:31 esc_opdata_tenant.yang
-rw-r--r--  1 svallin  staff  6495 Jun  9 14:31 esc_policies.yang
-rw-r--r--  1 svallin  staff  2259 Jun  9 14:31 esc_rules.yang
-rw-r--r--  1 svallin  staff  5230 Jun  9 14:31 esc_scaling.yang
-rw-r--r--  1 svallin  staff  9684 Jun  9 14:31 esc_types.yang
-rw-r--r--  1 svallin  staff  3579 Jun  9 14:31 esc_volume.yang
```

Next step is to build the package based on these files. Create another directory where the NED Package will be created:

```
$ mkdir ~/esc/esc-ned
$ cd ~/esc
```

Now create the NED package like this:

```
$ ncs-make-package --netconf-ned esc-yang --dest esc-ned esc
```

The command above does:

- Create a NETCONF NED
- Store the resulting NED Package in `esc-ned`.
- Read YANG files from `esc-yang`.
- Call the NED Package `esc`.

The final step is now to **make** the ESC NED package. This step builds the necessary load files (fxs) for NSO based on the generated package. In order to do this change directory to the `src` directory in the generated NED. And then type **make** to build the package.

```
$ cd esc/src/
$ make

...
/Users/svallin/dev/trunk/ncs_dir/netsim/confd/bin/confdc --yangpath ../src/yang \
          `ls ../src/yang/esc_volume-ann.yang  > /dev/null 2>  \
              echo "-a ../src/yang/esc_volume-ann.yang"` \
          -c -o esc_volume.fxs  ../src/yang/esc_volume.yang
```

The final step is to modify the metadata-file for the NED. This file is located in the root directory of the generated NED.

```
$ ls
```

```
$ less meta-data.xml
```

```
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
```

```
  <name>esc</name>
  <package-version>1.0</package-version>
  <description>Generated netconf package</description>
  <ncs-min-version>2.2</ncs-min-version>
  <component>
    <name>esc</name>
    <ned>
      <netconf/>
      <device>
        <vendor>Acme</vendor>
      </device>
    </ned>
  </component>
</ncs-package></programlisting>
```

Modify that file in your favorite editor and change the fields. This is just for display purposes and has no impact on functionality

Using the ESC NED in NSO

The above described process generates a NED package for NSO. Add that package to your NSO installation. In your NSO run-time directory add it to the `packages` directory. Then request NSO to reload packages or restart NSO with option **with-package-reload**.

American Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam.
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at `www.cisco.com/go/offices`.