Upgrading the VNF Software Using LCS

ESC supports upgrading the VNF software application while updating a deployment. Using the policy datamodel, new Lifecycle Stages (conditions) are introduced to support the VNF upgrade. The VNF upgrade policies can be different for different VM groups. These policies are applicable for a group of VMs, and can be specified under <vm_group> rather than the entire deployment.

Upgrading VNF Software

ESC supports upgrading the initial or base image in a deployment. The ESC policy framework provides custom scripts to upgrade the software for new and existing VMs. Incremental updates are supported for the VMs, provided the ESC policy frameworks are up-to-date.

- **Upgrading Existing VMs**—The following ESC policy framework triggers script for upgrading existing VMs already deployed before the software version update.

  ```
  LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED
  ```

- **Upgrading New VMs**—The following ESC policy framework triggers script for upgrading new VMs when deployed, being recovered, or when scaling out.

  ```
  LCS::DEPLOY::POST_VM_ALIVE
  ```

For information on VNF Upgrade with Volume, see Upgrading VNF Software with Volume.

Updating VNF Software Version and triggering Software Upgrade

The scenario explains the procedure to trigger a software upgrade using the custom script. A CSR VM is upgraded in the example below. The service update using the `csr_dep2.xml` triggers the custom script action LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED. The LCS first disables monitoring of that VM, and then calls the `csr_upgrade.exp` script. The script connects to the CSR, scp's the specified upgrade .bin to the boot flash of the CSR, points the boot loader to that new bin file, and reboots the CSR VM. It then resets the bootup_time and enables monitoring. The bootup_time allows the CSR to finish rebooting without being redeployed by ESC.

Procedure

Step 1 Deploy the ESC VM.

Step 2 Upload the Day 0 configuration to the ESC VM as `/var/tmp/csp-csr-day0-config`.

Step 3 Upload the custom upgrade script to the ESC VM. For example, upload csr_upgrade.exp script to the ESC VM as `/var/tmp/csr_upgrade.exp`.

Step 4 Execute `chmod +x /var/tmp/csr_upgrade.exp`.

Step 5 Edit the initial deployment data model, for example dep.xml to include relevant IPs, username, password, and the upgrade version of the CSR.

Step 6 Edit the deployment data model's (dep.xml's) software version to reflect the upgraded CSR version.

Step 7 Upload the CSR upgrade to the home directory of the ESC user.

Step 8 Upgrade the deployed CSR VM. Run the command: `esc_nc_cli edit-config csr_dep2.xml`

Upgrading VNF Software with Volume

When a service is initially deployed, the data model has the policies configured for future software upgrade. When a deployment update request is received, VM upgrade is initiated as part of deployment update. LCS::DEPLOY_UPDATE::VM_PRE_VOLUME_DETACH is triggered before ESC detaches a volume. A script is supported at this lifecycle stage to unmount the volume before it is detached. ESC detaches and deletes the old volume which contains the old version of the software. After the volume is detached successfully, LCS::DEPLOY_UPDATE::VM_POST_VOLUME_DETACHED is triggered. A script is run at this LCS for further clean ups. When the new volume with a newer software version is attached, LCS::DEPLOY_UPDATE::VM_VOLUME_ATTACHED is triggered. ESC creates and attaches the new volume which contains the new version of the software. A script is run to mount the volume and trigger software installation. Once the volume is attached, LCS::DEPLOY_UPDATE::VM_SOFTWARE_VERSION_UPDATED is triggered after ESC has updated the software version of the VM. A script is run at this stage to complete the configuration for the software upgrade.

Data model for VNF Software Upgrade:

```
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>test</name>
      <deployments>
        <deployment>
          <name>dep</name>
          <vm_group>
            <name>Group1</name>
            <volumes>
              <volume nc:operation="delete">
                <name>v1.0</name>
                <volid>0</volid>
              </volume>
              <volume>
                <name>v2.0</name>
                <volid>1</volid>
                <sizeunit>GiB</sizeunit>
```

```
            <size>2</size>
            <bus>virtio</bus>
            <type>lvm</type>
            <image>Image-v2</image>
          </volume>
        </volumes>
        <software_version>2.0</software_version>
        <policies>
          <policy>
            <name>SVU1</name>
            <conditions>
              <condition>
                <name>LCS::DEPLOY_UPDATE::PRE_VM_VOLUME_DETACH</name>
              </condition>
            </conditions>
            <actions>
              <action>
                <name>LOG</name>
                <type>pre_defined</type>
              </action>
            </actions>
          </policy>
          <policy>
            <name>SVU2</name>
            <conditions>
              <condition>
                <name>LCS::DEPLOY_UPDATE::POST_VM_VOLUME_ATTACHED</name>
              </condition>
            </conditions>
            <actions>
              <action>
                <name>LOG</name>
                <type>pre_defined</type>
              </action>
            </actions>
          </policy>
          <policy>
            <name>SVU3</name>
            <conditions>
              <condition>
                <name>LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED</name>
              </condition>
            </conditions>
            <actions>
              <action>
                <name>LOG</name>
                <type>pre_defined</type>
              </action>
            </actions>
          </policy>
        </policies>
      </vm_group>
    </deployment>
  </deployments>
 </tenant>
</tenants>
</esc_datamodel>
```

In this data model, the existing volume v1.0 with volid of 0 is deleted. A new volume v2.0 with volid of 1 is added. The software version, <software_version> value is changed from 1.0 to 2.0. Three policies are added for the VNF software upgrade.

Note

- Instead of deleting and creating a new volume, you can update the volume properties. You can retain the name, vol_id, and image properties. If any of the above three properties change, then the volume will be deleted and created again.
- The volume size can be extended, and the bootable property can be changed. Other properties such as volume type, and image properties that are changed will trigger the volume to be created again.
- To update the volume id, you must remove the volume and add the volume again with a different volume id.
- The volume created by ESC cannot be updated by an out of band volume with same volume id, and vice versa.

Supported Lifecycle Stages (LCS) for VNF Software Upgrade with Volume

Each lifecycle stage has a condition and an action. Based on the condition, the action is executed. For information on policy driven data model, see Policy-Driven Data model. The following three conditions are configured for the VNF software upgrade:

Condition Name	Scope	Description
LCS::DEPLOY_UPDATE::VM_PRE_VOLUME_DETACH	Deployment	Triggered just before the ESC detaches a volume
LCS::DEPLOY_UPDATE::POST_VM_VOLUME_DETACHED	Deployment	Triggered immediately after ESC has detached a volume
LCS::DEPLOY_UPDATE::POST_VM_VOLUME_ATTACHED	Deployment	Triggered immediately after ESC has attached a new volume
LCS:DEPLOY_UPDATE:POST_VM_SOFTWARE_VERSION_UPDATED	Deployment	Triggered immediately after ESC has updated the software version of the VM

LCS::DEPLOY_UPDATE::PRE_VM_VOLUME_DETACH

This LCS condition is triggered before ESC detaches the volume. A script is run to unmount the volume before it is detached.

```
<policy>
    <name>SVU1</name>
    <conditions>
        <condition>
            <name>LCS::DEPLOY_UPDATE::PRE_VM_VOLUME_DETACH</name>
        </condition>
    </conditions>
    <actions>
        <action>
            <name>LOG</name>
            <type>pre_defined</type>
        </action>
```

```
    </actions>
</policy>
```

LCS::DEPLOY_UPDATE::POST_VM_VOLUME_ATTACHED

This LCS is triggered after the ESC has attached a new volume. A script is run to mount the volume and install new applications on the new volume.

```
<policy>
    <name>SVU2</name>
    <conditions>
        <condition>
            <name>LCS::DEPLOY_UPDATE::POST_VM_VOLUME_ATTACHED</name>
        </condition>
    </conditions>
    <actions>
        <action>
            <name>LOG</name>
            <type>pre_defined</type>
        </action>
    </actions>
</policy>
```

LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED

This LCS is triggered after the ESC has updated the software version of the VM. A Script is run to perform final configurations to complete the software upgrade.

```
<policy>
   <name>SVU3</name>
   <conditions>
      <condition>
         <name>LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED</name>
      </condition>
   </conditions>
   <actions>
      <action>
         <name>LOG</name>
         <type>pre_defined</type>
      </action>
   </actions>
</policy>
```

Note All three policies above show LOG action as the predefined action in the data model sample. If a script execution is needed, then a SCRIPT action can be added. See the Script action section below for a sample script.

Script Action

In the above examples, all the actions are pre-defined logs. You can have custom scripts instead.

```
<action>
   <name>unmount_volume</name>
   <type>SCRIPT</type>
   <properties>
      <property>
         <name>script_filename</name>
         <value>/opt/cisco/esc/esc-scripts/unmount.sh</value>
      </property>
```

```
      <property>
         <name>user_param</name>
         <value>value</value>
      </property>
   </properties>
</action>
```

All the property name and value pairs are passed to the script as space separated parameters. In the above example, the `unmount.sh` value will be be called by the scripts as follows:

```
/opt/cisco/esc/esc-scripts/unmount.sh user_param value
```

Prebuilt property names can be set to pass the ESC internal ids to the specified script. The prebulit property names are as follows:

```
  <property>
    <name>internal_deployment_id</name>
  </property>
  <property>
    <name>external_deployment_id</name>
  </property>
  <property>
    <name>deployment_name</name>
  </property>
  <property>
    <name>internal_tenant_id</name>
  </property>
  <property>
    <name>external_tenant_id</name>
  </property>
```

Here is an example of a script with the prebuilt property names and values, which ESC generates.

```
script_name.sh deployment_name my-deployment-name external_deployment_id
18fbcfd5-8b63-44e0-97ec-68de25902917
external_tenant_id my-tenant-id internal_deployment_id my-tenant-idmy-deployment-name
internal_tenant_id my-tenant-id
```

By default, ESC allows 15 minutes for the script execution to complete. Some scripts may take longer time to complete. An optional property can be specified to extend the timeout value in seconds. In the example below, the timeout of the script is set to 3600 seconds.

```
<property>
    <name>wait_max_timeout</name>
    <value>3600</value>
</property>
```

Notifications for Virtual Network Function Software Upgrade

Notifications are triggered at each stage of the VNF Software upgrade.

Volume Detached

```
status SUCCESS
        status_code 200
        status_message Detached 1 volume: [Volume=test-esc-1,volid=1]
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid eb4fe3b5-138d-41a3-b6ff-d6fa9035ca6c
```

```
        vm_group Group1
        vm_source {
            vmid cd4eeb61-61db-45a6-9da1-793be08c4de6
            hostid 8e96b8830d7bfbb337ce665586210fcca9644cbe238240e207350735
            hostname my-server-5
            software_version 1.0
            interfaces {
                interface {
                    nicid 0
                    type virtual
                    port_id 26412180-45cf-4f0b-ab45-d05bb7ca7091
                    network 943fda9e-79f8-400c-b442-3506f102721a
                    subnet e313b95c-ca1f-4c81-8d60-c9e721a85d0b
                    ip_address 192.168.0.56
                    mac_address fa:16:3e:18:90:1e
                    netmask 255.255.255.0
                    gateway 192.168.0.1
                }
            }
            volumes {
                volume {
                    display_name test-esc-1__v0_0_0_1
                    external_id 5d008a12-6fb1-492a-b648-4cf7fc8c68b1
                    bus virtio
                    type lvm
                    size 2
                }
            }
        }
        vm_target {
        }
        event {
            type VM_UPDATED
        }
    }
}
```

Volume Removed

```
notification {
    eventTime 2016-11-24T00:27:25.457+00:00
    escEvent {
        status SUCCESS
        status_code 200
        status_message Removed 1 volume: [Volume=test-esc-3,volid=1]
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid f938ca24-d0c2-42b3-a757-66b0543fe0a6
        vm_group Group1
        vm_source {
            vmid 91379ad1-1cfc-4a10-abaf-068d01ae92b9
            hostid 101f55110748903af4844a2517e854f64843b9ac8d880ad68be8af59
            hostname my-server-4
            software_version 1.0
            interfaces {
                interface {
                    nicid 0
                    type virtual
                    port_id a8201c3e-2c6e-4313-94d0-1b4eee14f08a
                    network 943fda9e-79f8-400c-b442-3506f102721a
                    subnet e313b95c-ca1f-4c81-8d60-c9e721a85d0b
                    ip_address 192.168.0.220
                    mac_address fa:16:3e:eb:bd:77
```

```
                    netmask 255.255.255.0
                    gateway 192.168.0.1
                }
            }
        }
        vm_target {
        }
        event {
            type VM_UPDATED
        }
    }
}
```

Volume Attached

```
notification {
    eventTime 2016-11-23T19:54:48.105+00:00
        status_message Attached 1 volume: [Volume=test-esc-2,volid=0]
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid eb4fe3b5-138d-41a3-b6ff-d6fa9035ca6c
        vm_group Group1
        vm_source {
            vmid cd4eeb61-61db-45a6-9da1-793be08c4de6
            hostid 8e96b8830d7bfbb337ce665586210fcca9644cbe238240e207350735
            hostname my-server-5
            software_version 1.1
            interfaces {
                interface {
                    nicid 0
                    type virtual
                    port_id 26412180-45cf-4f0b-ab45-d05bb7ca7091
                    network 943fda9e-79f8-400c-b442-3506f102721a
                    subnet e313b95c-ca1f-4c81-8d60-c9e721a85d0b
                    ip_address 192.168.0.56
                    mac_address fa:16:3e:18:90:1e
                    netmask 255.255.255.0
                    gateway 192.168.0.1
                }
            }
            volumes {
                volume {
                    display_name test-esc-2__v0_0_0_1
                    external_id bf5c9a01-e9fb-42fa-89ee-73699d6c519c
                    bus virtio
                    type lvm
                    size 2
                }
            }
        }
        vm_target {
        }
        event {
            type VM_UPDATED
        }
    }
}
```

Software Version Updated

```
notification {
    eventTime 2016-11-23T20:06:56.75+00:00
```

```
    escEvent {
        status SUCCESS
        status_code 200
        status_message VM Software Updated. VM name:
[dep_Group1_0_c9edef63-4d9d-43ea-af1b-16527ed2edae], previous version: [1.0], current
version: [1.1]
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid eb4fe3b5-138d-41a3-b6ff-d6fa9035ca6c
        vm_group Group1
        vm_source {
            vmid cd4eeb61-61db-45a6-9da1-793be08c4de6
            hostid 8e96b8830d7bfbb337ce665586210fcca9644cbe238240e207350735
            hostname my-server-5
            software_version 1.1
            interfaces {
                interface {
                    nicid 0
                    type virtual
                    port_id 26412180-45cf-4f0b-ab45-d05bb7ca7091
                    network 943fda9e-79f8-400c-b442-3506f102721a
                    subnet e313b95c-ca1f-4c81-8d60-c9e721a85d0b
                    ip_address 192.168.0.56
                    mac_address fa:16:3e:18:90:1e
                    netmask 255.255.255.0
                    gateway 192.168.0.1
                }
            }
            volumes {
                volume {
                    display_name test-esc-2__v0_0_0_1
                    external_id bf5c9a01-e9fb-42fa-89ee-73699d6c519c
                    bus virtio
                    type lvm
                    size 2
                }
            }
        }
        vm_target {
        }
        event {
            type VM_SOFTWARE_VERSION_UPDATED
        }
    }
}
```

Service Updated

```
notification {
    eventTime 2016-11-23T20:06:56.768+00:00
    escEvent {
        status SUCCESS
        status_code 200
        status_message Service group update completed successfully
        depname dep
        tenant test
        tenant_id 9132cc90b8324a1c95a6c00975af6206
        depid eb4fe3b5-138d-41a3-b6ff-d6fa9035ca6c
        vm_source {
        }
        vm_target {
        }
        event {
```

```
            type SERVICE_UPDATED
        }
    }
}
```

Upgrading VNF in a Deployment

ESC allows upgrading the VNF software in an already existing deployment in any of the following lifecycle stages.

- LCS—PRE SOFTWARE UPGRADE-SCRIPT ACTION
- LCS—POST SOFTWARE UPGRADE-SCRIPT ACTION

The NB can choose to use PRE, POST or BOTH to execute the custom action script.

For details on custom scripts, see custom scripts in Script Actions. For lifecycle stages, see Lifecycle Stage (LCS) Policy Conditions Defined at Different Stages.

The LCS_NOTIFY notification can be turned on or off for each of the lifecycle stage. For any software_version change, the final notification for each VM is VM_SOFTWARE_VERSION_UPDATED. ESC receives the SERVICE_UPDATED notification for each of the deployment update.

ESC supports the following VNF software upgrade scenarios in an existing deployment.

- VNF upgrade after deployment
- VNF deployment and application upgrade in an existing deployment

For details on updating other resources in an existing deployment, see Updating an Existing Deployment.

VNF Upgrade After Deployment

The VNF upgrade can happen in a single or staged transaction.

ESC adds the LCS policy and changes the software version in a single transaction.

In the two staged transaction, ESC adds the LCS policy in the first transaction, and triggers the software upgrade with the software version change in the second transaction.

Notifications

- LCS_NOTIFY—LCS::DEPLOY_UPDATE::PRE_VM_SOFTWARE_VERSION_UPDATE
- LCS_NOTIFY—LCS::DEPLOY_UPDATE::POST_VM_SOFTWARE_VERSION_UPDATED
- VM_SOFTWARE_VERSION_UPDATED
- SERVICE_UPDATED

Error

ESC performs an early validation on the VNF upgrade process. An error occurs if the custom script file does not exist. The transaction is rejected and no notifications are sent to the NFVO.

An error occurs if the custom script times out. The following notifications are sent to the NFVO.

- LCS::DEPLOY_UPDATE::PRE_VM_SOFTWARE_VERSION_UPDATE

- LCS::DEPLOY_UPDATE::PRE_VM_SOFTWARE_VERSION_UPDATE
- VM_SOFTWARE_VERSION_UPDATED
- SERVICE_UPDATED

VNF Deployment and Application Upgrade in an Existing Deployment

During the VNF deployment and application upgrade, ESC sends the following notifications to the NFVO.

- VM_DEPLOYED
- LCS_NOTIFY-LCS::DEPLOY::POST_VM_ALIVE
- VM_ALIVE
- SERVICE_ALIVE

Error

An error occurs when the custom script times out. The following notifications are sent to the NFVO.

- VM_DEPLOYED
- LCS::VM::POST_VM_ALIVE
- VM_DEPLOYED
- SERVICE_ALIVE