Brownfield Deployments

• Brownfield Enhacements to Support Openstack and ESC data reconciliation, on page 1

Brownfield Enhacements to Support Openstack and ESC data reconciliation

Brownfield Deployment:

Brownfield deployments are ESC VNF deployments that allow the *target* ESC VM to manage a live VNF on the VIM.

Brownfield Deployments help to migrate the live VNF management from a *source*ESC VM to a *target*ESC VM without any disruption to the actual live VNF. The brownfield deployment process uses new and existing ESC APIs to create deployment data within the ESC datastores on the target ESC VM without actually creating resources on the VIM - simply validating existing VIM resources if and when required.

Quick starting Brownfield Deployment:

If the brownfield XML files that are my-brownfield-import.xml and my-brownfield-deployment.xml exist, then create a deployment in the target ESC VM using the brownfield APIs as shown:

Create a Brownfield Data:

Create the data and fix the errors if any returned

Example Payload

```
admin@esc_vm]$ esc_nc_cli import-deployment-data CREATE my-tenant my-deployment
/tmp/my-brownfield-import.xml
Import Deployment Data
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--rpc=/tmp/tmp_esc_nc_cli.jtQHTuOubE
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <imported_data xmlns="http://www.cisco.com/esc/esc">
    <import>
      <deployment_name>dep-complete</deployment_name>
      <project_name>dave-2000</project_name>
      <vms>
        <vm_details>

<generated_name>my-deployment_vm-gro_0_a4e82d3e-a3a5-403e-b321-cc0d7b1a779e</generated_name>
"<!-- Output removed for brevity --> "
```

```
        </vm_details>
      </vms>
    </import>
  </imported_data>
</rpc-reply>
```

Deploy VNF

After deploying the VNF, wait for SERVICE_ALIVE notification. If there is an error, undeploy the VNF, fix the errors and re deploy the VNF.

Example payload:

```
[admin@esc_vm]$ esc_nc_cli edit-config /tmp/my-brownfield-deployment.xml
Configure
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--edit-config=/tmp/tmp_esc_nc_cli.53L6syLBhl
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <ok/>
</rpc-reply>
```

Finalize the deployment:

In this state, the ESC VM manages the VNF

Example payload:

```
[admin@esc_vm]$ esc_nc_cli import-deployment-data FINALIZE my-tenant my-deployment
Import Deployment Data
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--rpc=/tmp/tmp_esc_nc_cli.LY8Ai01yuz
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <imported_data xmlns="http://www.cisco.com/esc/esc">
    <import>
      <deployment_name>dep-complete</deployment_name>
      <project_name>dave-2000</project_name>
      <vms>
        <vm_details>

<generated_name>my-deployment_vm-gro_0_a4e82d3e-a3a5-403e-b321-cc0d7b1a779e</generated_name>
"<!-- Output removed for brevity --> "
        </vm_details>
      </vms>
    </import>
  </imported_data>
</rpc-reply>
```

Delete the import data:

Note The following step is optional

```
[admin@esc_vm]$ esc_nc_cli import-deployment-data DELETE my-tenant my-deployment
Import Deployment Data
/opt/cisco/esc/confd/bin/netconf-console --port=830 --host=127.0.0.1 --user=esc-nc-admin
--privKeyFile=/home/admin/.ssh/confd_id_rsa --privKeyType=rsa
--rpc=/tmp/tmp_esc_nc_cli.LY8Ai01yuz
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
```

```
  <imported_data xmlns="http://www.cisco.com/esc/esc">
    <import>
      <deployment_name>dep-complete</deployment_name>
      <project_name>dave-2000</project_name>
      <vms>
        <vm_details>

<generated_name>my-deployment_vm-gro_0_a4e82d3e-a3a5-403e-b321-cc0d7b1a779e</generated_name>
"<!-- Output removed for brevity --> "
        </vm_details>
      </vms>
    </import>
  </imported_data>
</rpc-reply>
```

Required files:

A Brownfield deployment requires at least two files to specify the VNF which is a **deployment XML** and an **import XML.** If the VNF uses VPC life cycle management (LCM) scripts that are GEN_VPC_CHASSIS_ID and GEN_VPC_SSH_KEYS during the original deployment, then further files specify the generated values. The same is discussed in the section "chassisID and user_key".

- Deployment XML:

Deployment XML details the XML data used during the original deployment. Ideally, XML is identical to the original *dep.xml* that is with the tenant's *vim_mapping* attribute set to true, but the ConfD API can be used to accurately generate the deployment XML on the source ESC VM if required.

- Import XML:

Import XML details the XML data that describes the VNF VMs and their ephemeral resources, sometimes known as *in-band* resources. The resource data exists on OpenStack, and ESC selects for the resource data during Brownfield deployment. All the in-band resources are supported including volumes, networks, subnets, and ports.

Note Both the import XML and the deployment XML specify all in-band and VIM resources required for the target deployment and therefore it is very important that both the XML are in sync with each other.

chassisID and user_key:

During a Brownfield deployment, if the original VNF deployment specifies LCM actions of type GEN_VPC_CHASSIS_ID, GEN_VPC_SSH_KEYS, or both, then the values that the LCM action executions created need to be specified in files and referenced appropriately.

Before the Brownfield deployment, the GEN_VPC_CHASSIS_ID LCM action generates a single string called a *chassisID,* and the generated value is saved to a local file on the *target* ESC VM.

The GEN_VPC_SSH_KEYS action generates two sets of SSH-related data that are SSH public and private key pairs called the *user_key*, and the generated values are saved to a local file on the *target* ESC VM.

Supported APIs:

Brownfield deployment requires two main APIs, they are, **ConfD API and ConfD or ESCManager REST APIs.**

- The ConfD API uses CREATE and FINALIZE keywords, and the most common access to the ConfD API is the esc_nc_cli script

- The ConfD or ESCManager REST API is to process a deployment XML.

ConfD: Brownfield CREATE:

The ConfD Brownfield CREATE API is used to load the import XML data that is the VNF resource data for VMs and their ephemeral data into ESC to be referenced during a VNF deployment.

The ConfD Brownfield CREATE API requires three mandatory arguments namely the tenant name, the deployment name, and a file specifying the import XML.

Example usage:

```
esc_nc_cli import-deployment-data CREATE my-tenant my-deployment /tmp/my-brownfield-import.xml
```

Upon invocation, the import XML contents are validated for structure and XML syntax and are stored in the ESC database. The actual data within the import XML is not validated for correctness as this can only be done at the deployment as shown in the following

Validation:

Upon invocation, perform the following validation steps and if there is an error, then fix and re-submit the data.

- XML is validated for syntax and to ensure mandatory values are present
- Import XML validates both the tenant and deployment names to ensure the values match
- If an index is specified, it must start at 0 for the VM group.

ConfD or ESCManager REST API: DEPLOY

Once import XML data loads and validates, the ConfD or ESCManager REST APIs are used to deploy the deployment XML data, in the same manner, that non- Brownfield deployment data is specified.

Example usage:

```
esc_nc_cli edit-config /tmp/my-brownfield-deployment.xml
```

Validation:

Upon invocation, ESC checks if the Brownfield CREATE operation previously exists for the same tenant and deployment, that is created during the ConfD Brownfield CREATE and if the deployment exists then ESC first validates all resource data to ensure the following:

- If the previously loaded import XML contains all the ephemeral resources specified in the deployment XML,
- If all resources exist on the VIM.

If either of the following fails, then the deployment itself will fail using familiar messaging.

Once the early validation passes, the deployment goes through the identical notification cycle as a non-Brownfield deployment, eventually generating a SERVICE_ALIVE notification or error notifications at the appropriate points in the workflow.

Note If there are any errors during deployment, for example, missing resources or VIM connectivity problems then the Brownfield deployment undeploys using the standard ESC undeployment APIs, once the underlying issue fixes, a re-deployment is attempted. This cycle of:

`deploy --> ERROR --> undeploy --> fix issue --> deploy` can be executed indefinitely until an error-free SERVICE_ALIVE notification is received.

ConfD: Brownfield FINALIZE

The ConfD Brownfield FINALIZE API is used to signal the *target* ESC VM which is ready to manage the VNF on the VIM as per any non-Brownfield VNF.

It requires two mandatory arguments that are the tenant name and the deployment name.

During the period where the Brownfield CREATE and DEPLOY APIs are used, but prior to this FINALIZE API called, the target ESC VM monitors the VNF, but recovery scenarios are not triggered if monitoring fails.

For example, if the VNF suddenly becomes un-pingable, then the target ESC VM does not invoke recovery policy nor it does not generate any notifications, therefore it is important that this final step occurs when the VNF on the VIM is active.

Furthermore, the target ESC VM does not implement VNF actions such as START, STOP, RECOVER, REBOOT, ENABLE/DISABLE MONITOR. If the actions are attempted, an error returns.

Validation

Brownfield deployments that have a SERVICE_ALIVE status is "finalized".

Example usage:

```
esc_nc_cli import-deployment-data FINALIZE my-tenant my-deployment
```

ConfD: Brownfield DELETE import data

The ConfD Brownfield DELETE API is used to remove all Brownfield data from the import tables.

It requires two mandatory arguments that are the tenant name and the deployment name.

Example usage

```
esc_nc_cli import-deployment-data DELETE my-tenant my-deployment
```

Validation

Only Brownfield deployments that are "finalized" can have their data deleted.

Example of import XML:

The following are some of the example import XML files or data snippets.

Basic VM plus an ephemeral volume and ports:

The following shows VNF with a single VM, one ephemeral volume, and two ephemeral ports. One port is specified using a single stack syntax, and the other port is specified using a dual-stack syntax.

Basic VM

```
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
```

```
    <vm_details>
      <name>my-vm</name>
      <uuid>f4cad63c-a1c1-48ef-a3cd-8dd20abd2118</uuid>
      <vm_group>my-vm-group</vm_group>
      <attached_volume>
        <volume_id>df49a4a5-7def-450c-9881-b886b1abbe7f</volume_id>
        <volume_name>my-inband-volume</volume_name>
      </attached_volume>

<generated_name>my-deployment_vm-gro_0_2b92d247-7c08-48b1-a9f4-ff0849a82153</generated_name>

      <port>
        <port_id>4c2aa65d-e3fa-4529-a3f5-099031ffe6c3</port_id>
        <nicid>0</nicid>
      </port>
      <port>
        <port_id>2c627b23-ce8d-482a-9ea2-21d77df611b9</port_id>
        <fixed_ips>
          <address_id>0</address_id>
          <ip_address>192.26.13.442</ip_address>
        </fixed_ips>
        <nicid>1</nicid>
      </port>
    </vm_details>
  </vms>
</import>
```

The associated deployment XML excluding the non-relevant constructs is shown as follows:

Original deployment XML

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>my-tenant</name>
      <vim_mapping>false</vim_mapping>  <!-- NOTE: Must always be "false" so that ESC does
 not try to create a new tenant -->
      <deployments>
        <deployment>
          <name>my-deployment</name>
          <vm_group>
            <name>my-vm-group</name>
            <image>Automation-Cirros-Image</image>
            <flavor>Automation-Cirros-Flavor</flavor>
            <vim_vm_name>my-vm</vim_vm_name>
            <bootup_time>180</bootup_time>
            <recovery_wait_time>180</recovery_wait_time>
            <volumes>
              <volume>
                <name>my-inband-volume</name>
                <volid>1</volid>
                <bus>virtio</bus>
                <size>2</size>
                <sizeunit>GiB</sizeunit>
                <type>LVM</type>
              </volume>
              <volume> <!-- NOTE: out of band resources do not need to be detailed in the
import XML -->
                <name>my-out-of-band-volume</name>
                <volid>0</volid>
                <bus>virtio</bus>
                <type>LVM</type>
              </volume>
            </volumes>
```

```
            <interfaces>
              <interface>
                <nicid>0</nicid>
                <network>esc-net</network>
              </interface>
              <interface>
                <nicid>1</nicid>
                <network>esc-net</network>
                <addresses>
                  <address>
                    <address_id>0</address_id>
                    <subnet>esc-subnet</subnet>
                  </address>
                </addresses>
              </interface>
              <interface> <!-- NOTE: out of band resources do not need to be detailed in
the import XML -->
                <nicid>2</nicid>
                <port>my-out-of-band-port</port>
              </interface>
            </interfaces>
 "<!-- Output removed for brevity --> "
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
  </tenants>
</esc_datamodel>
```

Basic VM plus ephemeral network:

The following shows VNF with a single VM, a single ephemeral network, and a single ephemeral port specified using dual-stack syntax.

The difference between Basic VM plus ephemeral network and Basic VM plus and ephemeral volume is that the ephemeral network needs to be detailed in the import XML and the deployment XML.

Basic VM

```
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
    <vm_details>
      <name>my-vm</name>
      <uuid>f4cad63c-a1c1-48ef-a3cd-8dd20abd2118</uuid>
      <vm_group>my-vm-group</vm_group>

<generated_name>my-deployment_vm-gro_0_2b92d247-7c08-48b1-a9f4-ff0849a82153</generated_name>

      <port>
        <port_id>2c627b23-ce8d-482a-9ea2-21d77df611b9</port_id>
        <fixed_ips>
          <address_id>0</address_id>
          <ip_address>10.120.04.198</ip_address>
        </fixed_ips>
        <nicid>0</nicid>
      </port>
    </vm_details>
  </vms>
  <network>
    <network_id>8abd5cb3-5107-4a63-bfd4-117a6d7e3824</network_id>
    <subnet>
      <subnet_id>a74bc215-172e-41f8-9f83-f578b4fb88d2</subnet_id>
```

```
      <name>my-suvnet</name>
    </subnet>
    <name>my-network</name>
  </network>
  <network>
    <network_id>xxxxx</network_id>
      </network>
</import>
```

The associated deployment XML excluding non-relevant constructs is shown as follows:

Original deployment XML

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>my-tenant</name>
      <vim_mapping>false</vim_mapping>  <!-- NOTE: Must always be "false" so that ESC does
 not try to create a new tenant -->
      <deployments>
        <deployment>
          <name>my-deployment</name>
          <networks>
            <network>
              <name>my-network</name>
              <shared>false</shared>
              <locator>
                <vim_id>default_openstack_vim</vim_id>
                <vim_project>davwebst</vim_project>
              </locator>
              <subnet>
                <name>my-subnet</name>
                <ipversion>ipv4</ipversion>
                <dhcp>true</dhcp>
                <address>192.168.1.150</address>
                <netmask>255.255.255.0</netmask>
                <gateway>192.168.1.1</gateway>
              </subnet>
            </network>
          <vm_group>
            <name>my-vm-group</name>
            <image>Automation-Cirros-Image</image>
            <flavor>Automation-Cirros-Flavor</flavor>
            <vim_vm_name>my-vm</vim_vm_name>
            <bootup_time>180</bootup_time>
            <recovery_wait_time>180</recovery_wait_time>
            <interfaces>
              <interface>
                <nicid>0</nicid>
                <network>my-network</network>
                <addresses>
                  <address>
                    <address_id>0</address_id>
                    <subnet>my-subnet</subnet>
                  </address>
                </addresses>
              </interface>
            </interfaces>
 "<!-- Output removed for brevity --> "
          </vm_group>
        </deployment>
      </deployments>
    </tenant>
```

```
  </tenants>
</esc_datamodel>
```

Scaled out deployment:

The following shows VNF with a single VM and ephemeral port, one which is scaled by the other, either through the initial deployment, or a scaling KPI trigger.

Example payload:

```
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
    <vm_details>
      <index>0</index> <!-- NOTE: index must start at zero, and is mandatory if multiple
vm details against the same vm group are specified -->
      <name>my-vm</name>
      <vm_group>my-vm-group</vm_group>

<generated_name>my-deployment_vm-gro_0_a8738b6c-1e0e-47b0-a469-db7cc63e6f92</generated_name>

      <port>
        <port_id>e974e20c-2e88-4321-beb9-5dd66e103865</port_id>
        <nicid>0</nicid>
      </port>
      <uuid>c06ab441-f895-4bd9-ab5a-9f731d42a3c1</uuid>
    </vm_details>
    <vm_details>
      <index>1</index> <!-- NOTE: index must be incremented by one if specifying vm details
 for a vm in the same vm group -->
      <name>my-vm_1</name>
      <vm_group>my-vm-group</vm_group>

<generated_name>deployment-brown_vm-gro_1_a5b5bb8b-e90e-47d4-95f0-e7481abce12e</generated_name>

      <port>
        <port_id>94290268-569d-46a2-bf73-0e81d8b18019</port_id>
        <nicid>0</nicid>
      </port>
      <uuid>317369cf-f722-4052-976b-035436fee303</uuid>
    </vm_details>
  </vms>
</import>
```

Note The scaled-out deployment is done by setting the scaling minimum and maximum values to "2" in the original deployment XML as shown:

```
<scaling>
  <min_active>2</min_active>
  <max_active>2</max_active>
</scaling>
```

ChassisID and User Key specification:

During a Brownfield deployment, if the original VNF deployment specifies LCM actions of type GEN_VPC_CHASSIS_ID, GEN_VPC_SSH_KEYS, or both, then the values that the LCM action executions created need to specify in files and referenced appropriately.

The values are specified using metadata configuration entries, with the following *type* key names supported:

```
• chassisID     - the chassis id
• user_key      - the private SSH user generated key
• user_kep.pub  - the pubic SSH user generated key
```

The following shows VNF with a single VM and ephemeral port and one which used both GEN_VPC_CHASSIS_ID and GEN_VPC_SSH_KEYS actions, therefore it requires three values that are *chassisID, user_key* and *user_key.pub* to exist within files and referenced in the import XML.

Note The actual data in the files must exist and be their unencrypted values.

Example payload:

```
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
    <vm_details>
      <name>my-vm</name>
      <uuid>5d892d0d-c127-40f7-8ecd-d7660461b96b</uuid>
      <vm_group>my-vm-group</vm_group>

<generated_name>deployment-brown_vm-gro_0_c0083a56-bcd9-46c9-93f2-3c0405915963</generated_name>

      <metadata>
        <configuration>
          <entry>
            <type>chassisID</type>
            <file>file:///tmp/vpc_data/chassisID</file>
          </entry>
          <entry>
            <type>user_key</type>
            <file>file:///tmp/vpc_data/user_key</file>
          </entry>
          <entry>
            <type>user_key.pub</type>
            <file>file:///tmp/vpc_data/user_key.pub</file>
          </entry>
        </configuration>
      </metadata>
      <port>
        <port_id>5c0293f9-27cf-4054-98ad-20fd8e7ed5fd</port_id>
        <nicid>0</nicid>
      </port>
    </vm_details>
  </vms>
</import>
```

Note The file name path is the only possible manner to specify the values, and the <file> value **must**start with "file:///".

Note If there are multiple VMs to be specified, then the block repeats for each VM. This is the case as the script actions are at a deployment level and are therefore executed per VM during a deployment, and needs specifying per VM in the import XML.

The associated deployment XML excluding non-relevant constructs is shown as follows:

Original deployment XML

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <tenants>
    <tenant>
      <name>my-tenant</name>
      <vim_mapping>false</vim_mapping>
      <deployments>
        <deployment>
          <name>my-deployment</name>
          <policies>
            <policy>
              <name>instantiate</name>
              <conditions>
                <condition>
                  <name>LCS::PRE_DEPLOY</name>
                </condition>
              </conditions>
              <actions>
                <action>
                  <name>GEN_VPC_CHASSIS_ID</name>
                  <type>SCRIPT</type>
                  <properties>
                    <property>
                      <name>CHASSIS_KEY</name>
                      <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
                    </property>
                    <property>
                      <name>script_filename</name>
                    <value>file:///opt/cisco/esc/esc-scripts/esc_vpc_chassis_id.py</value>

                    </property>
                  </properties>
                </action>
              </actions>
            </policy>
            <policy>
              <name>instantiate_start</name>
              <conditions>
                <condition>
                  <name>LCS::PRE_DEPLOY</name>
                </condition>
              </conditions>
              <actions>
                <action>
                  <name>GEN_VPC_SSH_KEYS</name>
                  <type>SCRIPT</type>
                  <properties>
                    <property>
                      <name>script_filename</name>

<value>file:///opt/cisco/esc/esc-scripts/esc-vpc-di-internal-keys.sh</value>
                    </property>
                  </properties>
```

```
              </action>
            </actions>
          </policy>
        </policies>
        <vm_group>
          <name>my-vm-group</name>
          <image>Automation-Cirros-Image</image>
          <flavor>Automation-Cirros-Flavor</flavor>
          <vim_vm_name>my-vm</vim_vm_name>
          <bootup_time>180</bootup_time>
          <recovery_wait_time>180</recovery_wait_time>
          <interfaces>
            <interface>
              <nicid>0</nicid>
              <network>esc-net</network>
            </interface>
          </interfaces>
 "<!-- Output removed for brevity --> "
        </vm_group>
      </deployment>
    </deployments>
  </tenant>
 </tenants>
</esc_datamodel>
```

Lifecycle Management Script execution:

LCM scripts are not run during a Brownfield deployment as the executed scripts create resources or perform other action on the VM which executes once. Running the LCM scripts twice during a Brownfield deployment results in resource duplication errors or leaked resources.

Within the import XML, the default *to not run* all scripts can be toggled to ensure all LCM scripts for all policies are run by default. The following shows the new parent policies element which is top level import element:

```
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
   "<!-- Output removed for brevity --> "
  </vms>
  <policies>
    <execute>true</execute> <!-- run all LCM scripts for all policies -->
  </policies>
</import>
```

Script actions are specified per vm group and or per vm group including the policy name. For example, the following snippets show changing the default script execution policy to true for all scripts, except for those in the *my-vm-group-2* vm group for the instantiate policy and script which is a part of a terminate policy.

```
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
"<!-- Output removed for brevity --> "
  </vms>
  <policies>
    <execute>true</execute> <!-- run all LCM scripts for all policies -->
    <policy>
      <execute>false</execute>
      <name>instantiate</name>
      <vm_group>my-vm-group-2</vm_group>  <!-- this is optional -->
    </policy>
    <policy>
```

```
      <execute>false</execute>
      <name>terminate</name>
    </policy>
  </policies>
</import>
```

Limitations:

When deploying in *brownfield mode*, these are the following limitations to be aware of:

- ETSI deployments are currently not supported.
- Only ESC ConfD supports the CREATE and FINALIZE actions that is any Brownfield deployment uses either the confd_cli or esc_nc_cli APIs if deploying locally or invoke the ConfD API directly if deploying remotely. The ESCManager REST API does not support the CREATE nor FINALIZE actions, but can be used to support the deployment step.
- Tenant specification has vim_mapping attribute set to false to avoid tenant creation which results in a VIM error as the tenant already exists.
- If a VNF contains multiple VMs, but not all the VM data is specified during the Brownfield import, then un deploy and re deploy the VNF with the adjusted data.
- Only Brownfield deployments on an OpenStack VIM are supported, CVIM and VMWare Brownfield deployments fails with an appropriate error.
- The Brownfield APIs allow a single deployment to be specified in a single invocation.

Generation of Brownfield data:

Brownfield data uses both the import and the deployment XML but does not have a specific API for generation as the APIs are created manually.

There are two main ways to generate the data from a source ESC VM, given the deployment and tenant name.

- **Generate Import and deployment XML through ConfD API:**

ESC VM uses the ConfD API to generate both the original deployment XML and provides the information to manually generate the import XML.

- **Deployment XML**

Execute the following command to extract and save the original deployment XML for the tenant *my-tenant* and deployment *my-deployment*:

```
[admin@esc_vm]$ echo "show running-config esc_datamodel tenants tenant my-tenant deployments
 deployment my-deployment | display xml" | sudo /opt/cisco/esc/confd/bin/confd_cli -u admin
 -C > /tmp/my-tenant.my-deployment.config.xml
```

The resultant file is the replication of the original deployment XML but *set vim_mapping to false under the tenant.*

- **Import XML**

Execute the following to extract and save all the operational data for the tenant *my-tenant* and deployment *my-deployment*:

```
[admin@esc_vm]$ echo "show esc_datamodel opdata tenants tenant my-tenant deployments
my-deployment" | sudo /opt/cisco/esc/confd/bin/confd_cli -u admin -C >
/tmp/my-tenant.my-deployment.opdata.xml
```

This command produces a structured XML document outlining the operational data for the deployment, which is used to manually create the final XML import file.

- **Generate import and deployment XML using a script:**

Brownfield import uses the script `/opt/cisco/esc/escscripts/export_brownfield_data.py` to create both the deployment XML and import XML files to be used without modification.

The script runs on the source ESC VM therefore the script needs to be copied to the source ESC VM first if it is an older ESC which does not have this script, and extract using the ConfD API all the relevant data given a tenant and deployment name.

Therefore, the deployment itself must exist within the ConfD data store, but it does not necessarily have to be in a SERVICE_ALIVE state as the data is extracted irrespective of the deployment's status.

The script requires two mandatory arguments namely the tenant name and the deployment name. ConfD access defaults to RSA key-based access, but if this is not enabled, authentication can be achieved using a name and password.

Furthermore, additional arguments exist to adjust the output of the final import XML based on metadata which cannot be gathered from the ConfD data alone and therefore requires operator actions.

The following shows the output from the Usage page:

export_brownfield_data.py – Usage

```
[admin@esc_vm] > /opt/cisco/esc/esc-scripts/export_brownfield_data.py.local -h

Usage: export_brownfield_data.py -t <tenant> -d <deployment>
                                 [-u <confd_user>] [-p <confd_password>] [-l
<local_data_directory>]
                                 [-e <[True|False]>]
[--policy:<name>:<[True|False]>:[<vm_group>]]...
                                 [-c <config_entry_path>]

Mandatory Arguments:
-t <tenant>       The deployment tenant name.
-d <deployment>   The deployment name.

Optional Arguments:
-u <confd_user>            When connecting to the ConfD API, use <confd_user>.  Defaults
to 'admin'.
-p <confd_password>        When connecting to the ConfD API, use <confd_password> for
username/password
                           access with the <confd_user>.

NOTE: If <confd_password> is **not** set, then RPC authentication is assumed to have been
enabled and is
                           used instead when accessing the ConfD API (and <confd_user> is
ignored if set).

-l <local_data_directory>  The full path to a local directory that contains two ESC data
files specifying
                          what the files generated by the ConfD API - the ConfD API is not
 used.
                           The two file names need to be in the format:
                              <tenant>.<deployment>.config.xml - configuration data from
ConfD
                            <tenant>.<deployment>.opdata.xml - operational data from ConfD


-e <[True|False]>         Execute all scripts in every policy for every VM group.  Defaults
 to 'False'
```

```
--policy [--policy <name>,<[True|False]>,[<vm_group>]],...[--policy
<name>,<[True|False]>,[<vm_group>]]

                          1 or more policies to execute all scripts for.  The policy name
 and a boolean
                          indicating if scripts should be run for that policy.
                          If the scripts are at a VM group level, then the VM group must
be specified,
                          otherwise it is optional.

                          Incorrectly specified policies will be ignored.  Examples:

                          --policy instantiate,False --policy
VM_PRE_DEPLOY,True,vm_group_xyzzy

-c <config_entry_path>    Generates a <configuration> block under <metadata> for every VM
 in the
                          import XML with three <entry> children for chassidID, user_key
and user_key.pub
                         The path/to/target is mandatory and used as the <file> value for
 each <entry>.

                          Specify the path using an absolute path.  Example:

                          -c /tmp/target/data
```

Example usage

export_brownfield_data.py - example usage

```
[admin@esc-vm]$ ./export_brownfield_data.py -t my-tenant -d my-deployment

*** Parse and validate arguments ...
`--> Tenant = my-tenant
`--> Deployment = my-deployment
`--> Execute all scripts for all policies for all VM groups = None

*** Python version 2 ***
*** Current working directory is /home/admin/BROWNFIELD ***
*** Writing temporary files to /var/tmp/tmp4fk4Sq ***

*** Generate configuration and opdata tempfiles ...
`--> config data ...
`--> Adding <vim_mapping>false</vim_mapping> to config data XML as <locator> tag found in
the body.
`--> operational data ...

*** Generate configuration data for import - i.e. dep.xml ...
`--> Writing to /home/admin/BROWNFIELD/my-tenant.my-deployment.config.xml
`--> Creating vm_group: inband port map (if any exist) ...
`--> Found 3 interfaces.
`--> Found IN BAND port for vm group vm-group-complete with nicid 0
`--> Found IN BAND port for vm group vm-group-complete with nicid 1
`--> Creating vm_group: inband volume map (if any exist) ...
`--> Looking at vm_group name vm-group-complete
`--> Found 2 volumes.
`--> Found IN BAND volume for vm group vm-group-complete with volid 1
`--> Adding ephermeral networks if they exist ...
`--> In band networks found
`--> Adding policies if they were specified ...
`--> Skipping policies as none were specified ...
`--> Writing to /home/admin/BROWNFIELD/my-tenant.my-deployment.import.xml
*** Done ***
```

```
[admin@esc-vm]$ ls -l *.xml
total 2
-rw-r--r--.  1 admin admin  4383 May 17 11:21 my-tenant.my-deployment.config.xml
-rw-r--r--.  1 admin admin  1250 May 17 11:21 my-tenant.my-deployment.import.xml

[admin@esc-vm]$ head -20 my-tenant.my-deployment.import.xml
<?xml version="1.0"?>
<import>
  <deployment_name>my-deployment</deployment_name>
  <project_name>my-tenant</project_name>
  <vms>
    <vm_details>
      <index>0</index>
      <name>my-vm</name>
      <vm_group>my-vm-group</vm_group>
      <attached_volume>
        <volume_id>828051ba-fda8-4526-910a-caf36562cc26</volume_id>
        <volume_name>my-in-band-volume</volume_name>
      </attached_volume>

<generated_name>my-deployment-vm-gro_0_a4e82d3e-a3a5-403e-b321-cc0d7b1a779e</generated_name>

      <port>
        <port_id>5e6b0e08-8639-4965-b82f-237ad6c9fe26</port_id>
        <nicid>0</nicid>
      </port>
      <port>
        <port_id>69a69157-f47f-4514-af0c-23395185b5e8</port_id>
```

The two resultant files are copied directly to the *target* ESC VM and used as inputs to the Brownfield APIs without modification.

Removal of management from *source* ESC VM:

Once a VNF migrates successfully from a source ESC VM to a target ESC VM, the VNF is removed from the source ESC VM. This is necessary because, at this point, there are two ESC VMs managing a single VNF on OpenStack, and both ESC VMs responds if the VNF became unreachable.

Following are the four techniques which can be considered:

- Depending on the circumstances, the entire source ESC VM is shutdown if it is not managing other VNFs.
- The ESC VM is wiped clean of data, if it is not managing other VNFs.
- If the ESC VM needs to stay functional, and if the VM is a standalone configuration, then the VimManager service shuts down during a maintenance window and un deployment submits through any supported ESC API for the tenant and deployment combination. This results in an internal ESC warning as the VimManager is not reachable and therefore, the VIM cannot confirm the VNFs deletion. But this is only a warning and all ESC and ConfD data is removed for the deployment.
- If stopping VimManager service is not an option in H/A or A/A environments where a maintenance window is not acceptable for such operations, then the specific VimManager connection that the deployment uses can be invalidated by changing the password or URL to incorrect values. Then an un deployment submitted through any supported ESC API for the tenant and deployment combination can be attempted. Again, this generates an internal warning but the ESC and ConfD data is removed for the deployment.