SO ETSI Alignment - Instantiate a VNF through a SVNFM.

Should you find any issues with the document, please contact/inform gareth.roper@est.tech

As this guide may become outdated temporarily, please refer to this Readme for any additional changes if you are running into issues: https://github.com/onap/so/blob/master/adapters/etsi-sol003-adapter/etsi-sol003-adapter-application/Readme.txt

Introduction

This guide will describe the steps required to execute the ETSI “Instantiate VNF” and “Terminate VNF” workflows using ONAP. The initial requirements you need for this guide are as follows:

- A stable Dublin ONAP installation.
- ESR component enabled in ONAP installation.
- VNFM-Adapter component enabled ONAP in installation.
- A VNFM that is aligned to the Sol003 Interface (https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/003/02.03.01_60/gs_NFV-SOL003v020301p.pdf)
- A Sol004 Aligned VNF package (https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/004/02.03.01_60/gs_nfv-sol004v020301p.pdf)

Initial Configurations

The following configurations need to be complete in order to execute the ETSI workflows and instantiate through the VNFM-Adapter.

Log into the MariaDB Database

Find your Mariadb pod:

```
kubectl -n onap get pods | grep maria
```

Exec into the Mariadb pod, in a default ONAP installation the pod will be named "mariadb-galera-mariadb-galera-0". (replace <PODNAME> with the Mariadb pod name):

```
kubectl -n onap -it exec <PODNAME> bash
```

Log into the SQL database:

```
mysql -uroot -psecretpassword
```

Connect to the catalogdb database:

```
connect catalogdb;
```

Now you can follow the next steps in order to configure your ONAP's Mariadb database.

Enable the ETSI “Instantiate/Create” & “Terminate/Delete” building blocks.

Firstly, you will need to add the ETSI Create & Delete building blocks, this is done by inserting them into the “building_block_detail” table in the Mariadb’s “catalogdb” database.

Insert the ETSI Create & Delete Building Blocks into the table:

```
insert into building_block_detail(building_block_name, resource_type, target_action) values
(“EtsiVnfInstantiateBB”, “VNF”, “ACTIVATE”);
```

Insert Delete Building Block:

```
insert into building_block_detail(building_block_name, resource_type, target_action) values (“EtsiVnfDeleteBB”,
“VNF”, “DEACTIVATE”);
```

View the “building_block_detail” table:

```
select * from building_block_detail;
```

You should now have entries in your “building_block_detail” table as shown in the image below.

```
| 325 | EtsiVnfInstantiateBB                    | VNF                 | ACTIVATE
| 328 | EtsiVnfDeleteBB                         | VNF                 | DEACTIVATE
+-----+-----------------------------------------+---------------------+---------------
```

Update the orchestration_flow_reference table

Note: Standard VNF instantiation is unlikely to work once this step has been completed.

The next step to take is to set which building blocks are triggered on a VNF instantiate request. We will also be setting the correct sequence for these building blocks.

View the VNF Create/Delete sequences from the “orchestration_flow_reference” table:

```
select * from orchestration_flow_reference where COMPOSITE_ACTION="VNF-Create";
```

```
select * from orchestration_flow_reference where COMPOSITE_ACTION="VNF-Delete";
```

Remove current entries for “VNF-Create” & “VNF-Delete” and add updated ones:

Retrieve “ID” from “northbound_request_ref_lookup” table. Take note of the “ID” value for “VNF-Create” and “VNF-Delete”:

```
select * from northbound_request_ref_lookup where REQUEST_SCOPE='Vnf' and IS_ALACARTE is true;
```

Remove current VNF-Insert and insert ETSI VNF-Create, replace <ID> with the corresponding value retrieved from the “northbound_request_ref_lookup” table:

```
delete from orchestration_flow_reference where COMPOSITE_ACTION = "VNF-Create";

insert into orchestration_flow_reference (COMPOSITE_ACTION,SEQ_NO,FLOW_NAME,FLOW_VERSION,NB_REQ_REF_LOOKUP_ID )
values ("VNF-Create",1,"AssignVnfBB",1,<ID>);
insert into orchestration_flow_reference (COMPOSITE_ACTION,SEQ_NO,FLOW_NAME,FLOW_VERSION,NB_REQ_REF_LOOKUP_ID )
values ("VNF-Create",2,"EtsiVnfInstantiateBB",1,<ID>);
insert into orchestration_flow_reference (COMPOSITE_ACTION,SEQ_NO,FLOW_NAME,FLOW_VERSION,NB_REQ_REF_LOOKUP_ID )
values ("VNF-Create",3,"ActivateVnfBB",1,<ID>);
```

Remove current VNF-Delete and insert ETSI VNF-Delete, replace <ID> with the corresponding value retrieved from the “northbound_request_ref_lookup” table::

```
delete from orchestration_flow_reference where COMPOSITE_ACTION = "VNF-Delete";

insert into orchestration_flow_reference (COMPOSITE_ACTION,SEQ_NO,FLOW_NAME,FLOW_VERSION,NB_REQ_REF_LOOKUP_ID )
values ("VNF-Delete",1,"EtsiVnfDeleteBB",1,<ID>);
insert into orchestration_flow_reference (COMPOSITE_ACTION,SEQ_NO,FLOW_NAME,FLOW_VERSION,NB_REQ_REF_LOOKUP_ID )
values ("VNF-Delete",2,"UnassignVnfBB",1,<ID>);
```

You have now enabled the ETSI building blocks and configured the sequence of building blocks to execute. Your “orchestration_flow_reference” should now be the same as shown below.

```
MariaDB [catalogdb]> select * from orchestration_flow_reference where COMPOSITE_ACTION="VNF-Create"
+-----+------------------+--------+----------------------+--------------+----------------------+
| id  | COMPOSITE_ACTION | SEQ_NO | FLOW_NAME            | FLOW_VERSION | NB_REQ_REF_LOOKUP_ID |
+-----+------------------+--------+----------------------+--------------+----------------------+
| 745 | VNF-Create       |      3 | ActivateVnfBB        |            1 |                  119 |
| 727 | VNF-Create       |      1 | AssignVnfBB          |            1 |                  119 |
| 730 | VNF-Create       |      2 | EtsiVnfInstantiateBB |            1 |                  119 |
+-----+------------------+--------+----------------------+--------------+----------------------+
3 rows in set (0.01 sec)

MariaDB [catalogdb]> select * from orchestration_flow_reference where COMPOSITE_ACTION="VNF-Delete"
+-----+------------------+--------+-----------------+--------------+----------------------+
| id  | COMPOSITE_ACTION | SEQ_NO | FLOW_NAME       | FLOW_VERSION | NB_REQ_REF_LOOKUP_ID |
+-----+------------------+--------+-----------------+--------------+----------------------+
| 736 | VNF-Delete       |      1 | EtsiVnfDeleteBB |            1 |                  122 |
| 739 | VNF-Delete       |      2 | UnassignVnfBB   |            1 |                  122 |
+-----+------------------+--------+-----------------+--------------+----------------------+
2 rows in set (0.00 sec)
```

Update the "orchestration_status_state_transition_directive" table

Note: It is currently unknown if this change, alone, will break VNF instantiation.

The last step that needs to be taken in the MariaDB, is to update the state transition table, in order to allow our ETSI Create building blocks to correctly change the operation status of a VNF. If the operation status is not allowed to change correctly, then our ETSI building block will be skipped and will not be executed.

View the current "orchestration_status_state_transition_directive" setup. It should look the same as the image shown below.

```
select * from orchestration_status_state_transition_directive where RESOURCE_TYPE='VNF' and
ORCHESTRATION_STATUS='Created';
```

id	RESOURCE_TYPE	ORCHESTRATION_STATUS	TARGET_ACTION	FLOW_DIRECTIVE
1323	VNF	CREATED	ACTIVATE	FAIL
1053	VNF	CREATED	ASSIGN	CONTINUE
1575	VNF	CREATED	CHANGE_MODEL	FAIL
1458	VNF	CREATED	DEACTIVATE	SILENT_SUCCESS
1188	VNF	CREATED	UNASSIGN	CONTINUE

Update the row that decides when a "VNF" with an orchestration status of "CREATED" has a target action of "ACTIVATE" to "CONTINUE" instead of "FAIL" using the following command:

```
update orchestration_status_state_transition_directive set FLOW_DIRECTIVE='CONTINUE' where RESOURCE_TYPE='VNF'
and ORCHESTRATION_STATUS='CREATED' and TARGET_ACTION='ACTIVATE' and FLOW_DIRECTIVE='FAIL';
```

The image below shows how your DB should look after running the previous command.

id	RESOURCE_TYPE	ORCHESTRATION_STATUS	TARGET_ACTION	FLOW_DIRECTIVE
1323	VNF	CREATED	ACTIVATE	CONTINUE
1053	VNF	CREATED	ASSIGN	CONTINUE
1575	VNF	CREATED	CHANGE_MODEL	FAIL
1458	VNF	CREATED	DEACTIVATE	SILENT_SUCCESS
1188	VNF	CREATED	UNASSIGN	CONTINUE

The transition directive is now set up correctly, allowing all of your ETSI building blocks to be executed correctly.

Adding your VNFM to ONAP ESR

Now you will need to send a curl command to A&AI, in order to add the VNFM to ESR/A&AI.

Please ensure you have ESR added to your ONAP installation before attempting this step. Here is a link to setting up ESR: ESR setup. Next, you will need to populate the ESR VNFM List with information relating to the VNFM that you want to instantiate your VNFs through.

Adding your VNFM to ONAP ESR using CURL:

In order to use the curl command method, you will need to log into one of your ONAP pods and then send the command. (This prevents us needing to go and get the AAI service IP and external port.)

In order to log into one of your pods, first find any pod, this example with use the BPMN-INFRA pod, with this command:

```
kubectl -n onap get pods | grep bpmn
```

Then take the full pod name and put it into this command instead of <PODNAME>:

```
kubectl -n onap exec -it <PODNAME> sh
```

Once inside the pod you can run the following command which creates a VNFM, in ESR, with ID “ExampleVnfm”. Edit this curl command to your needs before using it.

```
curl -X PUT -H 'Accept: application/json' -H 'Authorization: Basic YWFpQGFhaS5vbmFwLm9yZzpkZW1vMTIzNDU2IQ==' -H
'Content-Type: application/json' -H 'X-FromAppId:12' -H 'X-TransactionId: 12' https://aai.onap:8443/aai/v15
/external-system/esr-vnfm-list/esr-vnfm/ExampleVnfm -d '{"vnfmId": "ExampleVnfm", "name": "ExampleVnfmName",
"type": "ExampleVnfmType", "vendor": "est"}'
```

One important thing to note in this curl command is the type: "ExampleVnfmType". This will be used in a later step for specifying which VNFM you want to instantiate through, take note of this.

Once you have entered the previous information you need to add the “service-url” to your “esr-system-info” section of this VNFM you just added. Please note, that the “service-url” in the following curl command was designed to work with the “so-vnfm-simulator”, you will need to change this to match your specific VNFM’s “service-url”.

You will need to put this data into the "external-system" and "aai" API paths listed below. This is done with the following curl commands:

AAI

```
curl -X PUT -H 'Accept: application/json' -H 'Authorization: Basic YWFpQGFhaS5vbmFwLm9yZzpkZW1vMTIzNDU2IQ==' -H
'Content-Type: application/json' -H 'X-FromAppId:12' -H 'X-Transaction
Id: 12' https://aai.onap:8443/aai/v15/cloud-infrastructure/cloud-regions/cloud-region/<CLOUD_OWNER>
/<CLOUD_REGION_ID>/esr-system-info-list/esr-system-info/ExampleVnfm -d '{"name": "ExampleVnfm", "system-t
ype": "ExampleVnfmType", "vimId": "myCloud", "vendor": "EST", "version": "V1.0", "certificateUrl": "", "url":
"http://so-vnfm-simulator.onap:9095/vnflcm/v1/", "user-name": "testUser", "password":
 ""}'
```

Please note you will need to replace <CLOUD_OWNER> and <CLOUD_REGION_ID> with their respective values in your ONAP deployment.

External-System

```
curl -X PUT -H 'Accept: application/json' -H 'Authorization: Basic YWFpQGFhaS5vbmFwLm9yZzpkZW1vMTIzNDU2IQ==' -H
'Content-Type: application/json' -H 'X-FromAppId:12' -H 'X-TransactionId: 12' https://aai.onap:8443/aai/v15
/external-system/esr-vnfm-list/esr-vnfm/ExampleVnfm/esr-system-info-list/esr-system-info/ExampleEsrSystemInfo -
d '{"esr-system-info-id": "ExampleEsrSystemInfo", "type": "ExampleVnfmType", "user-name": "user", "password":
"password", "system-type": "VNFM", "service-url": "http://so-vnfm-simulator.onap:9095/vnflcm/v1"}'
```

You have now entered your VNFM into the ESR/AAI components.

Here are the equivalent GET commands for checking what is currently in your ESR/AAI list (change the IDs to match the IDs you used earlier):

```
curl -H 'Accept: application/json' -H 'Authorization: Basic YWFpQGFhaS5vbmFwLm9yZzpkZW1vMTIzNDU2IQ==' -H
'Content-Type: application/json' -H 'X-FromAppId:12' -H 'X-TransactionId: 12' https://aai.onap:8443/aai/v15
/external-system/esr-vnfm-list/


curl -H 'Accept: application/json' -H 'Authorization: Basic YWFpQGFhaS5vbmFwLm9yZzpkZW1vMTIzNDU2IQ==' -H
'Content-Type: application/json' -H 'X-FromAppId:12' -H 'X-TransactionId: 12' https://aai.onap:8443/aai/v15
/external-system/esr-vnfm-list/esr-vnfm/ExampleVnfmId/esr-system-info-list/esr-system-info
```

Change the SDC Endpoint from HTTPS to HTTP

Next we will need to enable our VNFM-Adapter to communicate with SDC, this is done by changing the communication from using HTTPS to use HTTP. This will be done by editing the “override.yaml” in the VNFM-Adapter’s Helm charts.

The file that needs to be edited is the “override.yaml” found, by default, in the directory:

/root/oom/kubernetes/so/charts/so-vnfm-adapter/resources/config/overrides

Within this file you will need to find the following endpoint attribute:

endpoint: **https**://sdc-be.{{ include "common.namespace" . }}:**8443**

This endpoint, needs to be changed to the following:

endpoint: **http**://sdc-be.{{ include "common.namespace" . }}:**8080**

You will then need to update and redeploy your Helm charts in order to pick up the change. I recommend completing the next section below before doing a helm deploy, as the next section requires editing of the same file. Once this is done, your VNFM-Adapter pod should terminate and recreate a pod which has the endpoint changes.

Your “override.yaml” for the VNFM-Adapter should be the same as the following image:

```
aai:
  auth: 2A11B07DB6214A839394AA1EC5844695F5114FC407FF5422625FB00175A3DCB8A1FF745F22867EFA72D5369D5
  version: v15
  endpoint: https://aai.{{ include "common.namespace" . }}:8443
spring:
  security:
    usercredentials:
    - username: vnfm
      password: '$2a$10$Fh9ffgPw2vnmsghsRD3ZauBL1aKXebigbq3BB1RPWtE62UDILsjke'
      role: BPEL-Client
    - username: mso_admin
      password: '$2a$10$Fh9ffgPw2vnmsghsRD3ZauBL1aKXebigbq3BB1RPWtE62UDILsjke'
      role: ACTUATOR
server:
    port: {{ index .Values.containerPort }}
mso:
  key: 07a7159d3bf51a0e53be7a8f89699be7
  site-name: localSite
  logPath: ./logs/vnfm-adapter
  msb-ip: msb-iag.{{ include "common.namespace" . }}
  msb-port: 80
sdc:
  username: mso
  password: 76966BDD3C7414A03F7037264FF2E6C8EEC6C28F2B67F2840A1ED857C0260FEE731D73F47F828E5527125
  key: 566B754875657232314F5548556D3665
  endpoint: http://sdc-be.{{ include "common.namespace" . }}:8080
vnfmadapter:
  endpoint: http://so-vnfm-adapter.{{ include "common.namespace" . }}:9092
```

Configure security settings for VNFM adapter

Communication between the SO BPMN flow, the VNFM adapter and the VNFM/simulator will not succeed unless the security settings have been set correctly. Please see README.txt in the SO gerrit repo

Zip Package VNFD Path

The last step related to the Helm charts of the VNFM-Adapter, is to add a custom variable to the “override.yaml” of the VNFM-Adapter that points the VNFM-Adapter to the VNFD within your services’ package. You will need to add the following entry to the “sdc” section of the “override.yaml” (found here: /root/oom/kubernetes/so/charts/so-vnfm-adapter/resources/config/overrides):

toscametapath: Artifacts/Deployment/OTHER/TOSCA.meta

This entry needs to be entered in the sdc section of the “override.yaml”, as shown in the image below:

```
aai:
  auth: 2A11B07DB6214A839394AA1EC5844695F5114FC407FF5422625FB00175A3DCB8A1FF745F2
  version: v15
  endpoint: https://aai.{{ include "common.namespace" . }}:8443
spring:
  security:
    usercredentials:
    - username: vnfm
      password: '$2a$10$Fh9ffgPw2vnmsghsRD3ZauBL1aKXebigbq3BB1RPWtE62UDILsjke'
      role: BPEL-Client
    - username: mso_admin
      password: '$2a$10$Fh9ffgPw2vnmsghsRD3ZauBL1aKXebigbq3BB1RPWtE62UDILsjke'
      role: ACTUATOR
server:
    port: {{ index .Values.containerPort }}
mso:
  key: 07a7159d3bf51a0e53be7a8f89699be7
  site-name: localSite
  logPath: ./logs/vnfm-adapter
  msb-ip: msb-iag.{{ include "common.namespace" . }}
  msb-port: 80
sdc:
  username: mso
  password: 76966BDD3C7414A03F7037264FF2E6C8EEC6C28F2B67F2840A1ED857C0260FEE731D7
  key: 566B754875657232314F5548556D3665
  endpoint: http://sdc-be.{{ include "common.namespace" . }}:8080
  toscametapath: Artifacts/Deployment/OTHER/TOSCA.meta
vnfmadapter:
  endpoint: http://so-vnfm-adapter.{{ include "common.namespace" . }}:9092
```

Upload VNF Image to VNFM

Currently, there is no implementation of the package management interface in the VNFM-Adapter, this means that the VNF image needs to be onboarded to your VNFM before instantiation. The VNF image will then be selected by using the VNF descriptor, found in one of the artifacts within the SDC onboarding package, mentioned later in this guide ("descriptor.yaml").

This is an important step, which unfortunately can be drastically different depending on the specific vendor's VNFM.

Onboarding a Virtual Software Product (VSP) with an ETSI HEAT Template.

Login to the ONAP Portal as the designer.

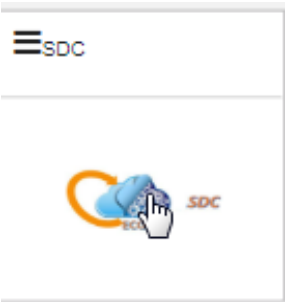
https://portal.api.simpledemo.onap.org:30225/ONAPPORTAL/login.htm

Designer Login: **cs0008**

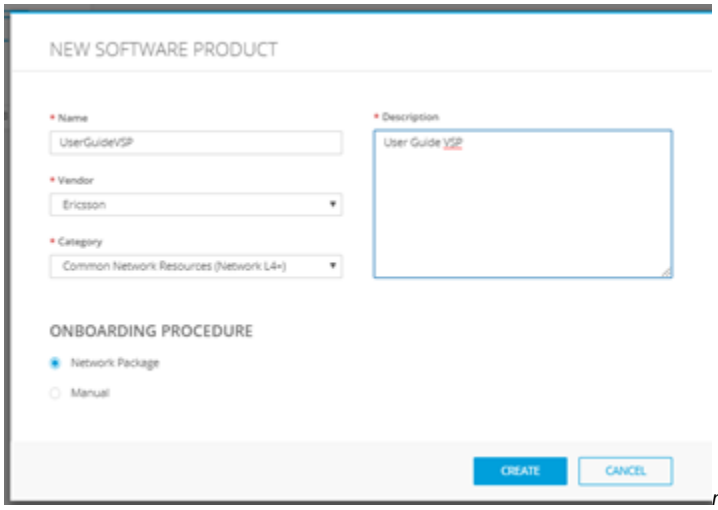
Password: **demo123456!**

Select the SDC App. It is possible that your browser will block the scripts run by the Portal, you will need to enable them in order to proceed.

Close the splashscreen.



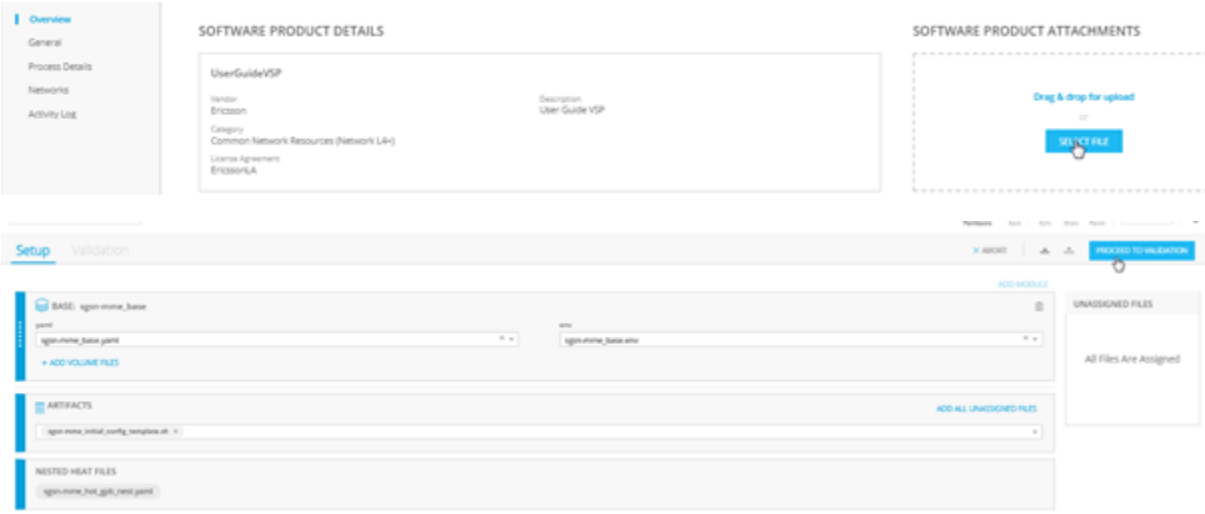
1. Click the “ONBOARD” tab near the top of the window.
2. Create *“VLM”*if you have not created a "VLM" before.
    a. From the “ONBOARD” page, click create VLM.
    b. Enter name and description, then select create.
    c. Click on the plus button beside *Entitlement Pool*.
    d. Add Name and Manufacturing Reference Number, then click save.
    e. Click on the plus button beside *License Key Groups*.
    f. Add Name and select type, then click save.
    g. Click on the plus button beside *Feature Groups*.
    h. In the general tab, add name, Description and Part Number. Move to the Entitlements Pools tab, select the entitlement pool you just created and click the rightward arrow. Move to the License Key Group tab, select the license key group you just created and click the rightward arrow. Once these three things are done, click save.
    i. Click on the plus button beside *License Agreement*.
    j. Add Name and select License Term. Next move to the Feature Groups tab, select the Feature Group you just created and click the rightward arrow. Lastly click save.
    k. On the overview page, select submit in the top right corner of the screen.
    l. Enter a commit comment and click Commit & Submit.
3. Click the “ONBOARD” tab near the top of the window.
4. Click “CREATE NEW VSP” and fill in the required information. Make sure to select “Network Package” for the “ONBOARDING PROCEDURE” section. Then click “CREATE”.



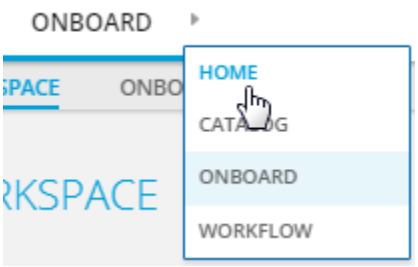
ng a Virtual Software Product (VS

5. Now you will need to click where it says “! Missing” underneath “License Agreement”. Simply select a “Licensing Version” and “License Agreement /Feature Group” from the drop downs.

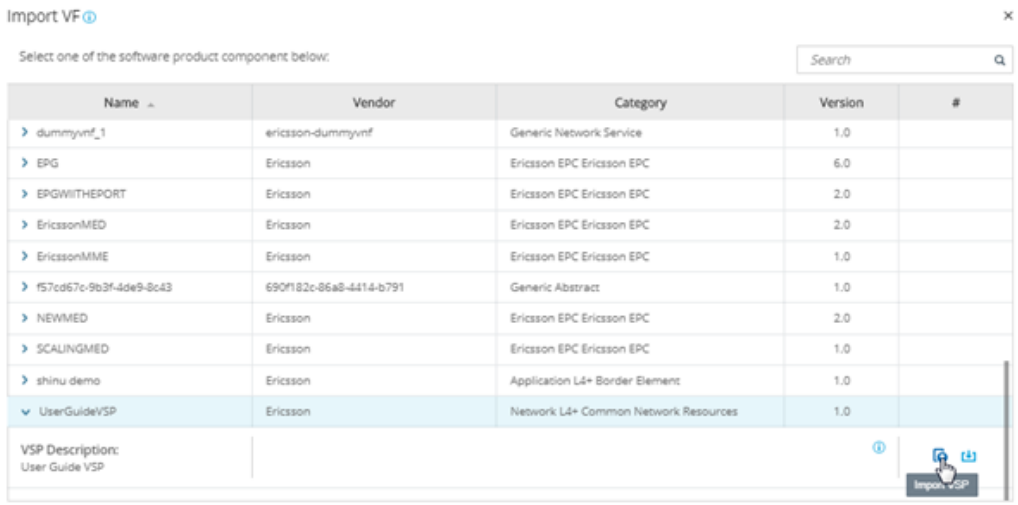
6. Next click the “Overview” tab on the left hand side. Then press “Select File” in the “SOFTWARE PRODUCT ATTACHMENTS” section and select your prepared zip (see Example Zip Section below). If you are onboarding a supported zip, then click “PROCEED TO VALIDATION” once you can see your added files. You can safely ignore any warnings that come up at this step, but not any errors. *(Note: The package validation does not support csars currently. This is due to change in the future however.)* ***For this guide, you will need to onboard a package matching the example package’s format/structure, found at the bottom of this guide. Currently we use a zip folder with an artifact that contains the VNFD.***



6. Now click “Submit”. Then click “ONBOARD” in the top left. Hover your mouse over the small grey triangle that is just to the right of the “ONBOARD” tab and select “HOME”.

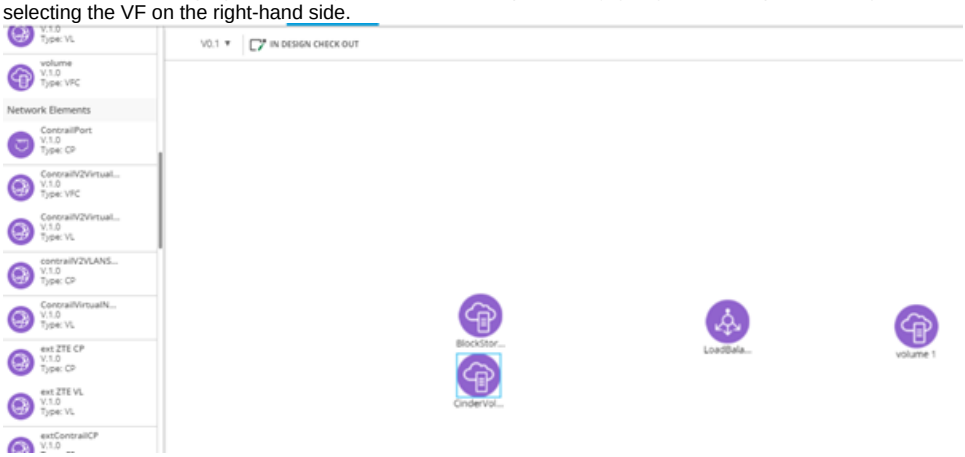


7. Hover over the “IMPORT” square and select “Import VSP”. Find your VSP click the drop-down arrow beside it and then press the “Import VSP” icon at the far right of the line that drops down. As shown here:



You will now be brought to the draft page of your VF. Follow the instructions below in order to certify it.

1. First, give your VF a name you will remember, as we will be using this in the following section, then click "Create" in the top-right hand corner.
2. Now you can go to the “Composition” tab on the left-hand side. You can drag/drop different components into the central window, depending on your VF’s needs, but you should leave it as default for now. When importing your VSP package it should configure the VF correctly.
3. You can also edit the properties of any VFs and their components by going to the “Properties Assignment” tab on the left-hand side, then selecting the VF on the right-hand side.

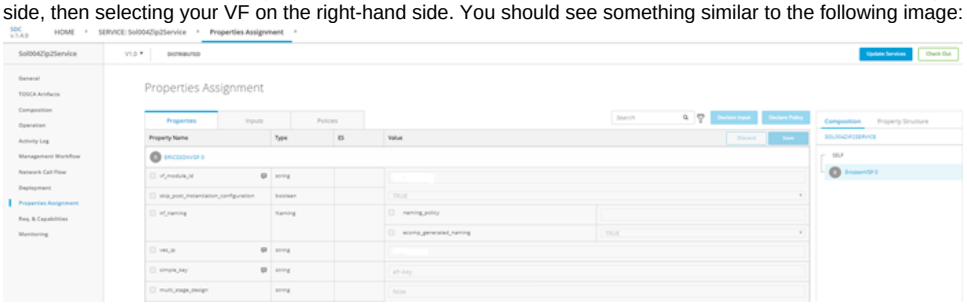


4. Once you have configured your VF, click “Certify” in the top right-hand corner.
5. Your VF is certified and has been added to the SDC Catalog. It can be added into new Services now.

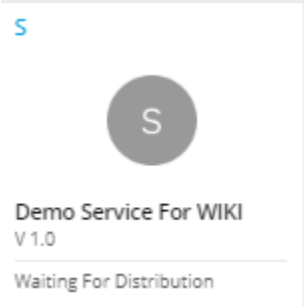
<u>Creating a Service and Approving/Distributing it</u>

1. Logged in as “**cs0008**”, in the “HOME” tab of the SDC ONAP Portal, hover over the “ADD” square and select “ADD SERVICE”. Fill in the required fields and press “Create” in the top right-hand corner.
2. Now you will be brought to the draft page of your Service. *(Note: You can also select the instantiation type at the bottom right of the page. In most cases you should use the Alacarte instantiation type.)*
3. Now go to the “Composition” tab on the left-hand side and drag/drop the VF, that you just created, into this service (you can search for the VF by name in the top left).

4. Next you will need to edit the properties of the VF that you just added to your Service by going to the “Properties Assignment” tab on the left-hand side, then selecting your VF on the right-hand side. You should see something similar to the following image:



5. Here you will need to change the property “nf_role” to match the **type** that you gave to your VNFM when entering it into the ESR VNFM list, the example earlier used “ExampleVnfmType” as the type. Make sure to press the blue “Save” button above the properties after making your change. This update ensures that the correct VNFM is selected when sending the instantiate requests through the VNFM-Adapter.
6. Once you have configured your Service, click “Submit for Testing” in the top right-hand corner.
7. Now you will need to logout and sign into the Portal as the Tester account (**jm0007**). Then go into SDC. You should see your Service in the “HOME” tab of SDC.
8. Click on your Service then click “Start Testing” in the top right-hand corner. Then click “Accept” and fill in the required field.
9. Now your complete Service has been certified and added to the SDC Catalog.
10. The next step is approving the Service for distribution. You will need to sign into the Governor account in the ONAP Portal (**gv0001**) and then click into SDC.
11. Find your Service in the “HOME” tab of SDC, it should say “Waiting for Distribution” beneath it, click into it.

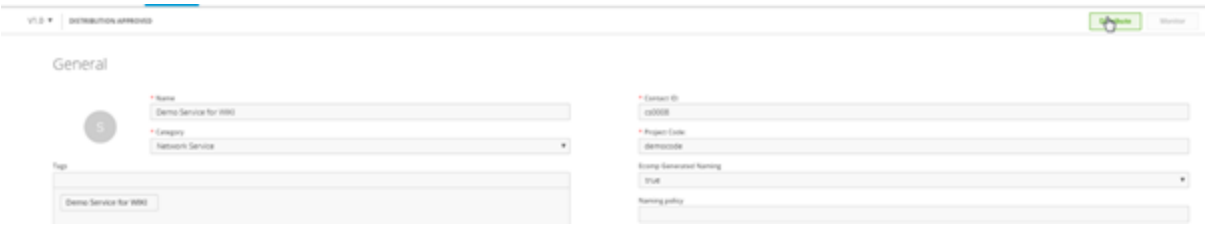


12. Now click “Approve” in the top right-hand corner and fill in the required field.

13. The final step in SDC is to distribute the Service, as it has been approved for distribution.

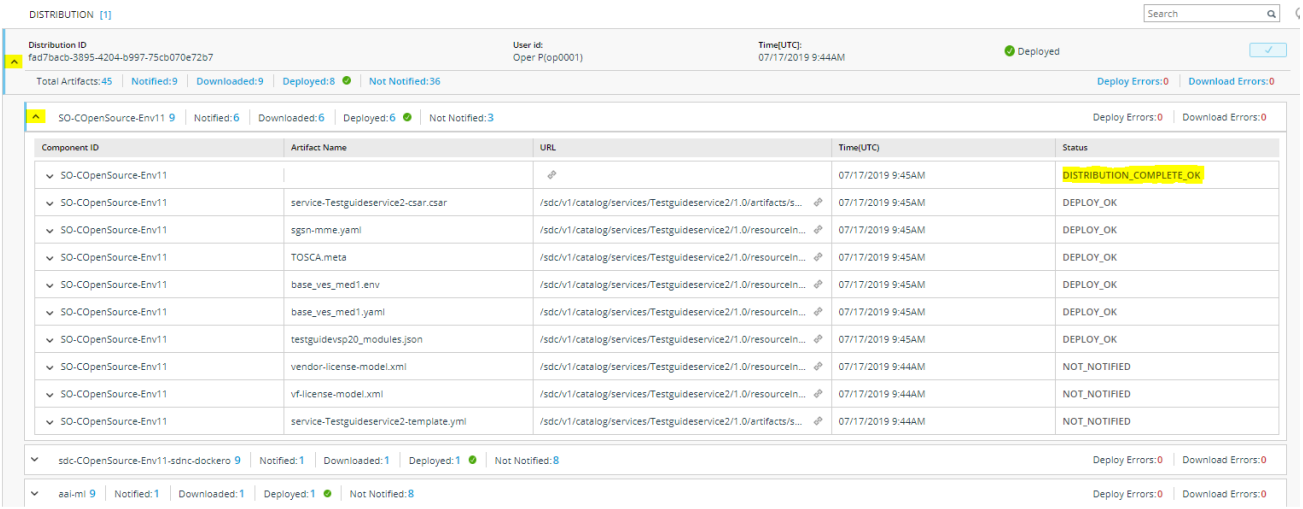
14. Sign into the ONAP Portal as the Operator (**op0001**) and find your Service in the SDC “HOME” tab, then click into it.

15. Now click the “Distribute” button in the top right-hand corner.*(After distributing, you can click the “Monitor” button in the top right-hand corner to see the status of the distribution.)*



16. Your Service, and it’s contained VFs, are now distributed and will be sent to the required components. (i.e A&AI, SDNC etc.)

17. Click "Monitor" in the top-right hand corner. Then ensure there are no errors received. In order to see the distribution status, click the arrows on the left, highlighted in the picture below. Then you should see "DISTRIBUTION_STATUS_OK" once the service has been distributed successfully.



Preloading SDNC

The next step is to preload SDNC with the required attributes for your VNF. Here is the link to the SDNC OpenDaylight RestConf API Documentation: http://portal.api.simpledemo.onap.org:30202/apidoc/explorer/index.html

You will be required to sign in once you access this site, the credentials are as follows:

Username: **admin**

Password: **Kp8bJ4SXszM0WXlhak3eHlcse2gAw84vaoGGmJvUy2U**

Next click on **VNF-API**.

Then use the following endpoint to post the preload JSON, found below.

Endpoint: **restconf/operations/VNF-API:vnf-topology-operation**

The following section of code is an example of the JSON that needs to be uploaded to the SDNC OpenDaylight RestConf API Documentation site.

Please **note** that you will need to set the attributes "generic-vnf-name" and "vnf-name" to the exact name that you will use when instantiating the VNF through VID. The attributes "generic-vnf-type" and "vnf-type" need to have the exact same name as the VSP that you imported to SDC, to create the VF.

Preload for SDNC

```
{
        "input": {
                "request-information": {
                        "notification-url": "openecomp.org",
                        "order-number": "1",
                        "order-version": "1",
                        "request-action": "PreloadVNFRequest",
                        "request-id": "robot21"
                },
                "sdnc-request-header": {
                        "svc-action": "reserve",
                        "svc-notification-url": "http://openecomp.org:8080/adapters/rest/SDNCNotify",
                        "svc-request-id": "robot21"
                },
                "vnf-topology-information": {
                        "vnf-assignments": {
                                "availability-zones": [],
                                "vnf-networks": [],
                                "vnf-vms": []
                        },
                        "vnf-parameters": [{
                                "vnf-parameter-name": "additionalParams",
                                "vnf-parameter-value": "{\"key_1\": \"value_1\"}"
                        }, {
                                "vnf-parameter-name": "extVirtualLinks",
                                "vnf-parameter-value": "{}"

                        }],
                        "vnf-topology-identifier": {
                                "generic-vnf-name": "VnfInstantiateName",
                                "generic-vnf-type": "VspPackageName",
                                "service-type": "vCPE",
                                "vnf-name": "VnfInstantiateName",
                                "vnf-type": "VspPackageName"
                        }
                }
        }
}
```

Currently the ETSI Aligned Workflows support using only the parameters "additionalParams" and "extVirtualLinks", as shown above. The datatype of these can be found in the Sol003 Specifications.

The data must be JSON and contain only escaped strings. Here are examples of both:

addtionalParameters:

Example of additionalParameters parameter.

```
{\"enableRollback\": \"false\"}
```

extVirtualLinks:

Example of extVirtualLinks Parameter

```
[{\"id\":\"3b94d0be-6e37-4a01-920f-512e96803fc9\",\"tenant\":{\"cloudOwner\":\"CloudOwner\",\"regionName\":\"
RegionOne\",\"tenantId\":\"f3d66580-7eff-4da5-8d27-91f984ad0c0b\"},\"resourceId\":\"e6e1a04d-c599-4b09-bc16-
688834d0ac50\",\"extCps\":[{\"cpdId\":\"a83f86e0-7e9b-4514-9198-2d9eba91bd8e\",\"cpConfig\":[{\"cpInstanceId\":
\"f966673d-fb96-41d4-8e5c-659f1c8c6bcc\",\"linkPortId\":null,\"cpProtocolData\":null}]}],\"extLinkPorts\":null}]
```

Using VID to send Instantiate Request

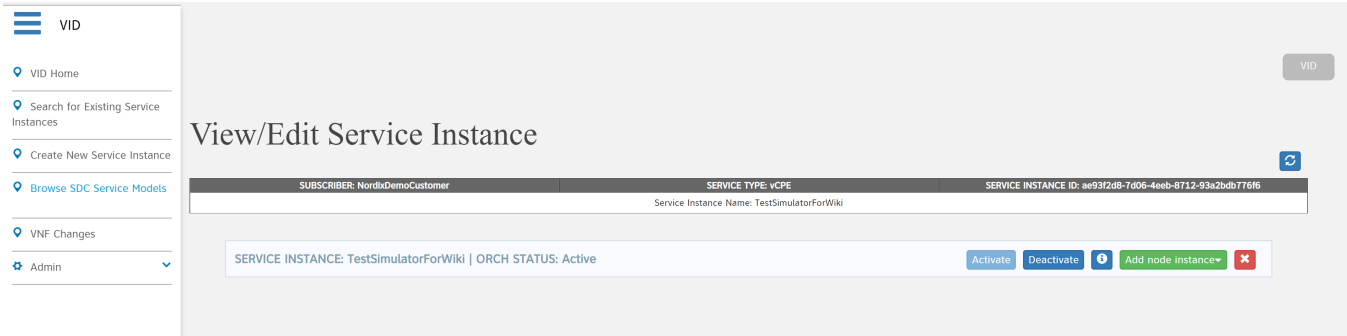
Before executing this section you will need to preload SDNC, as per previous step, or the VNF instantiation will fail.

In order to access the VID (Virtual Infrastructure Deployment) component through the portal, you will need to login with the id “demo”. Once logged in to VID, first ensure that the GR-API is set. First you will need to instantiate the service, once this is done you can then instantiate the VNF. This will be when the ETSI Workflows are run.

Then you will need to select “Deploy an SDC Service” on the left-hand side of the GUI. You should see your distributed service in the list of services shown here. (Note: if you cannot see your services here then you will need to go back to SDC, logged in as op0001, to check the status of the distribution.)

1. Press "Deploy" on the left-hand side of the service you have distributed.
2. Fill out the required fields and press "Confirm".
3. Wait for the Service to be instantiated.
4. Press "Close" at bottom of pop-up window.

Now you should be brought to the "View/Edit Service Instance" page, focused on the service you just instantiated, as shown below:



Instantiate VNF:

1. Press "Add node instance" and select your VNF.
2. Fill out the required fields and press "Confirm".
3. Leave this VID page open, as this step can take quite some time, depending on a number of factors.
4. Monitor the VNF instantiation through your VNFM GUI and back through the VNFM-Adapter logs and finally the BPMN logs.
5. Upon success, your VNF should be instantiated correctly.

Deleting VNF:

1. If you wish to delete the VNF you will need to simply travel back to the service instance that you instantiated your VNF through.
2. Select the red X on the right-hand side of the VNF instance, as shown in the image below:

3. The VNF should begin terminating now, it may take quite some time, depending on a number of factors.

4. Monitor the VNFM GUI and other logs until success.

<u>Monitoring Logs (BPMN, VNFM-Adapter and VNFM)</u>

There are 3 stages of logs to monitor throughout the process of instantiating your service, and sending your request through the VNFM-Adapter, to your VNFM.

The initial service instantiation request will be recorded in the BPMN-INFRA pod’s logs, so if you log into this pod you will be able to view them.

The VNF instantiation request will appear first in the BPMN-INFRA pod’s logs, then once the ETSI Building Block is being executed you will see entries going through the VNFM-Adapter pod’s logs. Followed finally by the VNFM itself receiving a request from the VNFM-Adapter. This should all be recorded throughout the “debug.logs” on each of the mentioned pods.

The other areas to monitor would be your VNFM’s GUI (if applicable), your Openstack Tenant’s logs as well as it’s server list and the SO-Monitoring tool (in order to see the BPMN flow’s progress).

<u>Example Zip VNF Package</u>

Please follow the structure laid out below for creating your onboarding package.

<u>Structure</u>:

- 5 files (2 .yaml, 1 .meta, 1 .json, 1 .env)

        - base.yaml
        - descriptor.yaml
        - base.env
        - MANIFEST.json
        - TOSCA.meta
    - Compressed in a **Zip** folder.
    - No directories. (Flat structure)

Files:

base.yaml - This file will be a very simple HEAT template, as it is just required in order to be able to instantiate the Service once its distributed.

descriptor.yaml - This file will contain the VNFD (Virtual Network Function Descriptor). It must be structured to match what the VNFM-Adapter searches for.

base.env - This file simply contains some environment variables for the base.yaml.

MANIFEST.json - This file lists all of the other files contained within it's package.

TOSCA.meta - This important file contains the path of the VNFD, which will be used by the VNFM-Adapter.

Please find example versions of the files below:

base.yaml

base.yaml

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

parameters:
  simple_name_0:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
  simple_key:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
  simple_image_name:
    type: string
    label: Image ID
    description: Image to be used for compute instance
  simple_flavor_name:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used
  vnf_id:
    type: string
    label: VNF ID
    description: The VNF ID is provided by ONAP
  vf_module_id:
    type: string
    label: vFirewall module ID
    description: The vFirewall Module ID is provided by ONAP
  simple_netid:
    type: string
    label: Netid
    description: netid
  public_net_id:
    type: string
    label: Netid
    description: public NetId
  ves_ip:
    type: string
    label: Netid
    description: public ves_ip
  node_ip:
    type: string
    label: Netid
```

```
    description: public ves_ip



resources:

  simple_0_private_port:
    type: OS::Neutron::Port
    properties:
      network: { get_param: simple_netid }
      fixed_ips:
      - ip_address: { get_param: node_ip }


  simple_0:
    type: OS::Nova::Server
    properties:
      availability_zone: nova
      key_name: { get_param: simple_key }
      image: { get_param: simple_image_name }
      flavor: { get_param: simple_flavor_name }
      name: { get_param: simple_name_0 }
      metadata: {vnf_id: { get_param: vnf_id }, vf_module_id: { get_param: vf_module_id }}
      networks:
      - port: { get_resource: simple_0_private_port }
      user_data_format: RAW
      user_data:
        str_replace:
          params:
            __ves_ip__: { get_param: ves_ip }
            __vnfId__: { get_param: vnf_id }

          template: |
            #!/bin/bash

            echo "the value we got for vndID was : __vnfId__" >> /tmp/vnfid.log

outputs:
  oam_management_v4_address:
    description: The IP address of the oam_management_v4_address
    value: { get_param: node_ip  }
```

descriptor.yaml

descriptor.yaml

```
tosca_definitions_version: tosca_simple_yaml_1_1

imports:
    - etsi_nfv_sol001_vnfd_0_10_0_type.yaml

node_types:
    Wiki.Demo.VnfmImageId:
        derived_from: tosca.nodes.nfv.VNF
        properties:
            descriptor_id:
                type: string
                constraints: [ valid_values: [ VnfmImageId ] ]
                default: VnfmImageId
```

The "descriptor.yaml" is the most important file within the package, as it provides the ID/Name of the VNF package for the VNFM to use when instantiating. It must follow the structure above, or the VNFM-Adapter will not be able to locate the VNFD.

Don't forget to replace "VnfmImageId" with the ID of your VNF package.

base.env

base.env

```
parameters:
  simple_image_name: UBUNTU16
  simple_flavor_name: m1.small
  simple_name_0: SIMPLEUBU
  simple_key: demo-key
  vnf_id: VESMED
  vf_module_id: vfModuleId
  simple_netid:  onap_vip
  public_net_id: nova_floating
  ves_ip: 172.55.10.10
  node_ip: 172.55.10.10
```

MANIFEST.json

MANIFEST.json

```
{
  "name": "MMEPackage",
  "description": "Test",
  "version": "0.0",
  "data": [
    {
      "isBase": true,
      "file": "base.yaml",
      "type": "HEAT",
      "data": [
        {
          "file": "base.env",
          "type": "HEAT_ENV"
        }
      ]
    },
    {
      "file": "descriptor.yaml",
      "type": "OTHER"
    },
    {
      "file": "TOSCA.meta",
      "type": "OTHER"
    }
  ]
}
```

TOSCA.meta

TOSCA.meta

```
TOSCA-Meta-File-Version: 1.0
CSAR-Version: 1.1
Created-by: Demo
Entry-Definitions: Artifacts/Deployment/OTHER/descriptor.yaml
```

The MANIFEST.json and TOSCA.meta are extremely important, if either are incorrectly formatted it will either fail to onboard or fail to distribute when you get to that step.

Ensure that the file names all match and your indentation/quotes are all correct, as it could save you a lot of time.