vFWCL instantiation, testing, and debugging

Pre-requisite

Setup the OOM Infrastructure; I've used OOM on Rancher in OpenStack

Running vFW demo - Close-loop

Video of onboarding

I had a hickup at the end, due to the fact I already had another vFW deployed, hence the ip it tried to assign was used. To fix this, I remove the existing stack.

Video of instantiation

I had a hickup for the vFW_PG due to the fact I pre-loaded on the wrong instance. After realizing, all went well.

1. Let's start by running the init goal (assuming K8S namespace, where ONAP has been installed is "onap")

```
cd oom/kubernetes/robot
$ ./demo-k8s.sh onap init
```
Starting Xvfb on display :89 with res 1280x1024x24
Executing robot tests at log level TRACE
==============================================================================
OpenECOMP ETE
==============================================================================
OpenECOMP ETE.Robot
==============================================================================
OpenECOMP ETE.Robot.Testsuites
==============================================================================
OpenECOMP ETE.Robot.Testsuites.Demo :: Executes the VNF Orchestration Test ...
Initialize Customer And Models | PASS |
------------------------------------------------------------------------------
OpenECOMP ETE.Robot.Testsuites.Demo :: Executes the VNF Orchestration Test ... | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
OpenECOMP ETE.Robot.Testsuites | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
OpenECOMP ETE.Robot | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
OpenECOMP ETE | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output: /share/logs/demo/InitDemo/output.xml
Log: /share/logs/demo/InitDemo/log.html
Report: /share/logs/demo/InitDemo/report.html

2. Login into the VNC. Password is password

<kubernetes-vm-ip>:30211

3. Open the browser and navigate to the ONAP Portal

Login using the Designer user. cs0008/demo123456!


4. Virtual Licence Model creation
   a. Open SDC application, click on the OnBoard tab.
      i. click Create new VLM (Licence Model)
         1. Use onap as Vendor Name, and enter a description
         2. click save
         3. click Licence Key Group and Add Licence KeyGroup, then fill in the required fields
         4. click Entitlements Pools and Add Entitlement Pool, then fill in the required fields
         5. click Feature Groups and Add Feature Group, then fill in the required fields. Also, under the Entitlement Pools tab, drag the created entitlement pool to the left. Same for the License Key Groups
         6. click Licence Agreements and Add Licence Agreement, then fill in the required fields. Under the tab Features Groups, drag the feature group created previously.
         7. then check-in and submit
      8. go back to OnBoard page

5. Vendor Software Product onboarding and testing
   a. click Create a new VSP
      i. First we create the vFW sinc; give it a name, i.e. vFW_SINC. Select the Vendor (onap) and the Category (Firewall) and give it a description.
      ii. Click on the warning, and add a licence model
      iii. Get the zip package: vfw-sinc.zip
      iv. Click on overview, and import the zip
      v. Click Proceed to validation then check-in then submit
   b. click Create a new VSP
      i. Then we create the vFW packet generator; give it a name, i.e. vFW_PG. Select the Vendor (onap) and the Category (Firewall) and give it a description.
ii. Click on the warning, and add a licence model
iii. Get the zip package: vfw_pg.zip
iv. Click on overview, and import the zip
v. Click Proceed to validation then check-in then submit

Go to SDC home. Click on the top right icon with the orange arrow.
i. Import the VSP one by one
ii. Submit for both testing

d. Logout and Login as the tester: jm0007/demo123456!
e. Go to the SDC portal
f. Test and accept the two VSP

6. Service Creation
a. Logout and login as the designer: cs0008/demo123456!
b. Go to the SDC home page
c. Click Add a Service
d. Fill in the required field
e. Click Create
f. Click on the Composition left tab
g. In the search bar, type “vFW” to narrow down the created VSP, and drag them both.
h. Then click Submit for Testing

7. Service Testing
a. Logout and Login as the tester: jm0007/demo123456!
b. Go to the SDC portal
c. Test and accept the service

8. Service Approval
a. Logout and Login as the governor: gv0001/demo123456!
b. Go to the SDC portal
c. Approve the service

9. Service Distribution
a. Logout and Login as the operator: op0001/demo123456!
b. Go to the SDC portal
c. Distribute the service
d. Click on the left tab monitor and click on arrow to open the distribution status
e. Wait until everything is distributed (green tick)

10. Service Instance creation:
a. Logout and Login as the user: demo/demo123456!
b. Go to the VID portal
c. Click the Browse SDC Service Models tab
d. Click Deploy on the service to deploy
e. Fill in the required field, call it vFW_Service for instance. Once done, this will redirect you to a new screen
f. Click Add VNF, and select the vFW_SINC VNF first
g. Fill in the required field. Call it vFW_SINC_VNF, for instance.
h. Click Add VNF, and select the vFW_PG_VNF first
i. Fill in the required field. Call it vFW_PG_VNF, for instance.

11. SDNC preload:
a. Then go to the SDNC Admin portal and create an account

   <kubernetes-host-ip>:30201/signup

b. Login into the SDNC admin portal

   <kubernetes-host-ip>:30201/login
c. Click Profiles then Add VNF Profile
I. The **VNF Type** is the string that looks like this: VfwPg..base_vpkg..module-0 It can be copy/paste from VID, when attempting to create the VF-Module

II. Enter 100 for the **Availability Zone Count**

III. Enter vFW for **Equipment Role**

IV. Repeat the same for the other VNF

e. Pre-load the vFW SINC. Mind the following values:

- **service-type**: it's the service instance ID of the service instance created step 9
- **vnf-name**: the name to give to the VF-Module. The same name will have to be re-use when creating the VF-Module
- **vnf-type**: Same as the one used to add the profile in SDNC admin portal
- **generic-vnf-name**: The name of the created VNF, see step 9f
- **vfw_name_0**: is the same as the generic-vnf-name
- **generic-vnf-type**: Can be find in VID, please see video if not found.
- **dcae_collector_ip**: Has to be the IP address of the dcaedoks00 VM

Make sure `image_name`, `flavor_name`, `public_net_id`, `onap_private_net_id`, `onap_private_subnet_id`, `key_name` and `pub_key` reflect your environment.

```bash
curl -X POST \\http://<kubernetes-host-ip>:30202/restconf/operations/VNF-API:preload-vnf-topology-operation \\-H 'accept: application/json' \\-H 'authorization: Basic YWRtaW46S3A4Yko0U1hzek0wV1hsaGFrM2V1bGNsZTJnQXc4NHZhb0d0d8Up2VXkyVQ==' \\-H 'content-type: application/json' \\-H 'x-fromappid: API client' \\-d '{
"input": {
  "vnf-topology-information": {
    "vnf-topology-identifier": {
      "service-type": "34992be5-b38c-46da-96b2-553e60f9c24b",
      "vnf-name": "VfwSINC_Module",
      "vnf-type": "VfwSINC..base_vfw..module-0",
      "generic-vnf-name": "vFW_SINC_VNF",
      "generic-vnf-type": "vFW_SINC 0"
    },
    "vnf-assignments": {
      "availability-zones": [],
      "vnf-networks": [],
      "vnf-vms": []
    },
    "vnf-parameters": [{
      "vnf-parameter-name": "image_name",
      "vnf-parameter-value": "trusty"
    },
    {"vnf-parameter-name": "flavor_name",
    "vnf-parameter-value": "m1.medium"},
    {"vnf-parameter-name": "public_net_id",
    "vnf-parameter-value": "d87ff178-3eb7-44df-a57b-84636dbdc817"},
    {"vnf-parameter-name": "unprotected_private_net_id",
    "vnf-parameter-value": "zdfwl1fw101_unprotected"},
    {"vnf-parameter-name": "unprotected_private_subnet_id",
    "vnf-parameter-value": "zdfwl1fw101_unprotected_sub"},
    {"vnf-parameter-name": "protected_private_net_id",
    "vnf-parameter-value": "zdfwl1fw101_protected"}]
  }
}
```
"vnf-parameter-name": "protected_private_subnet_id",
"vnf-parameter-value": "zdfw1fwl01_proTECTED_SUB"
},

[ "vnf-parameter-name": "onap_private_net_id",
  "vnf-parameter-value": "oam_onap_k0H4"
],

[ "vnf-parameter-name": "onap_private_subnet_id",
  "vnf-parameter-value": "oam_onap_k0H4"
],

[ "vnf-parameter-name": "unprotected_private_net_cidr",
  "vnf-parameter-value": "192.168.10.0/24"
],

[ "vnf-parameter-name": "protected_private_net_cidr",
  "vnf-parameter-value": "192.168.20.0/24"
],

[ "vnf-parameter-name": "onap_private_net_cidr",
  "vnf-parameter-value": "10.0.0.0/16"
],

[ "vnf-parameter-name": "vfw_private_ip_0",
  "vnf-parameter-value": "192.168.10.100"
],

[ "vnf-parameter-name": "vfw_private_ip_1",
  "vnf-parameter-value": "192.168.20.100"
],

[ "vnf-parameter-name": "vfw_private_ip_2",
  "vnf-parameter-value": "10.0.100.5"
],

[ "vnf-parameter-name": "vpg_private_ip_0",
  "vnf-parameter-value": "192.168.10.200"
],

[ "vnf-parameter-name": "vsn_private_ip_0",
  "vnf-parameter-value": "192.168.20.250"
],

[ "vnf-parameter-name": "vsn_private_ip_1",
  "vnf-parameter-value": "10.0.100.4"
],

[ "vnf-parameter-name": "vfw_name_0",
  "vnf-parameter-value": "vFW_SINC_VNF"
],

[ "vnf-parameter-name": "vsn_name_0",
  "vnf-parameter-value": "zdfw1fwl01snk01"
],

[ "vnf-parameter-name": "vnf_id",
  "vnf-parameter-value": "vFirewal_vSink_demo_app"
],

[ "vnf-parameter-name": "vf_module_id",
  "vnf-parameter-value": "vFirewall_vSink"
],

[ "vnf-parameter-name": "dcae_collector_ip",
  "vnf-parameter-value": "10.195.200.38"
],

[ "vnf-parameter-name": "dcae_collector_port",
  "vnf-parameter-value": "8080"
]
f. Pre-load the vFW PG. Mind the following values:
**service-type**: it's the service instance ID of the service instance created step 9

**vnf-name**: the name to give to the VF-Module. The same name will have to be re-use when creating the VF-Module

**vnf-type**: Same as the one used to add the profile in SDNC admin portal

**generic-vnf-name**: The name of the created VNF, see step 9h

**generic-vnf-type**: Can be find in VID, please see video if not found.

Make sure `image_name`, `flavor_name`, `public_net_id`, `onap_private_net_id`, `onap_private_subnet_id`, `key_name` and `pub_key` reflect your environment

```
curl -X POST  
-H 'accept: application/json'  
-H 'authorization: Basic YWRtaW46S3A4Yko0U1hzek0wV1hsaGFzM2VIbGNzETJnQXc4NNZhb0dHbUp2VXkyVQ=='  
-H 'content-type: application/json'  
-H 'x-fromappid: API client'  
-d '  
  "input": {  
    "vnf-topology-information": {  
      "vnf-topology-identifier": {  
        "service-type": "df6e075a-119a-4790-a470-2474a692e3ce",  
        "vnf-name": "vFW_PG_Module",  
        "vnf-type": "VfwPg..base_vpkg..module-0",  
        "generic-vnf-name": "vFW_PG_VNF",  
        "generic-vnf-type": "vFW_PG 0"  
      },  
      "vnf-assignments": {  
        "availability-zones": [  
        ],  
        "vnf-networks": [  
        ],  
        "vnf-vms": [  
        ]  
      },  
      "vnf-parameters": [  
        {  
          "vnf-parameter-name": "image_name",  
          "vnf-parameter-value": "trusty"  
        },  
        {  
          "vnf-parameter-name": "flavor_name",  
          "vnf-parameter-value": "m1.medium"  
        },  
        {  
          "vnf-parameter-name": "public_net_id",  
          "vnf-parameter-value": "d87ff178-3eb7-44df-a57b-84636dbdc817"  
        },  
        {  
          "vnf-parameter-name": "unprotected_private_net_id",  
          "vnf-parameter-value": "zdfw1fwl01_unprotected"  
        },  
        {  
          "vnf-parameter-name": "unprotected_private_subnet_id",  
          "vnf-parameter-value": "zdfw1fwl01_unprotected_sub"  
        },  
        {  
          "vnf-parameter-name": "onap_private_net_id",  
          "vnf-parameter-value": "oam_onap_k0H4"  
        },  
        {  
          "vnf-parameter-name": "onap_private_subnet_id",  
          "vnf-parameter-value": "oam_onap_k0H4"  
        },  
        {  
          "vnf-parameter-name": "unprotected_private_net_cidr",  
          "vnf-parameter-value": "192.168.10.0/24"  
        }  
      ]  
    }  
  
};
```
"request-information": {
  "request-id": "robot12",
  "order-version": "1",
  "notification-url": "openecomp.org",
  "order-number": "1",
  "request-action": "PreloadVNFRequest"
},
"sdnc-request-header": {
  "svc-request-id": "robot12",
  "svc-notification-url": "http://openecomp.org:8080/adapters/rest/SDNCNotify",
  "svc-action": "reserve"
}
}

Expected result:

```json
{
  "output": {
    "svc-request-id": "robot12",
    "response-code": "200",
    "ack-final-indicator": "Y"
  }
}
```

g. Create the VF-Module for vFW_SINC
   i. The instance name must be the `vnf-name` setup in the preload phase.
   ii. After a few minutes, the stack should be created.

h. Create the VF-Module for vFW_PG
   i. The instance name must be the `vnf-name` setup in the preload phase.
   ii. After a few minutes, the stack should be created.

Close loop

1. Run heatbridge robot tag to tell AAI about the relationship between the created HEAT stack (SINC one) and the service instance id.

   To run this, you need:
   - the heat stack name of the vSINC
   - the service instance id
2. Upload operational policy: this is to tell policy that for this specific instance, we should apply this policy.
   a. Retrieve from MSO Catalog the `modelInvariantUuid` for the `vFW_PG`. Specify in the below request the `service-model-name`, as defined step 5.c.

   ```bash
   curl -X GET 'http://<kubernetes-host>:30223/ecomp/mso/catalog/v2/serviceVnfs?serviceModelName=<service-model-names>'
   -H 'Accept: application/json'
   -H 'Authorization: Basic SW5mcmFQb3IyMVxObGllbnQ6cGFzc3dvcmQxJAM='
   -H 'Content-Type: application/json'
   -H 'X-FromAppId: Postman'
   -H 'X-TransactionId: get_service_vnfs'
   ```

Based on the payload below, result would be: `86a1bdd8-1f59-4796-bf30-3002108068f6`
b. Under
invoke the script as follows:

```
Usage: update-vfw-op-policy.sh <k8s-host> <policy-pdp-node-port> <policy-drools-node-port> <resource-id>
./update-vfw-op-policy.sh 10.195.197.53 30220 30221 86a1bddd8-1f59-4796-bf30-3002108068f
```

Result can look like, with debug enable (/bin/bash -x)

```
$ ./update-vfw-op-policy.sh 10.195.197.53 30220 30221 86a1bddd8-1f59-4796-bf30-3002108068f
+ '[' 4 -ne 4 ']
+ K8S_HOST=10.195.197.53
+ POLICY_PDP_PORT=30220
+ POLICY_DROOLS_PORT=30221
+ RESOURCE_ID=86a1bddd8-1f59-4796-bf30-3002108068f
+ echo
  + echo
    + echo 'Removing the vFW Policy from PDP..'
    Removing the vFW Policy from PDP..
+ echo
+ echo
  + curl -v -X DELETE --header 'Content-Type: application/json' --header 'Accept: text/plain' --
    header 'ClientAuth: cHl0aG9uOnRlc3Q=' --header 'Authorization: Basic dGVzdHBkcDphbHBoYTEyMw==' --
    header 'Environment: TEST' -d '{
      "pdpGroup": "default",
      "policyComponent": "PDP",
      "policyName": "com.BRMSParamvFirewall",
      "policyType": "BRMS_Param"
    }' http://10.195.197.53:30220/pdp/api/deletePolicy
  * Trying 10.195.197.53...
  * Connected to 10.195.197.53 (10.195.197.53) port 30220 (#0)
  > DELETE /pdp/api/deletePolicy HTTP/1.1
  > Host: 10.195.197.53:30220
  > User-Agent: curl/7.54.0
  > Content-Type: application/json
  > Accept: text/plain
  > ClientAuth: cHl0aG9uOnRlc3Q=
  > Authorization: Basic dGVzdHBkcDphbHBoYTEyMw==
  > Environment: TEST
  > Content-Length: 128
  >
  * upload completely sent off: 128 out of 128 bytes
  < HTTP/1.1 200 OK
  < Server: Apache-Coyote/1.1
  < Content-Type: text/plain; charset=ISO-8859-1
  < Content-Length: 91
  < Date: Wed, 20 Dec 2017 20:17:22 GMT
  <
  * Connection #0 to host 10.195.197.53 left intact
Transaction ID: af030f0c-0c2b-43a1-blec-6abf4ca73799 --The policy was successfully deleted.
+ sleep 20

+ echo
+ echo
+ echo 'Updating vFW Operational Policy ..'
```
Updating vFW Operational Policy ..
+ echo

  "policyConfigType": "BRMS_PARAM",
  "policyName": "com.BRMSParamvFirewall",
  "policyDescription": "BRMS Param vFirewall policy",
  "policyScope": "com",
  "attributes": {
    "WATCHING": {
      "controller": "amsterdam"
    }
  },
  "RULE": {
    "templateName": "ClosedLoopControlName",
    "closedLoopControlName": "ControlLoop-vFirewall-d0a1dfc6-94f5-4fd4-a5b5-4630b438850a",
    "controlLoopYaml": "controlLoop%0A+++version%3A+2.0.0%0A+++controlLoopName%3A+ControlLoop-vFirewall-d0a1dfc6-94f5-4fd4-a5b5-4630b438850a%0A+++trigger_policy%3A+unique-policy-id-1-modifyConfig%0A+++timeout%3A+1200%0A+++abatement%3A+true%0A+++policies%3A%0A++id%3A+policy-update-2.0.0.0%0A+++name%3A+modify-packet-gen+packet-card%0A+++template%3A+ModifyConfig%0A+++target%3A+unique-policy-id-1-modifyConfig%0A+++retry%3A+0%0A+++timeout%3A+300%0A+++success%3A+final_success%0A+++failure%3A+final_failure%0A+++failure_timeout%0A+++failure_retries%3A+1%0A+++failure_exception%3A+final_failure_exception%0A+++failure_guard%3A+final_failure_guard%0A
}"
}

* Trying 10.195.197.53...
* Connected to 10.195.197.53 (10.195.197.53) port 30220 (#0)
> PUT /pdp/api/updatePolicy HTTP/1.1
> Host: 10.195.197.53:30220
> User-Agent: curl/7.54.0
> Content-Type: application/json
> Accept: text/plain
> ClientAuth: ch10aG9uOnRlc3Q=
> Authorization: Basic dGVzdHBkcDphbHBoYTEyMw==
> Environment: TEST
> Content-Length: 1327
> Expect: 100-continue
> HTTP/1.1 100 Continue
> We are completely uploaded and fine
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 158
< Date: Wed, 20 Dec 2017 20:17:42 GMT
<
*
Connection #0 to host 10.195.197.53 left intact
Transaction ID: 65316b73-d193-446c-8ece-ee643a85f45f --Policy with the name com.Config_BRMS_Param_BRMSParamvFirewall.2.xml was successfully updated. + sleep 5
+ echo

+ echo 'Pushing the vFW Policy ..'
PUSHING the vFW Policy ..
+ echo

+ echo

  "pdpGroup": "default",
  "policyName": "com.BRMSParamvFirewall",
  "policyType": "BRMS_Param"
}''
B. http://10.195.197.53:30220/pdp/api/pushPolicy
* Trying 10.195.197.53...
* TCP_NODELAY set
* Connected to 10.195.197.53 (10.195.197.53) port 30220 (#0)
> PUT /pdp/api/pushPolicy HTTP/1.1
> Host: 10.195.197.53:30220
> User-Agent: curl/7.54.0
> Content-Type: application/json
> Accept: text/plain
> ClientAuth: cHl0aG9uOnRlc3Q=
> Authorization: Basic dGVzdHBkcDphbHBoYTEyMw==
> Environment: TEST
> Content-Length: 99
>
* upload completely sent off: 99 out of 99 bytes
HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: text/plain;charset=ISO-8859-1
< Content-Length: 162
< Date: Wed, 20 Dec 2017 20:17:48 GMT
<
> Transaction ID: e8bc4ae1-d0b0-483e-b1ba-871486661240 --Policy 'com.
Config_BRMS_Param_BRMSParamvFirewall.2.xml' was successfully pushed to the PDP group 'default'.
sleep 20
+ echo
+ echo
+ echo 'Restarting PDP-D ..'
Restarting PDP-D ..
+ echo
+ echo
++ kubectl --namespace onap-policy get pods
++ sed '/s/ .*//'
++ grep drools
+ POD=drools-870120400-5b5k1
+ kubectl --namespace onap-policy exec -it drools-870120400-5b5k1 -- bash -c 'source /opt/app
/policy/etc/profile.d/env.sh && policy stop && sleep 5 && policy start'
Defaulting container name to drools.
Use 'kubectl describe pod/drools-870120400-5b5k1' to see all of the containers in this pod.
[drools-pdp-controllers]

L []: Stopping Policy Management... Policy Management (pid=5452) is stopping... Policy Management has stopped.
[drools-pdp-controllers]
L []: Policy Management (pid 5722) is running
+ sleep 20
+ echo
+ echo
+ echo 'PDP-D amsterdam maven coordinates ..'
PDP-D amsterdam maven coordinates..
+ echo
+ echo
/controllers/amsterdam/drools
+ python -m json.tool
* Trying 10.195.197.53...
* TCP_NODELAY set
* Connected to 10.195.197.53 (10.195.197.53) port 30221 (#0)
* Server auth using Basic with user '0b3rt'
> GET /policy/pdp/engine/controllers/amsterdam/drools HTTP/1.1
> Host: 10.195.197.53:30221
> Authorization: Basic QDFiM3JD0jMxbnNOMzFu
> User-Agent: curl/7.54.0
> Accept: */*
> 
> < HTTP/1.1 200 OK
> < Date: Wed, 20 Dec 2017 20:18:49 GMT
> < Content-Type: application/json
> < Content-Length: 302
> < Server: Jetty(9.3.14.v20161028)
> 
> {382 bytes data}

```
} + echo
+ echo

+ echo 'PDP-D control loop updated ..'
PDP-D control loop updated ..
+ echo
+ echo

+ python -m json.tool
*   Trying 10.195.197.53...
* TCP_NODELAY set
* Connected to 10.195.197.53 (10.195.197.53) port 30221 (#0)
* Server auth using Basic with user '@1b3rt'
> GET /policy/pdp/engine/controllers/amsterdam/drools/facts/closedloop-amsterdam/org.onap.policy.controlloop.Params HTTP/1.1
> Host: 10.195.197.53:30221
> Authorization: Basic QDFiM3JD0jMxbnNOMzFu
> User-Agent: curl/7.54.0
> Accept: */*
> ...
> < HTTP/1.1 200 OK
> < Date: Wed, 20 Dec 2017 20:18:50 GMT
> < Content-Type: application/json
> < Content-Length: 3565
> < Server: Jetty(9.3.14.v20161028)
> 
> {1207 bytes data}

```
```
3. Mount APPC
   a. Get the VNF instance ID, either through VID or through AAI. Below the AAI request

```
curl -X GET \
https://<kubernetes-host>:30233/aai/v8/network/generic-vnfs/ \
-H 'Accept: application/json' \
-H 'Authorization: Basic QUFJOkFBSQ==' \
-H 'Content-Type: application/json' \
-H 'X-FromAppId: Postman' \
-H 'X-TransactionId: get_generic_vnf'
```

In the result, search for the `vFW_PG_VNF` and get its vnf-id. In the payload below, it would be `e6fd60b4-f436-4a21-963c-cc9060127633`
{  "relationship-key": "service-subscription.service-type",  "relationship-value": "vFWCL"},
{  "relationship-key": "service-instance.service-instance-id",  "relationship-value": "63b55891-ebc6-40bf-b884-2e2427280a83"}
],
"related-to-property": [
{  "property-key": "service-instance.service-instance-name",  "property-value": "vFWServiceInstance-20-12"}
]
}
]
"
"vf-modules": {  "vf-module": [
{  "vf-module-id": "c2fed873-263c-46b5-bb95-4dfaf6c02410",
  "vf-module-name": "vFW_PG_Module",
  "heat-stack-id": "vFW_PG_Module/850e7e4a-6c36-405c-8058-7f3fa25ca42e",
  "orchestration-status": "active",
  "is-base-vf-module": true,
  "resource-version": "1513791913543",
  "persona-model-id": "9c6c0369-9c14-4419-94c9-2abc6250087",
  "persona-model-version": "1"  
}  
}
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]
"
}
]


```json
{
  "node": {
    "node-id": "e6fd60b4-f436-4a21-963c-cc9060127633",
    "netconf-node-topology:available-capabilities": {
      "available-capability": [
        {
          "capability-origin": "device-advertised",
          "capability": "urn:ietf:params:netconf:capability:exi:1.0"
        },
        {
          "capability-origin": "device-advertised",
          "capability": "urn:ietf:params:netconf:capability:candidate:1.0"
        },
        {
          "capability-origin": "device-advertised",
          "capability": "urn:ietf:params:netconf:base:1.1"
        },
        {
          "capability-origin": "device-advertised",
          "capability": "urn:ietf:params:netconf:base:1.0"
        },
        {
          "capability-origin": "device-advertised",
        },
        {
          "capability-origin": "device-advertised",
        },
        {
          "capability-origin": "device-advertised",
        },
        {
          "capability-origin": "device-advertised",
        },
        {
          "capability-origin": "device-advertised",
        },
        {
          "capability-origin": "device-advertised",
          "capability": "(urn:ietf:params:xml:ns:yang:ietf-access-control-list?revision=2016-07-08)ietf-access-control-list"
        },
        {
          "capability-origin": "device-advertised",
        },
        {
          "capability-origin": "device-advertised",
        },
        {
          "capability-origin": "device-advertised",
        },
        {
          "capability-origin": "device-advertised",
        }
      ]
    }
  }
}
```
"capability-origin": "device-advertised",
"capability": ":device-advertised",
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
"capability": "(urn:ietf:params:xml:ns:yang:iana-if-type?revision=2014-05-08)iana-if-type"
},

"capability-origin": "device-advertised",
},

"capability-origin": "device-advertised",
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
"capability": "(urn:.ieee:params:xml:ns:yang:dot1q-types?revision=2015-06-26)dot1q-types"
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",
},
{
"capability-origin": "device-advertised",null
}
Using NETCONF, let's get the current streams being active in our Packet Generator. The number of streams will change along the time, this is the result of close-loop policy. When the traffic goes over a certain threshold, DCAE will publish an event on the unauthenticated DCAE_CL_OUTPUT topic that will be picked up by APPC, that will send a NETCONF request to the packet generator to adjust the traffic it's sending.
3. Send a NETCONF request to the packet generator to adjust the traffic:

```bash
curl -X GET \
  -H 'Accept: application/json' \
  -H 'Authorization: Basic YWRtaW46YWRtaW46S3A4Yko0U1hzek0wV1hsaGFrM2V1bGNzETJnQXc4NH2hb0dHbUp2VXkyVQ=='
```

4. Browse to the zdfw1w01snk01 on port 667 to see a graph representing the traffic being received:

```
http://<zdfw1fwl01snk01>:667/
```

As you can see in the below graph, looking at the top right square, we can see the first two fluctuations are going from very low to very high. This is when close-loop isn't running.

Once close-loop is running, you'll have some medium bars.

5. Check the events sent by Virtual Event Collector (VES) to Threshold Crossing Analytic (TCA) app:

```
curl -X GET \
  http://<K8S_IP>:3904/events/unauthenticated.SEC_MEASUREMENT_OUTPUT/group1/C1 \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/cambria'
```

The VES resides in the VNF itself, whereas the TCA is an application running on Cask. A DCAE component.

6. Check the events sent by TCA on unauthenticated.DCAE_CL_OUTPUT:

```
curl -X GET \
  http://<K8S_IP>:3904/events/unauthenticated.DCAE_CL_OUTPUT/group1/C1 \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/cambria'
```

Those events are the resulting of the TCA application, e.g. TCA has noticed an event was crossing a given threshold, hence is sending a message of that particular topic. Then Policy will grab this event and perform the appropriate action, as defined in the Policy. In the case of vFWCL, Policy will send an event on the APPC_CL topic, that APPC will consume. This will trigger a NETCONF request to the packet generator to adjust the traffic.

I hope everything worked for you, if not, please leave a comment. Thanks