HOW ONAP ORCHESTRATES A CNF TO WRCP

bin.yang@windriver.com

Contents

Overview

This is a detailed How-To document to illustrate how users could leverage ONAP to orchestrate CNF to WRCP 19.12 instance.

The general deployment topology is depicted as diagram below:

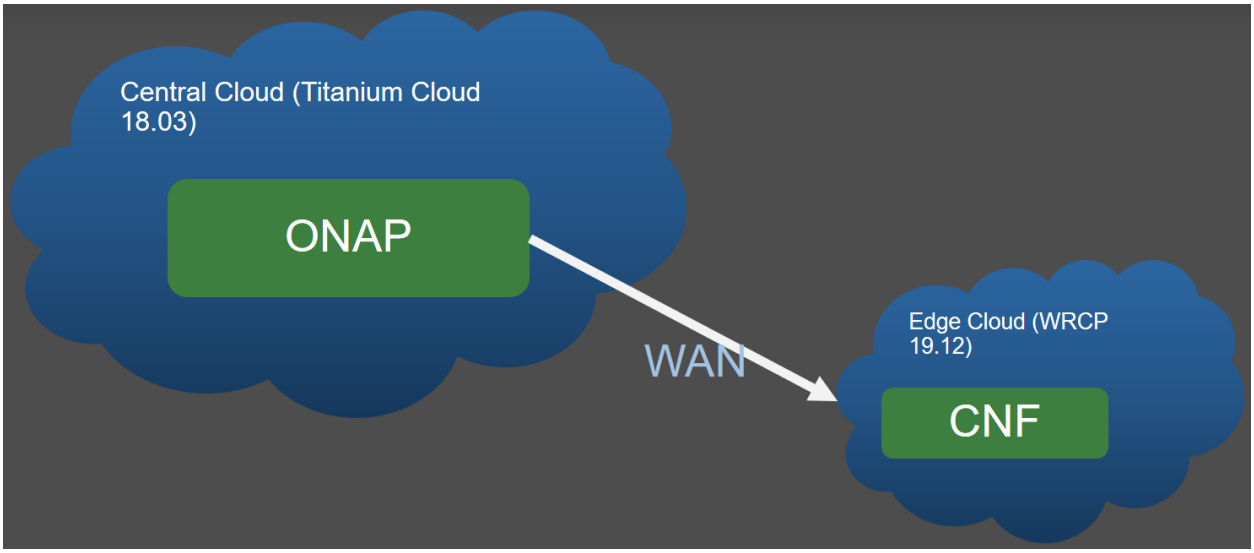


Diagram 1: ONAP and the cloud hosting it, WRCP, CNF topology

The comprehensive workflow consists of following phases:

Phase 1: WRCP 19.12 installation and provisions

Phase 2: CNF development and validation

Phase 3: ONAP installation and provisions

Phase 4: Register WRCP instance to ONAP

Phase 5: CNF onboarding and service design

Phase 6: Service Instantiation hence CNF instantiation and validation

Phase 7: Service deletion hence CNF deletion

Phase 1: WRCP 19.12 installation and provisions

WRCP 19.12 installation could be AIO Simplex, Duplex, and Standard type.

Due to CNF requiring multiple networking plane, the WRCP must be provisioned with:

- Datanetwork backend by SRIOV netdevice (at least 2 vlan ID to support 2 network planes of the cFW use case)
    - In case This SRIOV netdevice is not available, host netdevice passthrough could be used as well.
    - veth pair could also be used for demonstration purpose, with constraint that all pods should be scheduled to the same worker node
- Hugepage-2M: 512x2M for each NUMA node for worker nodes
- Dedicated Tenant and user with admin role
- Kubernetes service account with clusterrolebindings and privileges to operate various resource, including namespace, etc.

Phase 2: CNF development and validation

The CNF should be developed and validated over WRCP 19.12 directly (without ONAP's orchestration)

The example CNF is containerized Firewall use case (referred as cFW in context below):

https://gerrit.onap.org/r/gitweb?p=multicloud/k8s.git;a=tree;f=starlingx/demo;h=44ab83ca5c5c9f01082695b1aa9a6e71fdaeec20;hb=HEAD

It consists of 3 pods, connected through 2 network planes. The topology is depicted as below:

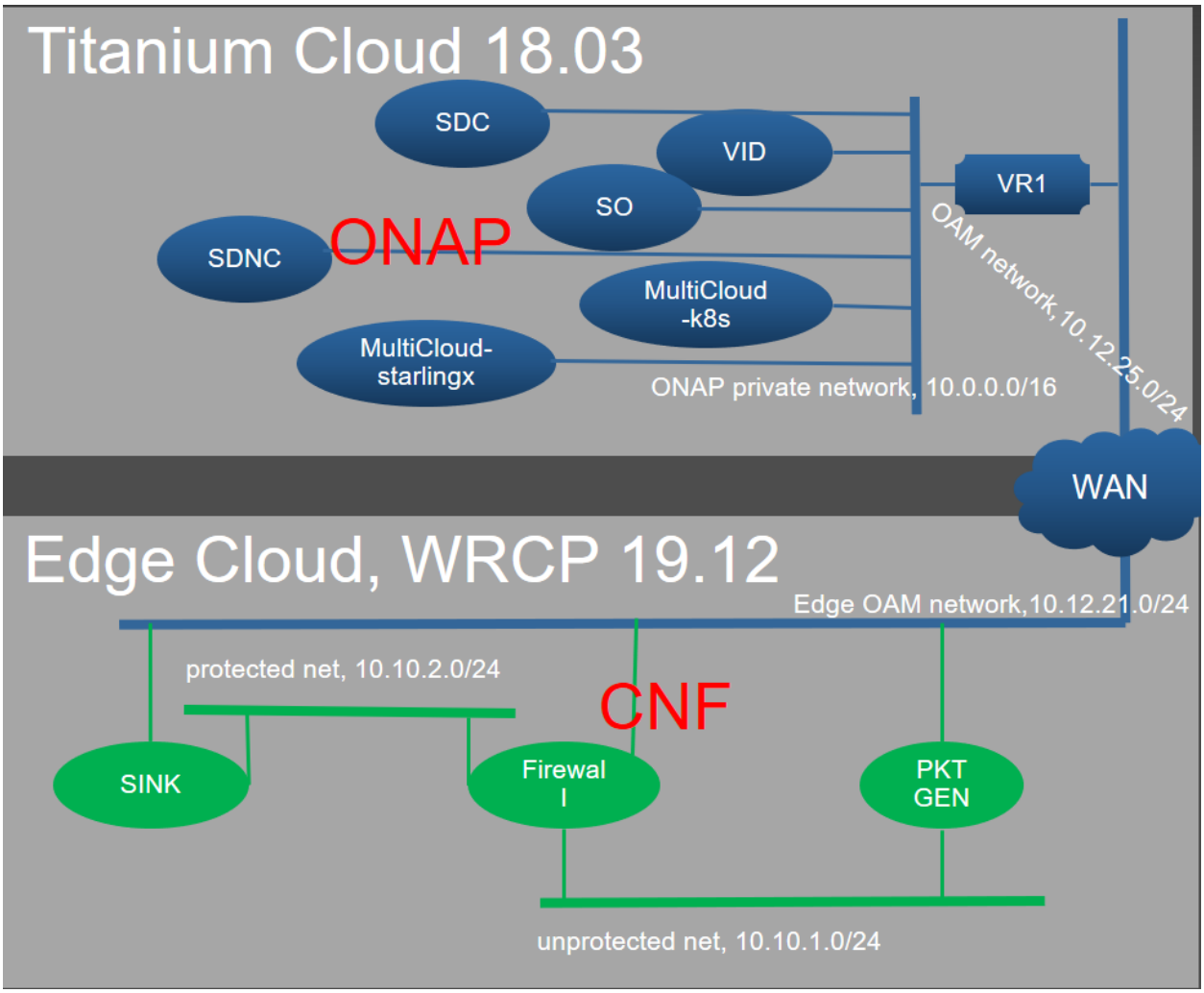


Diagram2: ONAP and cFW components

Use helm to validate the CNF,e.g. cFW:

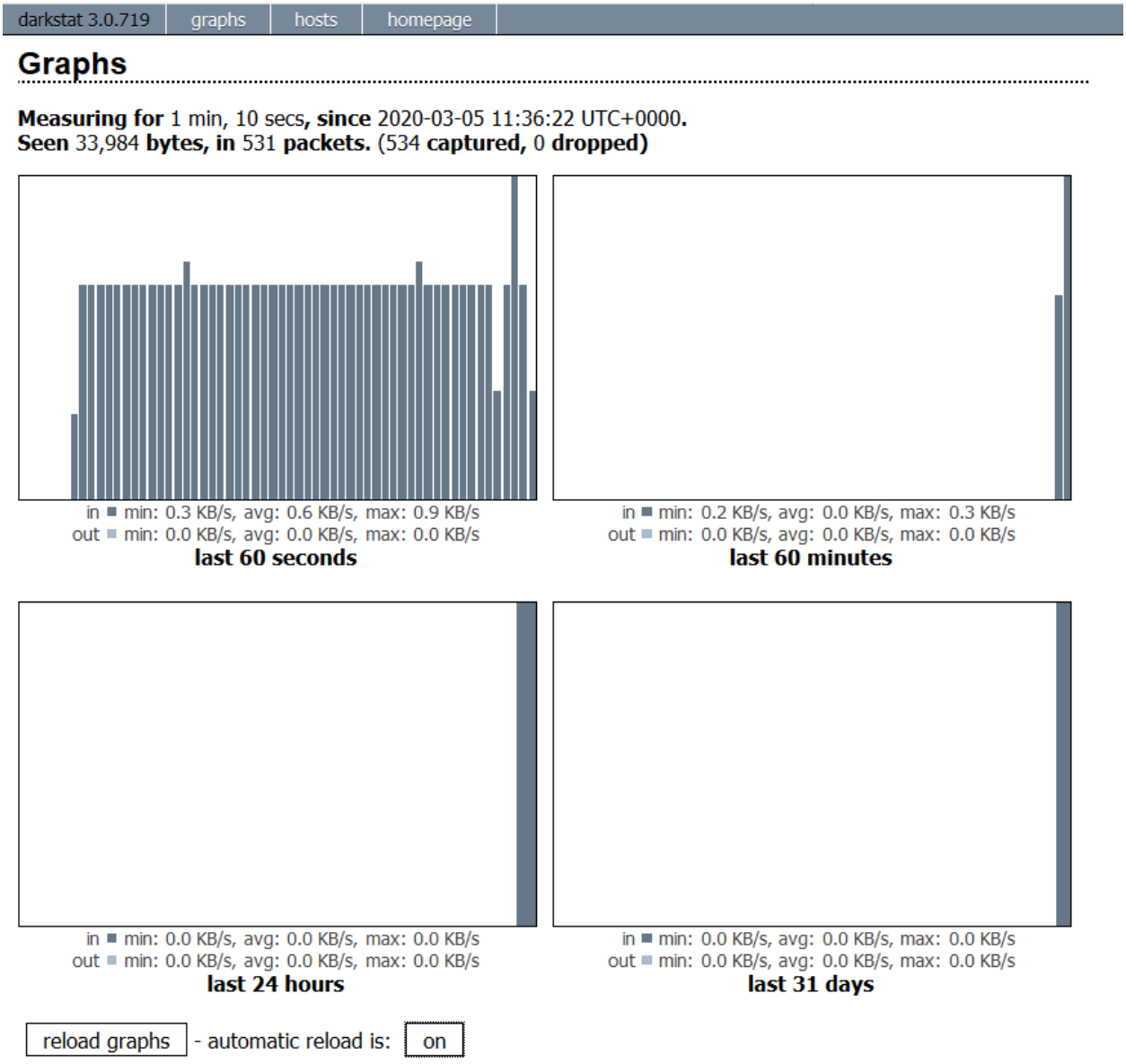
```
$ git clone "https://gerrit.onap.org/r/multicloud/k8s"
```

$ cd k8s/starlingx/demo

```
$helm install firewall-sriov -n cfw1
```

To validate of the deployed cFW, monitoring the traffic over SINK pod with your browser (Chrome, Firefox) to open following url (replace NODE_IP with any worker node IP of the WRCP instance):
http://$NODE_IP:30667/

You should observe traffic diagram likes below:



Snapshot 1: sink traffic monitor page

Phase 3: ONAP installation and provisions

Deploy ONAP instance with el alto release, with overrides to update multicloud services

There are 2 alternative approaches to deploy ONAP instance:

Approach 1: deploy the whole stack from openstack, including Heat stack, k8s cluster, and ONAP instance

prepare: download openrc file from openstack horizon, e.g. VIM-openrc

update the attached onap-oom-lite.env with openstack instance's information.

ONAP deployment might take hours so it is better to initiate the process by screen terminal:

$ screen -R onapdeploy

$ git clone https://git.onap.org/integration

```
$ cd integration/deployment/heat/onap-rke/
```

$ copy the attached file as `onap-oom-lite.env`

$ source <openrc file>

```
$ ./scripts/deploy.sh -n 15 -s mctest1  -i elalto -o elalto onap-oom-lite.env
```

You should observe following message from screen terminal while bootstrap process is accomplished.

```
Cloud-init v. 18.2 running 'modules:final' at Mon, 02 Mar 2020 03:24:21 +0000. Up 18.87 seconds.
Cloud-init v. 18.2 finished at Mon, 02 Mar 2020 04:17:14 +0000. Datasource DataSourceOpenStack [net,ver=2].  Up 3132.34 seconds
+ exit 0
```

Snapshot 2: ONAP deployment bootstrap accomplished

Approach 2: deploy the ONAP instance over existing k8s cluster

Update the attached 'integration-override.yaml' . then apply following commands

```
$ git clone --recurse-submodules -b elalto "https://git.onap.org/oom"
$ cd oom
$ rsync -avt kubernetes/helm/plugins ~/.helm/
$sed -i "/\^enabled:/a\      echo sleep 30s\n      sleep 30s" ~/.helm/plugins/deploy/deploy.sh
$ sed -i 's/for subchart in \*/for subchart in aaf cassandra mariadb-galera dmaap */' ~/.helm/plugins/deploy/deploy.sh
$ helm deploy dev local/onap -f ./kubernetes/onap/resources/environments/public-cloud.yaml -f ./integration-override.yaml --namespace onap
$ rsync -avt kubernetes/helm/plugins ~/.helm/
```

Update SO configurations

----ONAP SO VNF Adapter Rest API endpoint version shall be set to version “v2”

$ kubectl -n onap get configmap | grep so-so-bpmn-infra-app-configmap

$ kubectl -n onap edit configmap dev-so-so-bpmn-infra-app-configmap

in the section “vnf”, modify the rest endpoint:

```
      vnf:
        endpoint: http://so-openstack-adapter.onap:8087/services/VnfAdapter
        rest:
-         endpoint: http://so-openstack-adapter.onap:8087/services/rest/v1/vnfs
+         endpoint: http://so-openstack-adapter.onap:8087/services/rest/v2/vnfs
      volume-groups:
        rest:
          endpoint: http://so-openstack-adapter.onapg:8087/services/rest/v1/volume-groups
```

$ kubectl get po -n onap |grep bpmn-infra

$ kubectl -n onap delete pod dev-so-so-bpmn-infra-65945c685d-cfw92

check if pods restarted

$ kubectl -n onap get po | grep so-so

ONAP healthcheck

$ cd oom/kubernetes/robot/

$ ./ete-k8s.sh onap health

<example output of ONAP health check>

Populate ONAP with demonstration data by robot script

$ cd oom/kubernetes/robot/

$ ./demo-k8s.sh init

Now wait about half an hour for completion of the demo data population

Access to ONAP portals

Update hosts with following entries: <e.g. assume 10.12.6.76 is a k8s cluster node IP>

10.12.6.76 portal.api.simpledemo.onap.org

10.12.6.76 vid.api.simpledemo.onap.org

10.12.6.76 sdc.api.fe.simpledemo.onap.org

10.12.6.76 sdc.api.be.simpledemo.onap.org

10.12.6.76 sdc.workflow.plugin.simpledemo.onap.org

10.12.6.76 sdc.dcae.plugin.simpledemo.onap.org

10.12.6.76 portal-sdk.simpledemo.onap.org

10.12.6.76 policy.api.simpledemo.onap.org

10.12.6.76 aai.api.sparky.simpledemo.onap.org

10.12.6.76 cli.api.simpledemo.onap.org

10.12.6.76 msb.api.discovery.simpledemo.onap.org

10.12.6.76 msb.api.simpledemo.onap.org

10.12.6.76 clamp.api.simpledemo.onap.org

10.12.6.76 so.api.simpledemo.onap.org

10.12.6.76 sdnc.api.simpledemo.onap.org

10.12.6.76 so-monitoring

Use a browser (Chrome, Firefox) to open the following URL, and input username/password:

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

Here is the list of users with roles, passwords defaults to “demo123456!”:

Role	User ID	Password	
designer	cs0008	demo123456!	
tester	jm0007	demo123456!	
governance Rep	gv0001	demo123456!	
ops	op0001	demo123456!	
admin	demo	demo123456!	

Note 1: The first time to access applications of ONAP portal might end up with certification error, to workaround that, you need browse the following urls and add them as security exceptions.

1, https://sdc.api.fe.simpledemo.onap.org:30207/sdc1/portal#!/adminDashboard

2, https://vid.api.simpledemo.onap.org:30200/vid/welcome.htm

Note 2: in case that so-monitoring GUI fails to show up, use browser to open the following url directly: http://so-monitoring:30224/

Phase 4: Register WRCP instance to ONAP

Registering WRCP to ONAP demands following information and multiple manual steps via curl command or postman:

- Keystone endpoint URL for OpenStack API access, along with Project(Tenant) name, Domain name, User ID, Password
- Tiller endpoint URL and Service account Token for Kubernetes API access
- Figure out a cloud region ID, e.g. Cloud Owner = WRCP2, Cloud Region ID = STXRegionOne

Step 1: Create a SO Cloud Site

```
$ kubectl -n onap get pod | grep mariadb-galera

$ kubectl -n onap exec -ti dev-mariadb-galera-mariadb-galera-0 sh

        mysql --user=so_admin --password=so_Admin123

        USE catalogdb

        select * from cloud_sites;

        INSERT INTO cloud_sites(ID, REGION_ID, IDENTITY_SERVICE_ID, CLOUD_VERSION, CLLI,
ORCHESTRATOR) values("STXRegionOne", "STXRegionOne", "DEFAULT_KEYSTONE", "2.5",
"My_Complex", "multicloud");

        select * from cloud_sites;

        quit

$ exit
```

Step 2: Create an AAI Cloud Region along with complex

Post following RestAPI request to ONAP MSB endpoint via curl command or postman (replace those placeholder marked by <>):

```
$ curl -X PUT \

https://aai.api.sparky.simpledemo.onap.org:30233/aai/v16/cloud-
infrastructure/complexes/complex/My_Complex \
```

```
-H 'Accept: application/json' \
-H 'Authorization: Basic QUFJOkFBSQ==' \
-H 'Cache-Control: no-cache' \
-H 'Content-Type: application/json' \
-H 'Real-Time: true' \
-H 'X-FromAppId: jimmy-postman' \
-H 'X-TransactionId: 9999' \
-d '{
  "physical-location-id": "My_Complex",
  "data-center-code": "example-data-center-code-val-5556",
  "complex-name": "My_Complex",
  "identity-url": "example-identity-url-val-56898",
  "physical-location-type": "example-physical-location-type-val-7608",
  "street1": "example-street1-val-34205",
  "street2": "example-street2-val-99210",
  "city": "Beijing",
  "state": "example-state-val-59487",
  "postal-code": "100000",
  "country": "example-country-val-94173",
  "region": "example-region-val-13893",
  "latitude": "39.9042",
  "longitude": "106.4074",
  "elevation": "example-elevation-val-30253",
  "lata": "example-lata-val-46073"
  }' –k

$ CLOUD_OWNER=WRCP2
```

```
$ CLOUD_REGIONID=STXRegionOne

$ curl -X PUT \

https://aai.api.sparky.simpledemo.onap.org:30233/aai/v16/cloud-infrastructure/cloud-regions/cloud-region/${CLOUD_OWNER}/${CLOUD_REGIONID} \

-H 'Accept: application/json' \

-H 'Authorization: Basic QUFJOkFBSQ==' \

-H 'Cache-Control: no-cache' \

-H 'Content-Type: application/json' \

-H 'Postman-Token: 8b9b95ae-91d6-4436-90fa-69cb4d2db99c' \

-H 'Real-Time: true' \

-H 'X-FromAppId: jimmy-postman' \

-H 'X-TransactionId: 9999' \

-d '{

   "cloud-owner": "WRCP2",

   "cloud-region-id": "STXRegionOne",

   "cloud-type": "openstack",

   "owner-defined-type": "t1",

   "cloud-region-version": "starlingx",

   "complex-name": "My_Complex",

   "cloud-zone": "CloudZone",

   "sriov-automation": false,

   "identity-url": "",

   "cloud-extra-info":"{\"openstack-region-id\":\"RegionOne\",\"k8s-apiserver\":\"https://<wrcp controller IP>:6443\",\"k8s-apitoken\":\"<service account token>\"}",

    "relationship-list": {

       "relationship": [

          {

             "related-to": "complex",
```

```
            "relationship-label": "org.onap.relationships.inventory.LocatedIn",
            "related-link": "/aai/v16/cloud-infrastructure/complexes/complex/My_Complex",
            "relationship-data": [
              {
                "relationship-key": "complex.physical-location-id",
                "relationship-value": "My_Complex"
              }
            ]
          }
        ]
      },
    "esr-system-info-list": {
      "esr-system-info": [
        {
          "esr-system-info-id": "55f97d59-6cc3-49df-8e69-926565f00055",
          "service-url": "http://<wrcp controller IP>:5000/v3",
          "user-name": "<OpenStack username>",
          "password": "<Openstack user pass>",
          "system-type": "VIM",
          "ssl-insecure": true,
          "cloud-domain": "Default",
          "default-tenant": "<OpenStack Project/Tenant name, e.g. onap-sb-01>"
        }
      ]
    }
}' -k
```

Step 3: Trigger MultiCloud registration process:

```
$ curl -X POST \
https://msb.api.discovery.simpledemo.onap.org:30283/api/multicloud-starlingx/v1/${CLOUD_OWNER}/${CLOUD_REGIONID}/registry \
-H 'Accept: application/json' \
-H 'Cache-Control: no-cache' \
-H 'Content-Type: application/json' -k
```

Step 4: associate subscription with Cloud Region

Create Customer if necessary (e.g. democustomer1) :

```
    $ curl -k --location --request PUT 'https://aai.api.sparky.simpledemo.onap.org:30233/aai/v16/business/customers/customer/democustomer1' \
    --header 'Authorization: Basic QUFJOkFBSQ==' \
    --header 'X-FromAppId: AAI' \
    --header 'Accept: application/json' \
    --header 'Content-Type: application/json' \
    --header 'X-TransactionId: 808b54e3-e563-4144-a1b9-e24e2ed93d4f' \
    --data-raw '{
      "global-customer-id": "democustomer1",
      "subscriber-name": "democustomer1",
      "subscriber-type": "INFRA"
    }'
```

Step 4: Add service type “cfw-k8s”:

```
$ curl -k --location --request PUT 'https://aai.api.sparky.simpledemo.onap.org:30233/aai/v16/service-design-and-creation/services/service/cfw-k8s' \
        --header 'Authorization: Basic QUFJOkFBSQ==' \
        --header 'X-FromAppId: AAI' \
        --header 'Accept: application/json' \
        --header 'X-TransactionId: 808b54e3-e563-4144-a1b9-e24e2ed93d4f' \
        --header 'Content-Type: application/json' \
        --data-raw '{
        "service-id": "cfw-k8s",
        "service-description": "cfw-k8s"
        }'

        $ curl -k --location --request PUT 'https://aai.api.sparky.simpledemo.onap.org:30233/aai/v16/business/customers/customer/democustomer1/service-subscriptions/service-subscription/cfw-k8s' \
        --header 'Authorization: Basic QUFJOkFBSQ==' \
        --header 'X-FromAppId: AAI' \
        --header 'Accept: application/json' \
        --header 'Content-Type: application/json' \
        --header 'X-TransactionId: 808b54e3-e563-4144-a1b9-e24e2ed93d4f' \
        --data-raw '{
           "service-id": "cfw-k8s"
        }'
```

Step 5: Associate subscription to Cloud Region:

```
$ curl -k --location --request PUT 'https://aai.api.sparky.simpledemo.onap.org:30233/aai/v16/business/customers/customer/democustomer1/service-subscriptions/service-subscription/cfw-k8s/relationship-list/relationship' \
```

```
--header 'Authorization: Basic QUFJOkFBSQ==' \
--header 'X-FromAppId: AAI' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--header 'X-TransactionId: 808b54e3-e563-4144-a1b9-e24e2ed93d4f' \
--data-raw '{
  "related-to": "tenant",
  "related-link": "/aai/v16/cloud-infrastructure/cloud-regions/cloud-region/WRCP2/STXRegionOne/tenants/tenant/fd32fdd20ff5467ebef2de63468eb2e4",
  "relationship-data": [
    {
      "relationship-key": "cloud-region.cloud-owner",
      "relationship-value": "WRCP2"
    },
    {
      "relationship-key": "cloud-region.cloud-region-id",
      "relationship-value": "STXRegionOne"
    },
    {
      "relationship-key": "tenant.tenant-id",
      "relationship-value": "fd32fdd20ff5467ebef2de63468eb2e4"
    }
  ],
  "related-to-property": [
    {
      "property-key": "tenant.tenant-name",
      "property-value": "onap-sb-01"
```

```
        }
    ]
}'
```

Phase 5: CNF onboarding and service design

Refer to Phase 2, which develops and validates the CNF (cFW in this case), the following steps should be followed to onboard it to ONAP for service design.

Step 1: create a tar ball for the CNF helm chart

```
$ cd k8s/starlingx/demo

$ CNF_NAME=" cfwsriov1"

$ CNF_ARTIFACT_NAME="${CNF_NAME}_cloudtech_k8s_charts.tgz"

$ tar -czvf $CNF_ARTIFACT_NAME    firewall-sriov/
```

Step 2: Wrap helm chart tar ball into a dummy heat template artifact

Copy the attachment files here: base_dummy.yaml , base_dummy.env

then apply following commands:

```
$ cat <<EOF> MANIFEST.json

{

  "name": "",

  "description": "",

  "data": [

    {

      "file": "base_dummy.yaml",

      "type": "HEAT",

      "isBase": "true",

      "data": [

        {
```

```
            "file": "base_dummy.env",
            "type": "HEAT_ENV"
          }
        ]
      },
      {
        "file": "$CNF_ARTIFACT_NAME",
        "type": "CLOUD_TECHNOLOGY_SPECIFIC_ARTIFACTS"
      }
    ]
}
EOF
```

$ zip ${CNF_NAME}_vsp.zip base_dummy.env base_dummy.yaml MANIFEST.json $CNF_ARTIFACT_NAME

Now you have the VSP artifact named 'cfwsriov1_vsp.zip' ready for onboarding to SDC

Step 3: Onboard the VSP artifact 'cfwsriov1_vsp.zip' into ONAP SDC

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

Open SDC application from ONAP portal, onboard the artifact "cfwsriov1_vsp.zip" as VSP, import it as a VF 'Cfwsriov1' (follow the instructions: https://docs.onap.org/en/elalto/guides/onap-user/design/vfcreation/index.html) .

Create Service 'cfwsvc1' and add the VF 'cfwsriov1'', test the service model, approve it, and distribute it (follow the instructions: https://docs.onap.org/en/elalto/guides/onap-user/design/service-design/index.html).

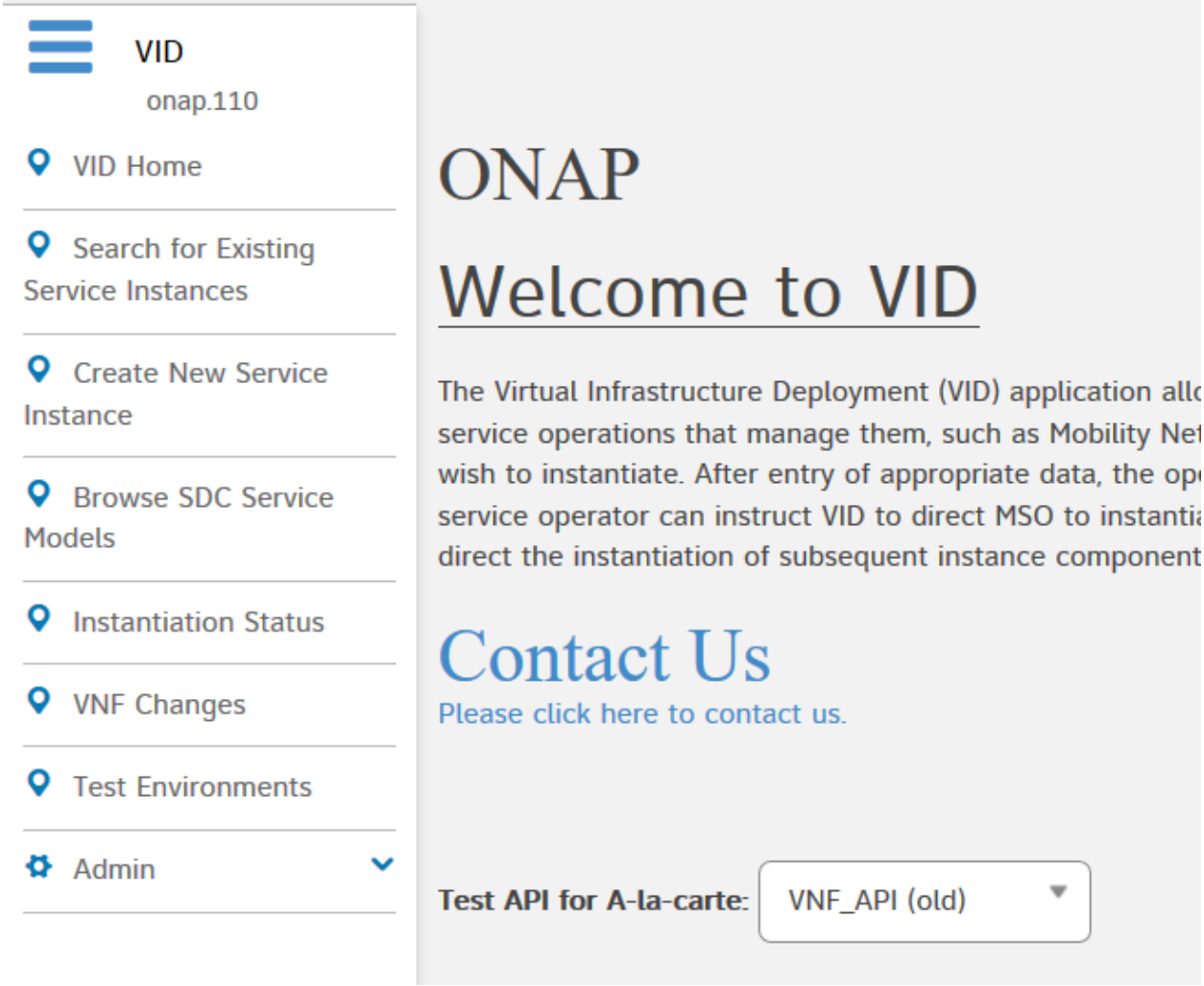
Phase 6: Service Instantiation hence CNF instantiation and validation

Once the service ‘cfwsvc1’ is distributed, you can instantiate it through VID application of ONAP portal.

Refer to https://docs.onap.org/en/elalto/submodules/vid.git/docs/instantiate.html for detailed instruction

Step 1, Create service instance

Create service instance ‘cfw1’ from service model ‘cfwsvc1’, make sure you select “VNF_API(old)” from the VID index page,



Then browse Service Models, select “cfwsvc1”, click “deploy” button, assign name with ‘cfw1’

Step 2, Add node instance ‘cfw1vf1’ for service instance ‘cfw1’

Add node instance ‘cfw1vf1’ for service instance ‘cfw1’, click the menu “add node instance” to add generic vnf: ’cfw1vf1’: input the generic vnf name, select the cloud region: WRCP2_STXRegion, and the tenant ‘onap-sb-01’, then click confirm button , wait for its completion

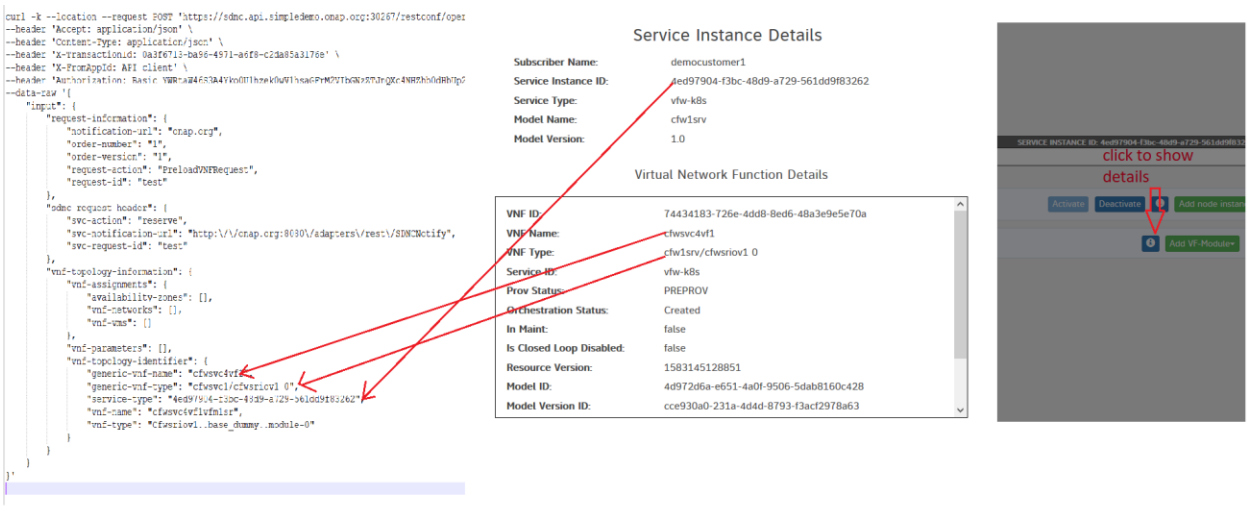
Step 3, Preload VNF topology

Preload VNF topology for VF Module ‘cfw1vf1vfmsriov1’ via curl command or postman

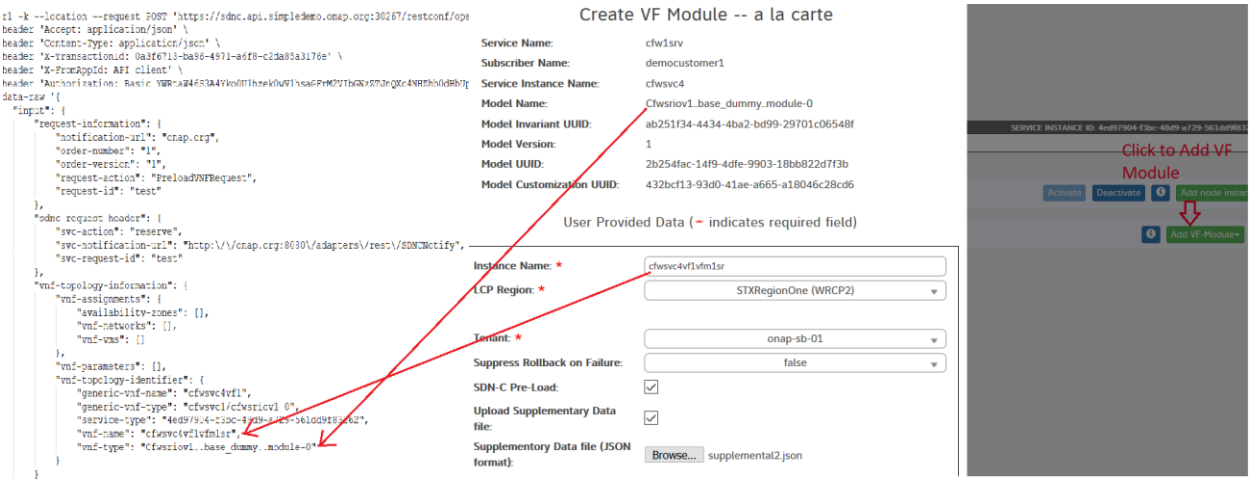
```
$ curl -k --location --request POST
'https://sdnc.api.simpledemo.onap.org:30267/restconf/operations/VNF-API:preload-vnf-topology-operation'
\
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--header 'X-TransactionId: 0a3f6713-ba96-4971-a6f8-c2da85a3176e' \
--header 'X-FromAppId: API client' \
--header 'Authorization: Basic
YWRtaW46S3A4Yko0U1hzek0wV1hsaGFrM2VIbGNzZTJnQXc4NHZhb0dHbUp2VXkyVQ==' \
--data-raw '{
  "input": {
    "request-information": {
      "notification-url": "onap.org",
      "order-number": "1",
      "order-version": "1",
      "request-action": "PreloadVNFRequest",
      "request-id": "test"
    },
    "sdnc-request-header": {
      "svc-action": "reserve",
      "svc-notification-url": "http:\/\/onap.org:8080\/adapters\/rest\/SDNCNotify",
      "svc-request-id": "test"
```

```
    },
    "vnf-topology-information": {
      "vnf-assignments": {
        "availability-zones": [],
        "vnf-networks": [],
        "vnf-vms": []
      },
      "vnf-parameters": [],
      "vnf-topology-identifier": {
        "generic-vnf-name": "cfw1vf1",
        "generic-vnf-type": "cfwsvc1/cfwsriov1 0",
        "service-type": "4ed97904-f3bc-48d9-a729-561dd9f83262",
        "vnf-name": " cfw1vf1vfmsriov1'",
        "vnf-type": "Cfwsriov1..base_dummy..module-0"
      }
    }
  }
}'
```

Check the following example about how to populate the VNF topology request body:



Snapshot 3: populate VNF Topology request data, generic VNF and service type



Snapshot 4: populate VNF Topology request data, VF module part

Note, do not click “Confirm” button until runs to step 5 below.

Step 4, prepare supplement data file

Prepare override_values.yaml which override the helm charts’ values, encode the file content and put it into supplemental1.json for VF Module ‘cfw1vf1vfmsriov1’

```
$ cat << EOF > override_values.yaml

global:
```

```
 unprotectedNetProviderVlan: 29
 protectedNetProviderVlan: 30
 nodeAffinity:
  - label:
     labelkey: sriovdp
     op: In
     labelvalues:
      - enabled
  - label:
     labelkey:  kube-cpu-mgr-policy
     op: In
     labelvalues:
      - static
EOF

$ OVERRIDE_VALUES_YAML_BASE64=`cat override_values.yaml | base64 -w 0`

$ cat <<EOF > supplemental1.json
[
 {
  "name": "definition-name",
  "value": "Cfwsriov1..base_dummy..module-0"
 },
 {
  "name": "definition-version",
  "value": "1"
 },
```

```
 {
  "name": "profile-name",
  "value": "p1"
 },
 {
  "name": "template_type",
  "value": "heat"
 },
 {
  "name": "override_values_yaml_base64",
  "value": "$OVERRIDE_VALUES_YAML_BASE64"
 }
]
EOF
```

Step 5, Add VF Module

Add VF Module 'cfw1vf1vfmsriov1' for node instance 'cfw1vf1'

From the popup dialog window, input the VF module name, select the cloud region: WRCP2_STXRegion, and the tenant 'onap-sb-01', check on the following options:

- SDN-C Pre-Lload
- Upload Supplementary Data file

Then upload the file 'supplemental1.json', click "confirm" button to start the VF module creation process and wait for its completion. This process will orchestrate cFW helm charts to WRCP instance. Hence you can check if the workload is deployed over WRCP instance with kubectl commands as well.

Step 6: validate the CNF deployment

Execute kubectl to check if deployments are there, check the pods status

```
$ kubectl get deployments -o wide --all-namespaces |grep firewall
```

```
$ kubectl get pods -o wide --all-namespaces |grep firewall
```

Monitor the traffic over SINK pod via browser: http://$NODE_IP:30667/ , you should observe the traffic statistics from the page. Refer to snapshot 1 for details.

Phase 7: Service deletion hence CNF deletion

With VID application GUI, Navigate to the deployed service instance ‘cfw1’ , delete VF module first, then delete node instance, then delete service.

Attachments

Attachment 1: integration-override.yaml

```
global:
  repository: nexus3.onap.org:10001
  pullPolicy: IfNotPresent
robot:
  enabled: true
  flavor: large
  appcUsername: "appc@appc.onap.org"
  appcPassword: "demo123456!"
  openStackKeyStoneUrl: "<keystone endpoint, e.g. http://10.12.25.2:5000>"
  openStackPublicNetId: "<tenant network UUID for public access,e.g. 971040b2-7059-49dc-b220-4fab50cb2ad4>"
  openStackTenantId: "<tenant ID of openstack user, e.g. 0e148b76ee8c42f78d37013bf6b7b1ae>"
  openStackUserName: "<tenant username, e.g. demo>"
  openStackUserDomain: "Default"
  openStackProjectName: "VIM"
  ubuntu14Image: "<glance image name for ubuntu14, e.g. ubuntu-14-04-cloud-amd64>"
  ubuntu16Image: "<glance image name for ubuntu16, e.g. ubuntu-16-04-cloud-amd64>"
  openStackPrivateNetId: "eda70926-a53f-458c-a621-40a64e72643d"
  openStackPrivateSubnetId: "4ef0889a-406f-488b-934e-52b3ad6aef3a"
  openStackPrivateNetCidr: "10.0.0.0/16"
  openStackSecurityGroup: "aa534410-959e-4c40-9480-b3ae5ec1d8d8"
  openStackOamNetworkCidrPrefix: "10.0"
  dcaeCollectorIp: "10.12.6.149"
  kubernetesExternalIp: "10.12.6.149"
  vnfPubKey: "<public key for ssh access to vnf>"
```

```
  demoArtifactsVersion: "1.6.0-SNAPSHOT"
  demoArtifactsRepoUrl: "https://nexus.onap.org/content/repositories/releases"
  scriptVersion: "1.6.0-SNAPSHOT"
  nfsIpAddress: "10.12.6.253"
  config:
    openStackEncryptedPasswordHere: "bbaef6cd76625ab9eb60deedeae7dbb9"
    openStackSoEncryptedPassword: ""
so:
  enabled: true
  so-catalog-db-adapter:
    config:
      openStackUserName: "<tenant username, e.g. demo>"
      openStackKeyStoneUrl: "<keystone endpoint, e.g. http://10.12.25.2:5000>"
      openStackEncryptedPasswordHere: ""
  so-bpmn-infra:
    config:
appc:
  enabled: true
  replicaCount: 3
  config:
    enableClustering: true
    openStackType: "OpenStackProvider"
    openStackName: "OpenStack"
    openStackKeyStoneUrl: "<keystone endpoint, e.g. http://10.12.25.2:5000/v2.0>"
    openStackServiceTenantName: "VIM"
    openStackDomain: "Default"
    openStackUserName: "<tenant username, e.g. demo>"
```

```
    openStackEncryptedPassword: "<tenant user password>"
  mariadb-galera:
    liveness:
      initialDelaySeconds: 180
      periodSeconds: 60
sdnc:
  enabled: true
  replicaCount: 3
  config:
    enableClustering: true
  mariadb-galera:
    liveness:
      initialDelaySeconds: 180
      periodSeconds: 60
aai:
  enabled: true
  liveness:
    initialDelaySeconds: 120
  aai-data-router:
    liveness:
      initialDelaySeconds: 120
  aai-sparky-be:
    liveness:
      initialDelaySeconds: 120
  aai-spike:
    liveness:
      initialDelaySeconds: 120
```

```
  aai-cassandra:
    replicaCount: 3
    liveness:
      periodSeconds: 120
    readiness:
      periodSeconds: 60
portal:
  enabled: true
  portal-cassandra:
    liveness:
      periodSeconds: 120
    readiness:
      periodSeconds: 60
vid:
  enabled: true
aaf:
  enabled: true
cassandra:
  enabled: true
  liveness:
    periodSeconds: 120
  readiness:
    periodSeconds: 60
clamp:
  enabled: true
cli:
  enabled: true
```

```
consul:
  enabled: true
contrib:
  enabled: true
dcaegen2:
  enabled: false
dmaap:
  enabled: true
  dmaap-dr-prov:
    mariadb:
      liveness:
        initialDelaySeconds: 180
        periodSeconds: 60
esr:
  enabled: true
log:
  enabled: false
  log-logstash:
    replicaCount: 1
sniro-emulator:
  enabled: true
oof:
  enabled: true
  oof-has:
    music:
      music-cassandra:
        replicaCount: 3
```

```
      liveness:
        periodSeconds: 120
      readiness:
        periodSeconds: 60
    music-tomcat:
      replicaCount: 1
mariadb-galera:
  enabled: true
  liveness:
    initialDelaySeconds: 180
    periodSeconds: 60
modeling:
  enabled: true
  mariadb-galera:
    liveness:
      initialDelaySeconds: 180
      periodSeconds: 60
msb:
  enabled: true
multicloud:
  enabled: true
  image: onap/multicloud/framework:1.5.1
  multicloud-starlingx:
    image: onap/multicloud/openstack-starlingx:1.5.5
  multicloud-k8s:
    image: onap/multicloud/k8s:0.5.0
nbi:
```

```
  enabled: false
policy:
  enabled: true
pomba:
  enabled: false
sdc:
  enabled: true
  sdc-cs:
    liveness:
      periodSeconds: 120
    readiness:
      periodSeconds: 60
  sdc-be:
    liveness:
      initialDelaySeconds: 120
      periodSeconds: 120
      timeoutSeconds: 15
    readiness:
      initialDelaySeconds: 120
      periodSeconds: 120
      timeoutSeconds: 15
  sdc-fe:
    livenessProbe:
      initialDelaySeconds: 120
      periodSeconds: 120
      timeoutSeconds: 15
    readinessProbe:
```

```
    initialDelaySeconds: 120
    periodSeconds: 120
    timeoutSeconds: 15
uui:
  enabled: false
vfc:
  enabled: false
  mariadb-galera:
   liveness:
    initialDelaySeconds: 180
    periodSeconds: 60
vnfsdk:
  enabled: false
cds:
  enabled: true
```

Attachment 2: onap-oom-lite.env

```
parameters:

  ubuntu_1804_image: <glance image name for ubuntu16, e.g. ubuntu-18.04>

  apt_proxy: ""
  docker_proxy: nexus3.onap.org:10001

  nfs_vm_flavor: m1.lm.xlarge
  k8s_vm_flavor: m1.lm.xlarge
```

```
orch_vm_flavor: m1.lm.medium

public_net_id: <tenant network UUID for public access,e.g. 971040b2-7059-49dc-b220-4fab50cb2ad4>

oam_network_cidr: 10.0.0.0/16
oam_ext_network_cidr: 10.100.0.0/16

integration_gerrit_branch: master

helm_deploy_delay: 30s

integration_override_yaml: >
 global:
   repository: __docker_proxy__
   pullPolicy: IfNotPresent
 robot:
   enabled: true
   flavor: large
   appcUsername: "appc@appc.onap.org"
   appcPassword: "demo123456!"
   openStackKeyStoneUrl: "<keystone endpoint, e.g. http://10.12.25.2:5000>"
   openStackPublicNetId: "__public_net_id__"
   openStackTenantId: "${OS_PROJECT_ID}"
   openStackUserName: "${OS_USERNAME}"
   openStackUserDomain: "${OS_USER_DOMAIN_NAME}"
   openStackProjectName: "${OS_PROJECT_NAME}"
   ubuntu14Image: "<glance image name for ubuntu14, e.g. ubuntu-14-04-cloud-amd64>"
```

```
  ubuntu16Image: "<glance image name for ubuntu16, e.g. ubuntu-16-04-cloud-amd64>"
  openStackPrivateNetId: "__oam_network_id__"
  openStackPrivateSubnetId: "__oam_subnet_id__"
  openStackPrivateNetCidr: "__oam_network_cidr__"
  openStackSecurityGroup: "__sec_group__"
  openStackOamNetworkCidrPrefix: "10.0"
  dcaeCollectorIp: "__k8s_01_vm_ip__"
  kubernetesExternalIp: "__k8s_01_vm_ip__"
  vnfPubKey: "<public key for ssh access to vnf>"
  demoArtifactsVersion: "1.6.0-SNAPSHOT"
  demoArtifactsRepoUrl: "https://nexus.onap.org/content/repositories/releases"
  scriptVersion: "1.6.0-SNAPSHOT"
  nfsIpAddress: "__nfs_ip_addr__"
  config:
   openStackEncryptedPasswordHere: "${OS_PASSWORD_ENCRYPTED_FOR_ROBOT}"
   openStackSoEncryptedPassword: "${OS_PASSWORD_ENCRYPTED}"
so:
  enabled: true
  so-catalog-db-adapter:
   config:
    openStackUserName: "${OS_USERNAME}"
    openStackKeyStoneUrl: "<keystone endpoint, e.g. http://10.12.25.2:5000/v2.0>"
    openStackEncryptedPasswordHere: "${OS_PASSWORD_ENCRYPTED}"
appc:
  enabled: true
  replicaCount: 3
  config:
```

```
    enableClustering: true
    openStackType: "OpenStackProvider"
    openStackName: "OpenStack"
    openStackKeyStoneUrl: "<keystone endpoint, e.g. http://10.12.25.2:5000/v2.0>"
    openStackServiceTenantName: "${OS_PROJECT_NAME}"
    openStackDomain: "${OS_USER_DOMAIN_NAME}"
    openStackUserName: "${OS_USERNAME}"
    openStackEncryptedPassword: "${OS_PASSWORD}"
  mariadb-galera:
    liveness:
      initialDelaySeconds: 180
      periodSeconds: 60
sdnc:
  enabled: true
  replicaCount: 3
  config:
    enableClustering: true
  mariadb-galera:
    liveness:
      initialDelaySeconds: 180
      periodSeconds: 60
aai:
  enabled: true
  liveness:
    initialDelaySeconds: 120
  aai-data-router:
    liveness:
```

```
    initialDelaySeconds: 120
  aai-sparky-be:
   liveness:
    initialDelaySeconds: 120
  aai-spike:
   liveness:
    initialDelaySeconds: 120
  aai-cassandra:
   replicaCount: 3
   liveness:
    periodSeconds: 120
   readiness:
    periodSeconds: 60
portal:
  enabled: true
  portal-cassandra:
   liveness:
    periodSeconds: 120
   readiness:
    periodSeconds: 60
vid:
  enabled: true
aaf:
  enabled: true
cassandra:
  enabled: true
  liveness:
```

```
    periodSeconds: 120
  readiness:
    periodSeconds: 60
clamp:
  enabled: true
cli:
  enabled: true
consul:
  enabled: true
contrib:
  enabled: true
dcaegen2:
  enabled: false
dmaap:
  enabled: true
  dmaap-dr-prov:
    mariadb:
      liveness:
        initialDelaySeconds: 180
        periodSeconds: 60
esr:
  enabled: true
log:
  enabled: false
  log-logstash:
    replicaCount: 1
sniro-emulator:
```

```
  enabled: true
oof:
  enabled: true
  oof-has:
    music:
      music-cassandra:
        replicaCount: 3
        liveness:
          periodSeconds: 120
        readiness:
          periodSeconds: 60
      music-tomcat:
        replicaCount: 1
mariadb-galera:
  enabled: true
  liveness:
    initialDelaySeconds: 180
    periodSeconds: 60
modeling:
  enabled: true
  mariadb-galera:
    liveness:
      initialDelaySeconds: 180
      periodSeconds: 60
msb:
  enabled: true
multicloud:
```

```
  enabled: true
  image: onap/multicloud/framework:1.5.1
  multicloud-starlingx:
    image: onap/multicloud/openstack-starlingx:1.5.5
  multicloud-k8s:
    image: onap/multicloud/k8s:0.5.0
nbi:
  enabled: false
policy:
  enabled: true
pomba:
  enabled: false
sdc:
  enabled: true
  sdc-cs:
    liveness:
      periodSeconds: 120
    readiness:
      periodSeconds: 60
  sdc-be:
    liveness:
      initialDelaySeconds: 120
      periodSeconds: 120
      timeoutSeconds: 15
    readiness:
      initialDelaySeconds: 120
      periodSeconds: 120
```

```
      timeoutSeconds: 15
    sdc-fe:
     livenessProbe:
      initialDelaySeconds: 120
      periodSeconds: 120
      timeoutSeconds: 15
     readinessProbe:
      initialDelaySeconds: 120
      periodSeconds: 120
      timeoutSeconds: 15
  uui:
   enabled: false
  vfc:
   enabled: false
   mariadb-galera:
    liveness:
      initialDelaySeconds: 180
      periodSeconds: 60
  vnfsdk:
   enabled: false
  cds:
   enabled: true
```

Attachment 3: base_dummy.yaml

```
# #==================LICENSE_START=========================================
```

```
heat_template_version: 2016-10-14
description: Heat template to deploy dummy VNF

parameters:
 dummy_name_0:
   type: string
   label: name of vm
   description: Dummy name

 vnf_id:
   type: string
   label: id of vnf
   description: Provided by ONAP

 vnf_name:
   type: string
   label: name of vnf
   description: Provided by ONAP

 vf_module_id:
   type: string
```

```
  label: vnf module id
  description: Provided by ONAP

 dummy_image_name:
  type: string
  label: Image name or ID
  description: Dummy image name

 dummy_flavor_name:
  type: string
  label: flavor
  description: Dummy flavor

resources:
 dummy_0:
  type: OS::Nova::Server
  properties:
   name: { get_param: dummy_name_0 }
   image: { get_param: dummy_image_name }
   flavor: { get_param: dummy_flavor_name }
   metadata: { vnf_name: { get_param: vnf_name }, vnf_id: { get_param: vnf_id }, vf_module_id:
{ get_param: vf_module_id }}
```

Attachment 4: base_dummy.env

```
parameters:
```

vnf_id: PROVIDED_BY_ONAP

vnf_name: PROVIDED_BY_ONAP

vf_module_id: PROVIDED_BY_ONAP

dummy_name_0: dummy_1_0

dummy_image_name: dummy

dummy_flavor_name: dummy.default