Deploying vFw and EdgeXFoundry Services on Kubernets Cluster with ONAP

vFW Helm Chart link:

https://github.com/onap/multicloud-k8s/tree/master/kud/demo/firewall

EdgeXFoundry Helm Chart link:

https://github.com/onap/multicloud-k8s/tree/master/kud/tests/vnfs/edgex/helm/edgex

Create CSAR with Helm chart as an artifact

The CSAR is a heat template package with Helm chart in it. Basic package consists of an environment file, base_dummy.yaml file (for the sake of example) and *MANIFEST.json* and the *tar.gz* file (of Helm chart). We need to zip all of these files before onboarding.

One thing to pay much attention to is the naming convention which must be followed while making the tgz.

NOTE: The Naming convention is for the helm chart tgz file.

Naming convention follows the format:

<free format string>***_cloudtech_***<technology>_<subtype>.extension

1. *Cloudtech* is a fixed pattern and should not be changed if not necessary
2. *Technology:* k8s, azure, aws, ….
3. *Subtype*: charts, day0, configtemplate, ...
4. *Extension*: zip, tgz, csar, …"

NOTE: The .tgz file must be a tgz created from the top level helm chart folder. I.e a folder that contains a Chart.yaml file in it. For vFW use case the content of the tgz file must be following

```
$ tar -czvf vfw_cloudtech_k8s_charts.tgz firewall/

$ tar -tf vfw_cloudtech_k8s_charts.tgz

firewall/.helmignore
firewall/Chart.yaml
firewall/templates/onap-private-net.yaml
firewall/templates/_helpers.tpl
firewall/templates/protected-private-net.yaml
firewall/templates/deployment.yaml
firewall/templates/unprotected-private-net.yaml
firewall/values.yaml
firewall/charts/sink/.helmignore
firewall/charts/sink/Chart.yaml
firewall/charts/sink/templates/configmap.yaml
firewall/charts/sink/templates/_helpers.tpl
firewall/charts/sink/templates/service.yaml
firewall/charts/sink/templates/deployment.yaml
firewall/charts/sink/values.yaml
firewall/charts/packetgen/.helmignore
firewall/charts/packetgen/Chart.yaml
firewall/charts/packetgen/templates/_helpers.tpl
firewall/charts/packetgen/templates/deployment.yaml
firewall/charts/packetgen/values.yaml
```

Listed below is an example of the contents inside a heat template package

vfw_k8s_demo.zip file is a zip of the 4 other files(*manifest.json, base_dummy.env, base_dummy.yaml, vfw_cloudtech_k8s_charts.tgz*) gets on boarded onto SDC.

```
$ vfw-k8s/package$ ls
MANIFEST.json  base_dummy.env base_dummy.yaml  vfw_cloudtech_k8s_charts.tgz vfw_k8s_demo.zip
```

MANIFEST.json

Key thing is note the addition of cloud artifact

type: "CLOUD_TECHNOLOGY_SPECIFIC_ARTIFACTS"

```
{
  "name": "",
  "description": "",
  "data": [
    {
      "file": "base_dummy.yaml",
      "type": "HEAT",
      "isBase": "true",
      "data": [
        {
          "file": "base_dummy.env",
          "type": "HEAT_ENV"
        }
      ]
    },
    {
      "file": "vfw_cloudtech_k8s_charts.tgz",
      "type": "CLOUD_TECHNOLOGY_SPECIFIC_ARTIFACTS"
    }
]
}
```

base_dummy.yaml

Designed to be minimal HEAT template.

```
# #==================LICENSE_START=========================================
# #
# # Copyright (C) 2019 Intel Corporation
# # SPDX-License-Identifier: Apache-2.0
# #
# #==================LICENSE_END===========================================

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 }}
```

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
```

Onboard the CSAR

For onboarding instructions please refer to steps 4-9 from the document here.

Distribute the CSAR

On onboarding, a service gets stored in SDC and as a final action, it is distributed to SO and other services. When distribution happens it takes tar.gz file and uploads to k8s plugin.

Steps for installing KUD Cloud

Follow the link to install KUD Kubernetes Deployment. KUD contains all the packages required for running vFw Usecase.

REGISTER KUD CLOUD REGION with K8s-Plugin

API to support Reachability for Kubernetes Cloud

The command to POST Connectivity Info

```
{
   "cloud-region" : "<name>",   // Must be unique across
   "cloud-owner" :  "<owner>",
   "other-connectivity-list" : {
         }

}
```

This is a multipart upload and here is how you do the POST for this.

#Using a json file (eg: post.json) containing content as above;

```
curl -i -F "metadata=<post.json;type=application/json>" -F file=@

/home/ad_kkkamine/.kube/config -X POST http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/connectivity-info
```

Command to GET Connectivity Info

```
curl -i -X GET http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/connectivity-info/{name}
```

Command to DELETE Connectivity Info

```
curl -i -X DELETE http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/connectivity-info/{name}
```

Command to UPDATE/PUT Connectivity Info

```
curl -i -d @update.json -X PUT http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/connectivity-info
```

Update K8sConfig

Workaround for R4 Dublin. This step will not be needed from R5.

Edit the configMap **helm-release-name-multicloud-k8s** for K8s plugin to make changes to the config like below to add **ovn-central-address:**

```
{
"ca-file": "/opt/multicloud/k8splugin/certs/root_ca.cer",
"server-cert": "/opt/multicloud/k8splugin/certs/multicloud-k8s.pub",
"server-key": "/opt/multicloud/k8splugin/certs/multicloud-k8s.pr",
"password": "c2VjcmV0bWFuYWdlbWVudHNlcnZpY2VzZWNyZXRwYXNzd29yZA==",
"database-type": "mongo",
"database-address": "multicloud-k8s-mongo",
"etcd-ip": "multicloud-k8s-etcd",
"plugin-dir": "/opt/multicloud/k8splugin/plugins",

 "ovn-central-address": "<IP address of the Kubernetes controller>:6641"
}
```

(the configMap is based on **oom/kubernetes/multicloud/charts/multicloud-k8s/resources/config/k8sconfig.json)**

Restart the Multoclod-K8s Plugin for the changes to take effect.

Register KUD Cloud region with AAI

With k8s cloud region, we need to add a tenant to the k8s cloud region. The 'easy' way is to have the ESR information (in step 1 of cloud registration) point to a real OpenStack tenant (e.g. the OOF tenant in the lab where we tested).

This will cause multicloud to add the tenant to the k8s cloud region and then, similar to #10 in the documentation here , the service-subscription can be added to that object.

NOTE: use same name cloud-region and cloud-owner name

An example is shown below for K8s cloud but following the steps 1,2,3 from here. The sample input below is for k8s cloud type.

Step 1 - Cloud Registration/ Create a cloud region to represent the instance.

Note: highlighted part of the body refers to an existing OpenStack tenant (OOF in this case). Has nothing to do with the K8s cloud region we are adding.

```
PUT https://{{AAI1_PUB_IP}}:{{AAI1_PUB_PORT}}/aai/v13/cloud-infrastructure/cloud-regions/cloud-region/k8scloudowner4/k8sregionfour
{
"cloud-owner": "k8scloudowner4",
"cloud-region-id": "k8sregionfour",
"cloud-type": "k8s",
"owner-defined-type": "t1",
"cloud-region-version": "1.0",
"complex-name": "clli1",
"cloud-zone": "CloudZone",
"sriov-automation": false,
  "cloud-extra-info":"{\"openstack-region-id\":\"k8sregionthree\"}",
"esr-system-info-list": {
        "esr-system-info": [
                {
                        "esr-system-info-id": "55f97d59-6cc3-49df-8e69-926565f00066",
                        "service-url": "http://10.12.25.2:5000/v3",
                        "user-name": "demo",
                        "password": "onapdemo",
                        "system-type": "VIM",
                        "ssl-insecure": true,
                        "cloud-domain": "Default",
                        "default-tenant": "OOF",
                        "tenant-id": "6bbd2981b210461dbc8fe846df1a7808",
                        "system-status": "active"
                       }
                ]
}
}
```

Step 2 – add a complex to the cloud

Note: just adding one that exists already

```
PUT https://{{AAI1_PUB_IP}}:{{AAI1_PUB_PORT}}/aai/v13/cloud-infrastructure/cloud-regions/cloud-region/k8scloudowner4/k8sregionfour/relationship-list/relationship
{
"related-to": "complex",
"related-link": "/aai/v13/cloud-infrastructure/complexes/complex/clli1",
"relationship-data": [
  {
    "relationship-key": "complex.physical-location-id",
    "relationship-value": "clli1"
  }
]
}
```

Step 3 - Trigger the Muticloud plugin registration process

```
POST http://{{MSB_IP}}:{{MSB_PORT}}/api/multicloud-titaniumcloud/v1/k8scloudowner4/k8sregionfour/registry
```

This registers the K8S cloud with Multicloud – it also reaches out and adds tenant information to the cloud (see example below – you'll see all kinds of flavor, image information that is associated with the OOF tenant).

If we had not done it this way, then we'd have to go in to AAI at this point and manually add a tenant to the cloud region. The first time I tried this (k8s region one), I just made up some random tenant id and put it in.)

The tenant is there so you can add the service-subscription to it:

Making a Service Type:

```
PUT https://{{AAI1_PUB_IP}}:{{AAI1_PUB_PORT}}/aai/v13/service-design-and-creation/services/service/vfw-k8s
{
        "service-description": "vfw-k8s",
        "service-id": "vfw-k8s"
}
```

Add subscription to service type to the customer (Demonstration in this case – which was already created by running the robot demo scripts)

```
PUT https://{{AAI1_PUB_IP}}:{{AAI1_PUB_PORT}}/aai/v16/business/customers/customer/Demonstration/service-subscriptions/service-subscription/vfw-k8s
{
        "service-type": "vfw-k8s"
}
```

Add Service-Subscription to the tenant (resource-version changes based on actual value at the time):

```
PUT https://{{AAI1_PUB_IP}}:{{AAI1_PUB_PORT}}/aai/v16/cloud-infrastructure/cloud-regions/cloud-region/k8scloudowner4/k8sregionfour/tenants/tenant/6bbd2981b210461dbc8fe846df1a7808?resource-version=1559345527327
{
"tenant-id": "6bbd2981b210461dbc8fe846df1a7808",
"tenant-name": "OOF",
"resource-version": "1559345527327",
"relationship-list": {
      "relationship": [
         {
            "related-to": "service-subscription",
            "relationship-label": "org.onap.relationships.inventory.Uses",
            "related-link": "/aai/v13/business/customers/customer/Demonstration/service-subscriptions/service-subscription/vfw-k8s",
            "relationship-data": [
               {
                  "relationship-key": "customer.global-customer-id",
                  "relationship-value": "Demonstration"
               },
               {
                  "relationship-key": "service-subscription.service-type",
                  "relationship-value": "vfw-k8s"
               }
            ]
         }
   ]
}
}
```

4. Addition of Cloud Region to SO Catalog DB

See starting around 4:45 of the video - https://wiki.onap.org/download/attachments/64006768/vfwk8s_cloud_registration_720.mp4

Additional SO Configuration

There is a configuration needed for SO – it is described here in the docs (in step 4): https://onap.readthedocs.io/en/latest/submodules/integration.git/docs/docs_vfwHPA.html#docs-vfw-hpa

But also replicated below for convenience:

Modify the SO bpmn configmap to change the SO vnf adapter endpoint to v2

oom-rancher# kubectl -n onap edit configmap dev-so-so-bpmn-infra-app-configmap

- vnf:

    endpoint: http://so-openstack-adapter.onap:8087/services/VnfAdapter

    rest:

      endpoint: http://so-openstack-adapter.onap:8087/services/rest/v1/vnfs

+ vnf:

    endpoint: http://so-openstack-adapter.onap:8087/services/VnfAdapter

    rest:

      endpoint: http://so-openstack-adapter.onap:8087/services/rest/v2/vnfs

Then delete the bpmn pod

oom-rancher# kubectl delete <pod-name> -n onap

Create Profile Manually

K8s-plugin artifacts start in the form of Definitions. These are nothing but Helm Charts wrapped with some metadata about the chart itself. Once the Definitions are created, we are ready to create some profiles so that we can customize that definition and instantiate it in Kubernetes.

(NOTE: Refer this link for complete API lists and documentation: MultiCloud K8s-Plugin-service API)

A profile contains the following:

1. **manifest.yaml**
    a. Contains the details for the profile and everything contained within
2. A **HELM** values override yaml file.
    a. It can have any name as long as it matches the corresponding entry in the **manifest.yaml**
3. Any number of files organized in a folder structure
    a. All these files should have a corresponding entry in **manifest.yaml** file

Creating a Profile Artifact

```
> cd multicloud-k8s/kud/tests/vnfs/testrb/helm/profile
> find .
manifest.yaml
override_values.yaml
testfol
testfol/subdir
testfol/subdir/deployment.yaml

#Create profile tar.gz
> cd profile
> tar -cf profile.tar *
> gzip profile.tar
> mv profile.tar.gz ../
```

The manifest file contains the following

```
---
version: v1
type:
 values: "values_override.yaml"
 configresource:
   - filepath: testfol/subdir/deployment.yaml
     chartpath: vault-consul-dev/templates/deployment.yaml
```

Note: values: "values_override.yaml" can **be** empty **file if** you are creating **a** dummy **profile**

Note: A dummy profile does not need any customization. The following is optional in the manifest file.

```
configresource:
   - filepath: testfol/subdir/deployment.yaml
     chartpath: vault-consul-dev/templates/deployment.yaml
```

We need to read the name of the Definition which was created while distribution of the service from SDC.

Command to read the Definition name and its version

On the ONAP K8s Rancher host execute following statement

```
kubectl logs -n onap `kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}' | grep multicloud-k8s | head -1` -c multicloud-k8s
```

From the output read the name of the definition which is "rb-name" and "rb-version" respectively

```
127.0.0.1 - - [15/Jul/2019:07:56:21 +0000] "POST /v1/rb/definition/test-rbdef/1/content HTTP/1.1"
```

Command to read (GET) Definition

With this information, we are ready to upload the profile with the following JSON data

```
{
  "rb-name": "test-rbdef",
  "rb-version": "1",
  "profile-name": "p1",
  "release-name": "r1", //If release-name is not provided, profile-name will be used
  "namespace": "testnamespace1",
  "kubernetes-version": "1.15.3"
}
```

Command to create (POST) Profile

```
curl -i -d @create_rbprofile.json -X POST http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/rb/definition/test-rbdef/1/profile
```

Command to UPLOAD artifact for Profile

```
curl -i --data-binary @profile.tar.gz -X POST http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/rb/definition/test-rbdef/1/profile/p1/content
```

Command to GET Profiles

```
# Get all Profiles
curl -i http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/rb/definition/test-rbdef/1/profile
# Get one Profile
curl -i http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/rb/definition/test-rbdef/1/profile/p1
```

Command to DELETE Profile

```
curl -i -X DELETE http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/rb/definition/test-rbdef/1/profile/p1
```

Instantiation

Instantiation is done by SO. SO then talks to Multi Cloud-broker via MSB and that in turn looks up the cloud region in AAI to find the endpoint. If k8sregion one is registered with AAI and SO makes a call with that, then the broker will know that it needs to talk to k8s-plugin based on the type of the registration.

This video shows the whole sequence of instantiation using VID:

https://wiki.onap.org/download/attachments/64006768/vfwk8s_deploy_delete_720.mp4

Create User parameters

In the VID user parameters are created in the following format during vfModule creation:

```
[/tmp/dublin-demos/demo-vfw-k8s/preload]$ cat supplemental.json | jq .
[
  {
    "name": "definition-name",
    "value": "VfwK8s..base_dummy..module-0"
  },
  {
    "name": "definition-version",
    "value": "1"
  },
  {
    "name": "profile-name",
    "value": "profile1"
  },
  {
    "name": "template_type",
    "value": "heat"
  }
]
```

Known Issues:

- Use the vFw Helm chart from the Master branch - (https://github.com/onap/multicloud-k8s/tree/master/kud/demo/firewall)
- Artifact broker version issue (Libo to update this section)
    - please refer to the description in ~~MULTICLOUD-749~~ - Document in wiki workaround needed in Dublin release CLOSED
- Recommended way tar the resource bundle image is tar.gz. helm package is not supported in Dublin.
- In Dublin Get all Definitions is not supported in K8s Plugin API's. Bug is filed for this.

Commands for instantiation using K8s API:

These commands can be used to interact directly with K8s Plugin with SO.

Command to Instantiate a Profile

```
curl -d @create_rbinstance.json http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/instance
```

Delete Instantiated Kubernetes resources

The **id** field from the returned JSON can be used to **DELETE** the resources created in the previous step. This executes a Delete operation using the Kubernetes API.

```
curl -X DELETE http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/instance/ZKMTSaxv
```

GET Instantiated Kubernetes resources

The **id field** from the returned JSON can be used to **GET** the resources created in the previous step. This executes a get operation using the Kubernetes API.

```
curl -X GET http://MSB_NODE_IP:30280/api/multicloud-k8s/v1/v1/instance/ZKMTSaxv
```