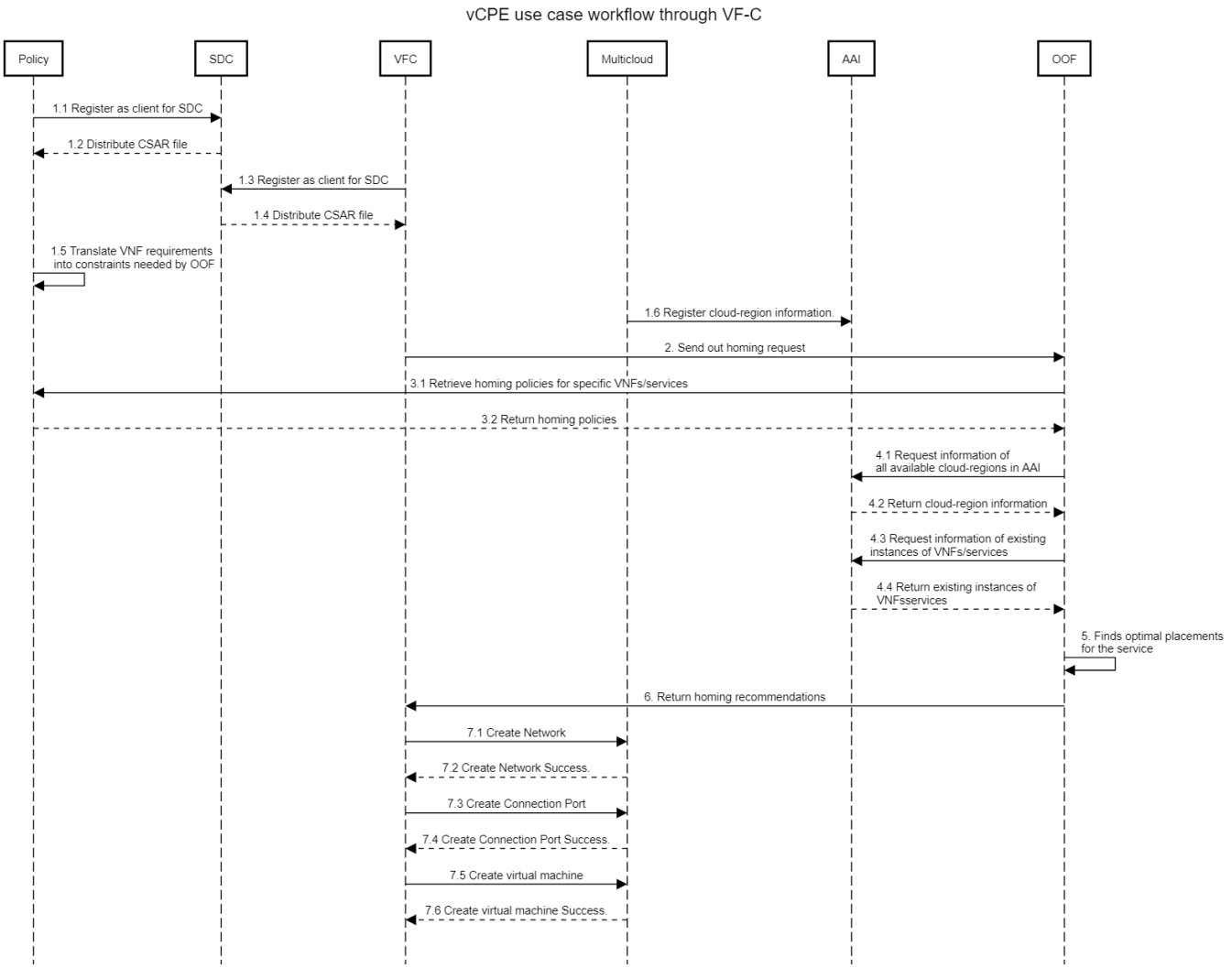
VF-C Casablanca HPA Design

DRAFT

This is a work in progress. Comments and suggestions gladly accepted. Draft will be removed once this is finalized.

HPA VF-C External Sequence Flow for Casablanca

vCPE use case workflow through VF-C



HPA VF-C External API Interaction for Casablanca

SDC

Supported HPA Capability Requirements(DRAFT)#LogicalNodei/ORequirements is referred.

SDC will provide below HPA information to VF-C

Logical Node i/O Requirements

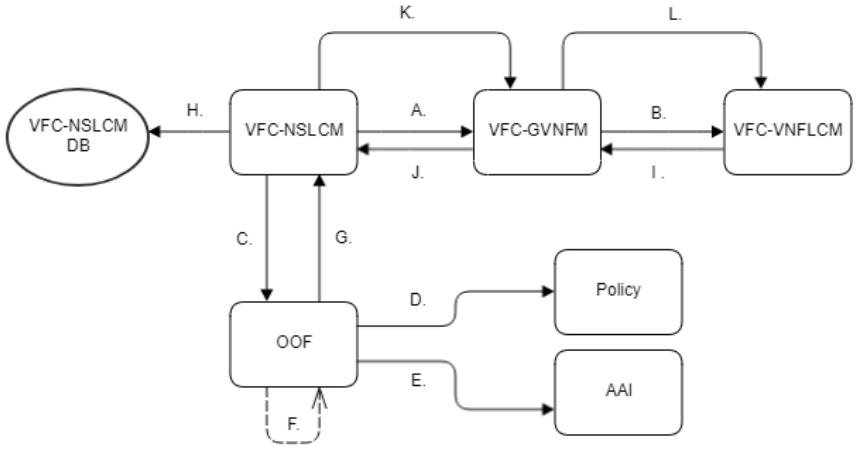
Capability Name	Capability Value	Descriptiopn
Capability Name	**Capability Value**	**Descriptiopn**
pciVendorId		PCI-SIG vendor ID for the device
pciDeviceId		PCI-SIG device ID for the device
pciNumDevices		Number of PCI devices required.

pciAddress		Geographic location of the PCI device via the standard PCI-SIG addressing model of Domain:Bus:device:function
pciDeviceLocalToNumaNode	required notRequired	Determines if I/O device affinity is required.

Network Interface Requirements

Capability Name	Capability Value	Description
nicFeature	LSO, LRO, RSS, RDMA	Long list of NIC related items such as LSO, LRO, RSS, RDMA, etc.
dataProcessingAccelerationLibray	Dpdk_Version	Name and version of the data processing acceleration library required. Orchestration can match any NIC that is known to be compatible with the specified library.
interfaceType	Virtio, PCI-Passthrough, **SR-IOV**, E1000, RTL8139, PCNET	Network interface type
vendorSpecificNicFeature	TBA	List of vendor specific NIC related items.

OOF



Note: Step I, J, K, L are all sub steps of Step A and B

```
A. VFC-NSLCM calls VFC-GVNFM driver
through RESTful api to create VNF.

B. VFC-GVNFM invokes VFC-VNFLCM through
RESTful api to do the

creation(which requires bunch of time)
and returns the VNF information(Almost
synchronous).

( I. VFC-LCM calls VFC-GVNFM to apply
for grant.

  J. VFC-GVNFM driver invokes VFC-NSLCM
to provide the grant info.

  K. VFC-NSLCM provides grant info
along with placement from OOF to VFC-
GVNFM driver
```

```
  L. VFC-GVNFM transmits the
information to VFC-VNFLCM to go through

     the following steps for
instantiation.)

During the creation period(Step B.):

C. VFC-NSLCM calls OOF for homing
allocation service using the VNF
information returned.

D. OOF calls POLICY to retrieve related
policies for such service/VNF.

E. OOF calls AAI to get all available
cloud regions or existing instances
inside the database.

F. OOF processes the homing allocation.

G. OOF respond to VFC-NSLCM with the
optimal placement.

H. VFC-NSLCM stores the data returned
by OOF into its database.
```

The integration between OOF and VFC can be divided into 4 phases:

- VFC send out homing request to OOF(Step C.)
- OOF retrieve the requirements(policies) for that service/VNF inside Policy(Step D.)
- OOF check AAI database for existing instances/available cloud regions(Step E.)
- OOF process homing allocation and return homing placement to VFC (Step F. and G.)

VFC send out homing request to OOF

A module inside NSLCM will call OOF to get homing allocation placement. And it will use the same API using by SO. Sample request using these API:

VFC-OOF API

```
{
  "requestInfo": {
    "transactionId": "2441780a-2710-4169-91ee-a9f52a705bb3 //UUID",
    "requestId": "2441780a-2710-4169-91ee-a9f52a705bb3 //UUID",
    "callbackUrl": "<callback URL from VFC>",
    "sourceId": "vfc",
    "requestType": "create",
    "numSolutions": 1,
    "optimizers": [
      "placement"
    ],
    "timeout": 600
  },
  "placementInfo": {
    "placementDemands": [
      {
        "resourceModuleName": "vBRG //<VNF name from CSAR>",
        "serviceResourceId": "<vnfInstanceId used inside VFC>",
        "resourceModelInfo": {
          "modelInvariantId": "no-resourceModelInvariantId",
          "modelVersionId": "no-resourceModelVersionId"
        }
      }
    ]
  },
  "serviceInfo": {
    "serviceInstanceId": "9fd24064-a335-478c-bbb0-3b71b7fbc55f",
    "serviceName": "vcpe",
    "modelInfo": {
      "modelInvariantId": "31390ef2-94a9-4cef-a09a-08f7d66540c9 //Id get from CSAR",
      "modelVersionId": "1a32426f-3616-47d0-96eb-b20cc7fff9be //Id get from CSAR"
    }
  }
}
```

OOF retrieve the requirements(policies) for that service/VNF inside Policy

Based on the request from VF-C, OOF need to retrieve policies for that VNF/service. Sample Policy schema will look like this:

Note: Since flavor_label and sriov_nic_label are not needed by VFC, the values inside directives will be leave as blank or default values.

OOF retrieve HPA policies for VFC

```
#
#Example 1: vCPE, Basic Capability and sriovNICNetwork
#one VNFC(VFC) with one basic capability requirement and two sriovNICNetwork requirements
#
{
  "service": "hpaPolicy",
  "policyName": "oofCasablanca.hpaPolicy_vFW",
  "description": "HPA policy for vFW",
  "templateVersion": "0.0.1",
  "version": "1.0",
  "priority": "3",
  "riskType": "test",
  "riskLevel": "2",
  "guard": "False",
  "content": {
    "resources": "vG",
    "identity": "hpaPolicy_vG",
    "policyScope": ["vCPE", "US", "INTERNATIONAL", "ip", "vG"],
    "policyType": "hpaPolicy",
    "flavorFeatures": [
```

```
      {
        "id" : "<vdu.Name>",
        "type":"vnfc/tocsa.nodes.nfv.Vdu.Compute",
        "directives":[
           {
             "directive_name":"flavor_directive",
             "attributes":[
                {
                  "attribute_name":"<Blank, or use Default value String 'flavor_name'>",
                  "attribute_value": "<Blank>"
                }
               ]
           }
          ],
        "flavorProperties": [
          {
            "hpa-feature": "basicCapabilities",
            "mandatory": "True",
            "architecture": "generic",
            "directives": [],
            "hpa-feature-attributes": [
              { "hpa-attribute-key": "numVirtualCpu", "hpa-attribute-value": "6", "operator": "=", "unit": "" }
            ]
          },
          {
            "hpa-feature": "basicCapabilities",
            "mandatory": "True",
            "architecture": "generic",
            "directives": [],
            "hpa-feature-attributes": [
              { "hpa-attribute-key": "virtualMemSize", "hpa-attribute-value": "6", "operator": "=", "unit":
"GB" },
            ]
          },
          {
            "hpa-feature": "sriovNICNetwork",
            "mandatory": "True",
            "architecture": "generic",
            "directives" : [
              {
                "directive_name": "sriovNICNetwork_directive",
                "attributes": [
                   { "attribute_name": "<Blank>",
                     "attribute_value": "<Blank>"
                   },
                   { "attribute_name": "<Blank>",
                     "attribute_value": "<Blank>"
                   }
                ]
              }
            ],
            "hpa-feature-attributes": [
              { "hpa-attribute-key": "pciVendorId", "hpa-attribute-value": "1234", "operator": "=", "unit": ""
},
              { "hpa-attribute-key": "pciDeviceId", "hpa-attribute-value": "5678", "operator": "=", "unit": ""
},
              { "hpa-attribute-key": "pciCount", "hpa-attribute-value": "1", "operator": ">=", "unit": "" }
            ]
          },
          {
            "hpa-feature": "sriovNICNetwork",
            "mandatory": "True",
            "architecture": "generic",
            "directives" : [
              {
                "directive_name": "sriovNICNetwork_directive",
                "attributes": [
                   { "attribute_name": "<Blank>",
                     "attribute_value": "<Blank>"
                   }
                   { "attribute_name": "<Blank>",
```

```
                    "attribute_value": "<Blank>"
                  }
                ]
              }
            ],
            "hpa-feature-attributes": [
              { "hpa-attribute-key": "pciVendorId", "hpa-attribute-value": "3333", "operator": "=", "unit": ""
},
              { "hpa-attribute-key": "pciDeviceId", "hpa-attribute-value": "7777", "operator": "=", "unit": ""
},
              { "hpa-attribute-key": "pciCount", "hpa-attribute-value": "1", "operator": ">=", "unit": "" }
            ]
          }
        ]
      }
    ]
  }
}
```

OOF check AAI database for existing instances/available cloud regions

The data in AAI still follows the routine schema designed in HPA Policies and Mappings. The only part has been changed is inside sriovNICNetwork. We added one 'directive' attributes to contain the 'vnic_type' and 'physicalNetwork' that needed by VF-C.

Here we just provide data for one sample flavor including the requirements for the previous example:

Data inside AAI

```
hpa-capability-id="b369fd3d-0b15-44e1-81b2-6210efc6dff9",
hpa-feature= "basicCapabilities",
architecture= "generic",
version= "v1",
hpa-attribute-key       hpa-attribute-value
numVirtualCpu           {value:6}
virtualMemSize          {value:6, unit:"GB"}


hpa-capability-id="f453fd3d-0b15-11w4-81b2-6210efc6dff9",
hpa-feature= "sriovNICNetwork",
architecture= "intel64",
version= "v1",
hpa-attribute-key       hpa-attribute-value
pciCount                {value: 1}
pciVendorId             {value: "8086"}
pciDeviceId             {value: "0443"}



hpa-capability-id="f453fd3d-0b15-11w4-81b2-873hf8oo98s0",
hpa-feature= "sriovNICNetwork",
architecture= "intel64",
version= "v1",
hpa-attribute-key       hpa-attribute-value
pciCount                {value: 1}
pciVendorId             {value: "6808"}
pciDeviceId             {value: "3440"}
```

OOF process homing allocation and return homing placement to VF-C

OOF will match the requirements inside the policies with the data of the available candidates(cloud-regions or existing candiates) to find an optimal solution to place that service.

After OOF gives out the most appropriate placement for that VNF/services, it will respond the solution back to VF-C with schema like below. Then a module in NSLCM will get that response and store them inside its Database for later instantiation.

Sample OOF-VFC response

```
{
  "requestId": "xxxx",
  "transactionId": " xxxx ",
  "statusMessage": " xxxx ",
  "requestStatus": "completed",
  "solutions": {
    "placementSolutions": [
      [
        { "resourceModuleName": " vG ",
          "serviceResourceId": " xxxx ",
          "solution": {
           "identifierType": "serviceInstanceId",
           "identifiers": [
             " xxxx "
             ],
           "cloudOwner": " xxxx "
          },
         "assignmentInfo": [
            {  "key":"isRehome",
               "value":"false"    },
            {  "key":"locationId",
               "value":"DLLSTX1A"    },
            {  "key":"locationType",
               "value":"openstack-cloud" },
            {  "key":"vimId",
               "value":"rackspace_DLLSTX1A"  },
            {  "key":"physicalLocationId",
               "value":"DLLSTX1223"    },
            {  "key":"oofDirectives",
               "value": { "directives":[
                 { "id":"<vdu.name>",
                   "type":"vnfc",
                   "directives":[
                      {  "type":"flavor_directive",
                         "attributes":[
                             {   "attribute_name":"flavor_name", //just a string 'flavor_name'
                                 "attribute_value":"<flavor_name>"  #VIM Flavor, which oof seleted.
                             }
                         ]
                      },
                      {  "type": "sriovNICNetwork_directive",
                         "attributes":[
                             {"attribute_name":"vnic_type", "attribute_value":"direct"},
                             {"attribute_name":"provider_network", "attribute_value":"physnet1"}
                         ]
                      },
                      {  "type": "sriovNICNetwork_directive",
                         "attributes": [
                             {"attribute_name":"vnic_type", "attribute_value":"direct"},
                             {"attribute_name":"provider_network", "attribute_value":"physnet2"}
                         ]
                       }
                    ]
                  }
                ]
               }
             }
           ]
         }
      ]
    ],
    "licenseSoutions": [
      {
       "resourceModuleName": "string",
```

```
        "serviceResourceId": "string",
        "entitlementPoolUUID": [
          "string"
         ],
        "licenseKeyGroupUUID": [
          "string"
         ],
        "entitlementPoolInvariantUUID": [
          "string"
         ],
        "licenseKeyGroupInvariantUUID": [
          "string"
         ]
       }
     ]
  }
}
```

MultiCloud

OpenStack Config SRIOV

Openstack configuration:

1. NIC configuration refer to **https://docs.openstack.org/neutron/pike/admin/config-sriov.html**
2. An example of a site having three types of compute nodes. 1st set of compute nodes have two SRIOV NIC cards with vendor/device id as 1234, 5678 and vendor/device id as 2345 &6789. 2nd set of compute nodes have two SRIOV-NIC of same type 4321 & 8765. And the third set of compute nodes don't have any SRIOV-NIC cards. And hence OpenStack administrator at the site creates three flavors to reflect the hardware the site has. As you see in this example, it is expected that alias format is followed. Alias value supposed to be of the form "NIC-sriov-<vendor ID>-<device ID>-<Provider network>

   $ openstack flavor create flavor1 --id auto --ram 512 --disk 40 --vcpus 4

   $ openstack flavor set flavor1 --property pci_passthrough:alias=sriov-nic-intel-8086-0443-physnet1:1

   $ openstack flavor set flavor1 --property pci_passthrough:alias=sriov-nic-intel-6808-3440-physnet2:1

Multi-cloud discovery

When it reads the flavors information from OpenStack site, if the pci_passthrough alias starts with SRIOV-NIV, then it assumes that it is SRIOV NIC type.

Next two integers are meant for vendor id and device id.

If it is present after device id, it is assumed to be provider network.

As part of discovery, it populates the A&AI with two PCIe features for **Flavor1**.

hpa-feature=”sriovNICNetwork”,

architecture=”{hw_arch}",

version=”v1”,

Hpa-attribute-key	Hpa-attribute-value
pciVendorId	8086
pciDeviceId	0443
pciCount	1

hpa-feature=”sriovNICNetwork”,

architecture=”{hw_arch}",

version=”v1”,

Hpa-attribute-key	Hpa-attribute-value
pciVendorId	6808
pciDeviceId	3440
pciCount	1

Multi-cloud API

Multi-cloud network API for VF-C

```
Create Network

Request

    {
        "tenant": "tenant1",
        "networkName": "ommnet",
        "shared": 1,
        "vlanTransparent": 1,
        "networkType": "vlan",
        "segmentationId": 202,
        "physicalNetwork": "ctrl",
        "routerExternal": 0
    }

Response

    {
        "returnCode": 0,
        "vimId": "11111",
        "vimName": "11111",
        "status": "ACTIVE",
        "id": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
        "name": "net1",
        "tenant": "tenant1",
        "networkName": "ommnet",
        "shared": 1,
        "vlanTransparent": 1,
        "networkType": "vlan",
        "segmentationId": 202,
        "physicalNetwork": "physnet1",
        "routerExternal": 0
    }
```

Multi-cloud subnet API for VF-C

Create Subnets

Request

```
    {
        "tenant": "tenant1",
        "network_id": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
        "subnetName": "subnet1",
        "cidr": "10.43.35.0/24",
        "ipVersion": 4,
        "enableDhcp": 1,
        "gatewayIp": "10.43.35.1",
        "dnsNameservers": [],
        "allocationPools": [{
            "start": "192.168.199.2",
            "end": "192.168.199.254"
        }],
        "hostRoutes": []
    }
```

Response

```
    {
        "returnCode": 0,
        "vimId": "11111",
        "vimName": "11111",
        "status": " ACTIVE",
        "id": " d62019d3-bc6e-4319-9c1d-6722fc136a23",
        "tenant": "tenant1",
        "network_id": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
        "name": "subnet1",
        "cidr": "10.43.35.0/24",
        "ipVersion": 4,
        "enableDhcp": 1,
        "gatewayIp": "10.43.35.1",
        "dnsNameservers": [],
        "allocationPools": [{
            "start": "192.168.199.2",
            "end": "192.168.199.254"
        }],
        "hostRoutes": []
    }
```

Multi-cloud Port API for VF-C

Create Virtual Port

Request

```
    {
        "networkId": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
        "subnetId": "c17afb1c-ab84-11e8-bfaa-77f95f421148",
        "name": "port1",
        "macAddress": "d4:5d:df:09:9e:19",
        "ip": "192.168.199.3",
        "vnicType": "direct"
    }
```

Response

```
    {
        "returnCode": 1,
        "vimId": "11111",
        "vimName": "11111",
        "cloud-owner": "cloudowner",
        "cloud-region-id": "cloudregion",
        "status": "success",
        "id": "393905cc-ab85-11e8-bb60-1f10d99adcef",
        "name": "port_a",
        "tenantId": "55ab62ae-ab85-11e8-ba44-03ef3e9bc9d6",
        "networkName": "ommnet",
        "networkId": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
        "subnetName": "subnet1",
        "subnetId": "c17afb1c-ab84-11e8-bfaa-77f95f421148",
        "macAddress": "d4:5d:df:09:9e:19",
        "ip": "192.168.199.3",
        "vnicType": "direct",
    }
```

Multi-cloud server API for VF-C

```
Create Server

Request
    {
        "tenant": "tenant1",
        "name": "vm1",
        "availabilityZone": "az1",
        "flavorName": "vm_large",
        "boot": {
            "type": 1,
            " volumeName": "volume1"
        },
        "flavorId": "vm_large_134213",
        "contextArray": [{
            "fileName": "test.yaml",
            "fileData": "…."
        }],
        "volumeArray": [{
            "volumeName": "vol1",
        }],
        "nicArray": [{
            "portId": "port_a"
        }],
        "metada": [{
            "keyName": "foo",
            "value": "foo value"
        }],
        "userdata": "abcdedf"
    }

Response

    {
        "vimId": "11111",
        "vimName": "11111",
        "cloud-owner": "cloudowner",
        "cloud-region-id": "cloudregion",
        "returnCode": "success",
        "id": "1234-23545",
        "name": "vm1",
        "tenantId": "tenant1",
        "boot": {
            "type": 1,
            " volumeName": "volume1"
        },
        "volumeArray": [{
            "volumeName": "vol1",
        }],
        "nicArray": [{
            "portId": "port_a"
        }],
        "availabilityZone": "zone1",
        "flavorId": "tenant1",
        "metada": [{
            "keyName": "foo",
            "value": "foo value"
        }],
    }
```

HPA VF-C Casablanca Stories

Key	Summary	T	Created	Updated	Due	Assignee	Reporter	P	Status	Resolution
VFC-641	Integration with OOF		Jan 08, 2018	Aug 12, 2023		Unassigned	None		CLOSED	Done
VFC-939	HPA feature support		Jun 25, 2018	Aug 12, 2023		Unassigned	None		CLOSED	Done

2 issues