OpenNESS 19.12 Integration

- OpenNESS 19.12 Design
- Gap Analysis for Integrating OpenNESS with ICN
  - Network Policy
  - DNS
  - Cross-Node communication
  - OS (Ubuntu)
- Openness Integration Design
  - Openness Microserivces
  - Openness integration for Multus, SR-IOV CNI, SR-IOV Network Device Plugin, FPGA, Bios, Topology Manager, CMK, NFD
  - Openness integration test plan for Multus, SR-IOV CNI, SR-IOV Network Device Plugin, Topology Manager, CMK, NFD
  - Add more realistic test cases for platform related micro-services
  - Task List
  - Application
- ICN Requirements for adding EAA support for geo-distributed producing and consuming applications
  - BACKGROUND
  - REQUIREMENT

OpenNESS 19.12 Design

Openness released 19.12 on December 21 2019 and this new release has removed the deployment mode (kubernetes + NTS). Two modes is supported now: Native deployment Mode (which is based on pure docker/libvirt) and Infrastructure Mode (which is based on kube-ovn), below are the brief summary of the difference of these 2 modes:

Functionality	Native Deployment Mode	Infrastructure Deployment Mode
Usage Scenarios	On-Premises Edge	Network Edge
Infrastructure	Virtualization base: docker/libvirt Orchestration: OpenNESS controller Network: docker network (container) + NTS (through new added KNI interface)	Orchestration: Kubernetes Network: kube-ovn CNI
Micro-Services in OpenNESS Controller	**Web UI:** controller UI Edge Node/Edge application lifecycle management **Core Network Configuration** Telemetry	**Core Network Configuration**: Configure the access network (e.g., LTE/CUPS, 5G) control plane Telemetry
Micro-Services in OpenNESS Node	**EAA**: application/service registration, authentication etc. **ELA/EVA/EDA**: used by controller to configure host interfaces, network policy (used by NTS), create/destroy application etc. **DNS**: for client to access MS in edge node **NTS**: traffic steering	**EAA**: application/service registration, authentication etc. **EIS**(Edge Interface Service), looks to be similar with providernet implemented in ovs4nfv k8s CNI **DNS**: for client to access MS in edge node
Application on-boarding	OpenNESS Controller Web UI or Restful API	Kubernetes (e.g. Kubectl apply -f application.yaml) **Note**: unlike 19.09, No UI used to on-board application
Edge node interface configuration	**ELA** (Edge LifeCycle Agent, Implemented by OpenNESS) – Configurated by OpenNESS controller	**EIS** (Edge Interface Service, which is an kubectl extension to configurate edge node host network adapter), use *e.g. kubectl interfaceservice attach $NODE_NAME $PCI_ADDRESS*
Traffic Policy configuration	**EDA** (Edge Dataplane Agent, Implemented by OpenNESS) – Configurated by OpenNESS controller	Kubenetes Network Policy CRD *e.g. kubectl apply -f network_policy.yml* **Note**: unlike 19.09, No UI used to configure policy

DataPlane Service	**NTS** (Implemented based on DPDK in OpenNESS) to provide additional KNI interface for container	kube-ovn + Network policy

Gap Analysis for Integrating OpenNESS with ICN

Network Policy

Network policy and DNS is used for traffic steering. Network policy is used for restrict access among services but NOT “proactively” forward the traffic, While the OpenNESS DNS service can help “redirect” the external client’s traffic to the edge application service

By default, in a Network Edge environment, all ingress traffic is blocked (services running inside of deployed applications are not reachable) and all egress traffic is enabled (pods are able to reach the internet). The following NetworkPolicy definition is used:

Default network policy: block all ingress

```
apiVersion: networking.k8s.io/v1
metadata:
  name: block-all-ingress
  namespace: default        # selects default namespace
spec:
  podSelector: {}           # matches all the pods in the default namespace
  policyTypes:
  - Ingress
  ingress: []               # no rules allowing ingress traffic = ingress blocked
```

Admin can enable access to certain service by applying a NetworkPolicy CRD. For example:

1. To deploy a Network Policy allowing ingress traffic on port 5000 (tcp and udp) from 192.168.1.0/24 network to OpenVINO consumer application pod, create the following specification file for this Network Policy:

Admin defined network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: openvino-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      name: openvino-cons-app
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 192.168.1.0/24
    ports:
    - protocol: TCP
      port: 5000
    - protocol: UDP
      port: 5000
```

2. Create the Network Policy:
kubectl apply -f network_policy.yml

DNS

DNS service can help “redirect” the external client’s traffic to the edge application service. This gap analysis is to investigate whether OpenNESS DNS can be used for ICN traffic steering or not.

OpenNESS provides DNS server which provides the microsevice’s ip address based on FQDN. OpenNESS extends kubectl utility with kubectl edgedns cmd to set/delete DNS entry. For example

1. define a file with below content: openvino-dns.json
   {
    "record_type":"A",

```
  "fqdn":"openvino.openness",
  "addresses":["10.16.0.10"]
}
```

2. Then use below command to add an entry in OpenNESS DNS server:
   kubectl edgedns set <edge_node_host_name> openvino-dns.json

Below are implement details of OpenNESS DNS server:

- Run as independent process/container in each Edge Node : ./edgednssvr -port 53 -fwdr=8.8.8.8 -db XXX.db // port: DNS server port; fwdr: forwarder ip used when cannot found FQDN in OpenNESS DNS DB; db: OpenNESS db file
- Provide 2 servers after running:
  Control Server: gRPC/IP based API to receive DNS record add/remove request – OpenNESS controller can call this interface to add DNS record
  DNS server: DNS service is based on https://github.com/miekg/dns
- DNS process flow: After get a DNS request, it will try to find the FQDN in local OpenNESS DNS db first, if not found, forward the request to an external forwarder (default is 8.8.8.8, set by “-fwdr“ parameter)

The OpenNESS DNS service is different from K8s’ CoreDNS to support different usages:

- CoreDNS: provides DNS service within K8s cluster, e.g. from app in container to find the service also running in container of the same cluster.
- OpenNESS DNS: provides DNS service for app of external host which is not running in the edge cluster to find a app (which may not be a K8s service, so its ip may not be recorded in coreDNS) in k8s cluster. e.g. in OpenNESS OpenVINO demo, the video stream generator is running in a separate host, admin needs manually (add a new name server in /etc/resolv.conf) set it’s DNS server IP to point to OpenNESS edge node DNS server then it can know how to send the stream.

Cross-Node communication

Edge apps can be divided into producer and consumer. This gap analysis is to investigate the communication between the producers and consumers which are on different edge nodes.

Edge applications must introduce themselves to OpenNESS framework and identify if they would like to activate new edge services or consume an existing service. Edge Application Agent (EAA) component is the handler of all the edge applications hosted by the OpenNESS edge node and acts as their point-of-contact.

OpenNESS-awareness involves (a) authentication, (b) service activation/deactivation, (c) service discovery, (d) service subscription, and (e) Websocket connection establishment. The Websocket connection retains a channel for EAA for notification forwarding to pre-subscribed consumer applications. Notifications are generated by "producer" edge applications and absorbed by "consumer" edge applications.

The sequence of operations for the producer application:

1. Authenticate with OpenNESS edge node
2. Activate new service and include the list of notifications involved
3. Send notifications to OpenNESS edge node according to business logic

The sequence of operations for the consumer application:

1. Authenticate with OpenNESS edge node
2. Discover the available services on OpenNESS edge platform
3. Subscribe to services of interest and listen for notifications

Edge apps will access eaa through eaa.openness (name.namespace) which is a kubernetes service:
https://github.com/open-ness/edgecontroller/blob/master/kube-ovn/openness.yaml#L18

For example: as following links show, openvino consumer will access http://eaa.openness:443/auth for authentication.
https://github.com/open-ness/edgeapps/blob/master/openvino/consumer/cmd/main.go#L24
https://github.com/open-ness/edgeapps/blob/master/openvino/consumer/cmd/main.go#L66

eaa is deployed as a deployment and only 1 eaa will be deployed:
https://github.com/open-ness/edgecontroller/blob/master/kube-ovn/openness.yaml#L41

Because all edge apps will access only 1 eaa, it doesn't matter that eaa is stateful.

For example:
only 1 eaa is deployed on node1. producer1 and producer2 will activate the new service with eaa. consumer1 and consumer2 will consume services stored in eaa. Because all the information are stored in only 1 eaa, there won't be issues.

```
        node1                                  node2

        eaa

producer1    consumer1                  producer2    consumer2
```

Because edge apps on different edge node all can access service eaa, the consumer can consume the service provided by producer which is on a different node.

For example:
producer1 is located in node1 and consumer2 is located on node2. The networking flow will be:
  producer1 -> service eaa -> pod eaa
  consumer2 -> service eaa -> pod eaa
node1 node2

eaa

producer1 consumer2

OS (Ubuntu)

OpenNESS only supports Centos but ICN is based on Ubuntu 18.04. This gap analysis is to investigate how to deploy OpenNESS on Ubuntu 18.04

OpenNESS only supports Centos but ICN is based on Ubuntu 18.04. By changing the ansible scripts of OpenNESS, it is able to deploy OpenNESS on Ubuntu 18.04. The following parts of ansible scripts need to change:
1. Following ansible roles can be removed for OpenNESS master: grub, cnca, multus, nfd. Ansible role grub can be removed for OpenNESS node. Because:

- grub is used to add hugepages to grub and hugepages are not useful for integration OpenNESS with ICN.
- cnca is not required for integration.
- multus has already been integrated with ICN.
- nfd will be integrated directly with ICN.

2. Centos uses yum to install packages and we need to use apt for Ubuntu.

3. Some packages which will be installed by ansible scripts should be removed or replaced:

- Some Centos packages doesn't exist on Ubuntu and these packages should be removed. For example, yum-utils, device-mapper-persistent-data.
- Some Centos packages' name are different for Ubuntu. For example, python2-pip should be replaced with python-pip, python-devel should be replaced with python-dev.

4. Selinux is not used on Ubuntu and need to remove the ansible scripts configuring selinux.

5. Epel repository is for Centos and Ubuntu doesn't need this repository.

6. Proxy will be set for yum and need to change the scripts to set proxy for apt.

7. Docker installation for Centos and Ubuntu are different. Need to change the scripts following the installation guide. For example: the docker repository is different for Centos and Ubuntu.

8. Auditd is used for Docker. Auditd is delivered with Centos by default but Ubuntu needs to install auditd.

9. Kubernetes installation for Centos and Ubuntu are different. Need to change the scripts following the installation guide. For example: gpg key is different for Centos and Ubuntu, ubuntu use deb and Centos uses repository.

10. cgroups driver is different for Centos (systemd) and Ubuntu (cgroups). By default, cgroups driver is cgroups and need to remove the ansible scripts which configures cgroups driver to systemd.

11. firewalld is used in Centos and need to change to ufw which is used by Ubuntu.

12. Packages are different for installing openvswitch and ovn. Centos uses RPMs. Ubuntu uses openvswitch-switch, ovn-common, ovn-central and ovn-host.

13. Topology manager and CPU manager is configured for edge node's kubelet. No need to use topology manager and can remove these.

Openness Integration Design

Openness Microserivces

We are planning to integrate Openness Infrastructure mode. The following figure shows the microservices of Openness infrastructure mode and also lists the microserivces that we propose to integrate.

Microservices of Openness Infrastructure mode	Description	Deployment method	Deployment of the component	Propose to integrate
eaa	application/service registration, authentication etc	deployment	edge node	yes
edgedns	for client to access microservices in edge node	daemonset (propose to change to deployment)	edge node	yes

interfaceservice	similar with providernet implemented in ovn4nfv-k8s-plugin	daemonset	edge node	no, will use ovn4nfv-k8s-plugin's provider network
cnca	Core Network Configuration: Configure the access network (e.g., LTE/CUPS, 5G) control plane	deployment	controller	no
syslog	log service for openness	daemonset	controller & edge node	no
multus	enabling attaching multiple network interfaces to pods	daemonset	controller & edge node	Already covered by ONAP4K8s - KUD
nfd	node feature discovery	daemonset	controller & edge node	Already covered by ONAP4K8s - KUD
sriov	sriov network device plugin & sriov cni	daemonset	controller & edge node	Already covered by ONAP4K8s - KUD
topology manager	kubernetes topology manager	Kubelet component	controller & edge node	Work in Progress to upgrade the K8s v16.0 integrate into ONAP4K8s - KUD
CMK	CPU Manager	part of kubelet	controller & edge node	Work in Progress - Integrate into ONAP4K8s - KUD
bios	Used for change BIOS and firmware configuration: CPU configuration, Cache and Memory configuration, PCIe Configuration, Power and Performance configuration, etc	privileged Pod	controller & edge node	Required for ICN? Already in ICN Metal3, could be enabled part of it
fpga	Open Programmable Acceleration Engine (OPAE) package consisting of a kernel driver and user space FPGA utils package that enables programming of the FPGA is used. sriov is used to configure the FPGA resources such as Virtual Functions and queues	pod	controller & edge node	Need to integrate into ONAP4K8s - KUD with FPGA device

Openness integration for Multus, SR-IOV CNI, SR-IOV Network Device Plugin, FPGA, Bios, Topology Manager, CMK, NFD

Microservice	Integration Detail	Components	Testing	Dependency	Request to openness team	Propose to integrate
multus	Version 3.3 Downloading the following 3 files: kustomization.yml**[1]** rename_default_net.yml**[2]** multus-daemonset.yml**[3]** multus-daemonset.yml**[4]** And then run the following command to kustomize the yml file and then apply. kubectl kustomize . | kubectl apply -f - Command kustomize will add parameter: "--rename-conf-file=true" to the daemonset yml file like following: Containers: - args: - --multus-conf-file=auto - --cni-version=0.3.1 **- --rename-conf-file=true** This parameter will add suffix ".old" to the original cni conf file. For example, ".old" is added to the kube-ovn conf file as below: [root@master net.d]# ls /etc/cni/net.d/ **00-kube-ovn.conflist.old** 00-multus.conf multus.d **[1]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/multus/files/kustomization.yml **[2]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/multus/files/rename_default_net.yml **[3]**https://raw.githubusercontent.com/intel/multus-cni/v3.3/images/multus-daemonset.yml **[4]**https://raw.githubusercontent.com/intel/multus-cni/v3.3/images/multus-daemonset.yml	multus running as daemonset	Not found	Nothing	Test cases are missing and need to ask where the test cases are.	No This is standard multus and only changes a parameter.
sriov cni	git clone https://github.com/intel/sriov-cni docker build . -t nfvpe/sriov-cni kubectl create -f ./images/k8s-v1.16/sriov-cni-daemonset.yaml **[1]** kubectl create -f openness-sriov-crd.yml**[2]** **[1]**https://github.com/intel/sriov-cni/blob/master/images/k8s-v1.16/sriov-cni-daemonset.yaml **[2]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/sriov/master/files/openness-sriov-crd.yml	sriov cni running as daemonset	Not found	SR-IOV enabled NIC	Test cases are missing and need to ask where the test cases are.	No This is standard sriov cni.

sriov network device plugin	git clone https://github.com/intel/sriov-network-device-plugin
	
	if fpga_sriov_userspace.enabled:
	patch FPGA_SRIOV_USERSPACE_DEV_PLUGIN.patch**[1]** to sriov network device plugin directory
	
	make image
	
	if fpga_sriov_userspace.enabled:
	kubectl create -f fpga_configMap**[2]**
	else:
	kubectl create -f sriov_configMap**[3]**
	
	kubectl create -f ./deployments/k8s-v1.16/sriovdp-daemonset.yaml**[4]**
	
	Provide ansible scripts to create VF and bind igb_uio driver.
	
	If FPGA is used, fpga_sriov_userspace.enabled should be set to true. Then FPGA_SRIOV_USERSPACE_DEV_PLUGIN.patch will be patched to sriov network device plugin. This patch enables sriov network device plugin to control fpga devices which are bounded to userspace driver. Fpga_configMap will be applied and this configmap will create resource intel_fec_5g and intel_fec_lte which is based on fpga device by specifying vendor_id, device_id and driver.
	
	[1]https://github.com/open-ness/edgecontroller/blob/master/fpga/FPGA_SRIOV_USERSPACE_DEV_PLUGIN.patch
	[2]https://github.com/open-ness/edgecontroller/blob/master/fpga/configMap.yaml
	[3]https://github.com/open-ness/edgecontroller/blob/master/sriov/configMap.yaml
	[4]https://github.com/intel/sriov-network-device-plugin/blob/master/deployments/k8s-v1.16/sriovdp-daemonset.yaml
		sriov network device plugin running as daemonset	Not found	SR-IOV enabled device	Test cases are missing and need to ask where the test cases are.	Yes
						Special patch will be patched to this project and is needed by FPGA.

fpga	bbdev_config_service, n3000-1-3-5-beta-rte-setup.zip, n3000-1-3-5-beta-cfg-2x2x25g-setup.zip, flexran-dpdk-bbdev-v19-10.patch, FPGA image for 5GNR vRAN are not available and need to ask openness team. fpga_sriov_userspace.enabled should be set to true. On master node, build the kubectl plugin rsu (Remote System Update) and move the binary file to directory the /usr/bin/. This plugin will create kubernetes jobs and run OPAE in those jobs. OPAE(Open Programmable Acceleration Engine) enables programming of the FPGA and is used to program the FPGA factory image or the user image (5GN FEC vRAN). The plugin also allows for obtaining basic FPGA telemetry such as temperature, power usage and FPGA image information. On worker node, using n3000-1-3-5-beta-rte-setup.zip (can be used to install OPAE), n3000-1-3-5-beta-cfg-2x2x25g-setup.zip to build docker image 'fpga-opae-pacn3000:1.0'. OPAE will be installed in this docker image. RSU will create a kubernetes job which uses image 'fpga-opae-pacn3000:1.0' as below: apiVersion: batch/v1 kind: Job metadata: name: fpga-opae-job spec: template: spec: containers: - securityContext: privileged: true name: fpga-opea **image: fpga-opae-pacn3000:1.0** imagePullPolicy: Never command: ["/bin/bash", "-c", "--"] **args: ["./check_if_modules_loaded.sh && fpgasupdate /root/images/<img_name> <RSU_PCI_bus_function_id> && rsu bmcimg (RSU_PCI_bus_function_id)"]** volumeMounts: - name: class mountPath: /sys/devices readOnly: false - name: image-dir mountPath: /root/images readOnly: false volumes: - hostPath: path: "/sys/devices" name: class - hostPath: path: "/temp/vran_images" name: image-dir restartPolicy: Never nodeSelector: kubernetes.io/hostname: samplenodename backoffLimit: 0 User FPGA images will be put in the directory /temp/vran_images/. To configure the VFs with the necessary number of queues for the vRAN workload the BBDEV configuration utility is to be run as a job within a privileged container. make build-docker-fpga-cfg kubectl create -f fpga-sample-configmap.yaml**[1]** kubectl create -f fpga-config-job.yaml**[2]** A sample pod requesting the FPGA (FEC) VF may look like this: apiVersion: v1 kind: Pod Metadata: name: test Labels: env: test spec: containers: - name: test image: centos:latest command: ["/bin/bash", "-c", "--"] args: ["while true; do sleep 300000; done;"] resources: requests: intel.com/intel_fec_5g: '1' limits: intel.com/intel_fec_5g: '1' **[1]**https://github.com/open-ness/edgecontroller/blob/master/fpga/fpga-sample-configmap.yaml **[2]**https://github.com/open-ness/edgecontroller/blob/master/fpga/fpga-config-job.yaml	kubectl plugin rsu	Not found	Intel® FPGA Programmable Acceleration Card (Intel FPGA PAC) N3000, DPDK 18.08, Hugepage support	1. Test cases are missing and need to ask where the test cases are. 2.bbdev_config_service, n3000-1-3-5-beta-rte-setup.zip, n3000-1-3-5-beta-cfg-2x2x25g-setup.zip, flexran-dpdk-bbdev-v19-10.patch are not available. Need to request these packages. 3. FPGA image for 5GNR vRAN is not available. Need to request this image. 4.What's the difference between flexran and vran	Yes This is developed by openness team and FPGA is requested by Srini.

bios	On master node, build the kubectl plugin biosfw and move the binary file to directory the /usr/bin/. This plugin will create a kubernetes job and run syscfg in that job. Intel® System Configuration Utility (Syscfg) is a command-line utility that can be used to save and restore BIOS and firmware settings to a file or to set and display individual settings.
	
	On worker node, using syscfg_package.zip to build docker image 'openness-biosfw'. Syscfg will be upzipped in this docker image. The kubernetes job created by kubectl plugin biosfw will use this image 'openness-biosfw' as below:
	apiVersion: batch/v1
	kind: Job
	metadata:
	  name: openness-biosfw-job
	spec:
	  backoffLimit: 0
	  activeDeadlineSeconds: 100
	  template:
	  spec:
	  restartPolicy: Never
	  containers:
	  - name: openness-biosfw-job
	  image: openness-biosfw
	  imagePullPolicy: Never
	  securityContext:
	  privileged: true
	  args: ["$(BIOSFW_COMMAND)"]
	  env:
	  - name: BIOSFW_COMMAND
	  valueFrom:
	  configMapKeyRef:
	  name: biosfw-config
	  key: COMMAND
	  volumeMounts:
	  - name: host-devices
	  mountPath: /dev/mem
	  - name: biosfw-config-volume
	  mountPath: /biosfw-config/
	  volumes:
	  - name: host-devices
	  hostPath:
	  path: /dev/mem
	  - name: biosfw-config-volume
	  configMap:
	  name: biosfw-config
		Kubectl plugin biosfw	Not found	certain Intel® Server platforms
				
				https://downloadcenter.intel.com/download/28713/Save-and-Restore-System-Configuration-Utility-SYSCFG-
					1.Test cases are missing and need to ask where the test cases are.
					
					2. Ask the server version, motherboard version, bios version for testing epa feature bios?
						Yes
						This is developed by openness team.

topology manager	Configure kubelet on the worker node as below:
	1. Set cpuManagerPolicy to static
	2. Set topologyManagerPolicy to best-effort
	# BEGIN OpenNESS configuration - General
	
	apiVersion: kubelet.config.k8s.io/v1beta1
	kind: KubeletConfiguration
	KubeletCgroups: "/systemd/system.slice"
	Authentication:
	  x509:
	  clientCAFile: /etc/kubernetes/pki/ca.crt
	clusterDNS:
	  - 10.96.0.10
	clusterDomain: cluster.local
	featureGates:
	  TopologyManager: True
	podPidsLimit: 2048
	# END OpenNESS configuration - General
	# BEGIN OpenNESS configuration - CPU Manager
	cpuManagerPolicy: static
	kubeReserved:
	  cpu: "1"
	# END OpenNESS configuration - CPU Manager
	# BEGIN OpenNESS configuration - Topology Manager
	topologyManagerPolicy: best-effort
	# END OpenNESS configuration - Topology Manager
		Kubelet component	Not found	K8s 1.16	Test cases are missing and need to ask where the test cases are.	No
						This is standard topology manager.

CMK	Download the following files:
	cmk-namespace.yaml**[1]** cmk-serviceaccount.yaml**[2]** cmk-rbac-rules.yaml**[3]** cmk-cluster-init-pod.yaml**[4]**
	
	Copy following files to the same directory as cmk-namespace.yaml, cmk-serviceaccount.yaml, cmk-rbac-rules.yaml and cmk-cluster-init-pod.yaml:
	Kustomization.yml**[5]** rewrite_args.yml.j2**[6]**
	
	Run the following command:
	kubectl kustomize . | kubectl apply -f -
	This kustomize command will change the parameters in cmk-cluster-init-pod.yaml:
	- args:
	  # Change this value to pass different options to cluster-init.
	  - "/cmk/cmk.py cluster-init --host-list=**node1,node2,node3** --saname=**cmk-serviceaccount** --namespace=**cmk-namespace**"
	
	On each worker node, clone the project https://github.com/intel/CPU-Manager-for-Kubernetes and then checkout the commit e3df769521558cff7734c568ac5d3882d4f41af9. Using command 'make' to build the docker image.
	
	[1]https://raw.githubusercontent.com/intel/CPU-Manager-for-Kubernetes/e3df769521558cff7734c568ac5d3882d4f41af9/resources/authorization/cmk-namespace.yaml
	[2]https://raw.githubusercontent.com/intel/CPU-Manager-for-Kubernetes/e3df769521558cff7734c568ac5d3882d4f41af9/resources/authorization/cmk-serviceaccount.yaml
	[3]https://raw.githubusercontent.com/intel/CPU-Manager-for-Kubernetes/e3df769521558cff7734c568ac5d3882d4f41af9/resources/authorization/cmk-rbac-rules.yaml
	[4]https://raw.githubusercontent.com/intel/CPU-Manager-for-Kubernetes/e3df769521558cff7734c568ac5d3882d4f41af9/resources/pods/cmk-cluster-init-pod.yaml
	[5]https://github.com/open-ness/openness-experience-kits/blob/master/roles/cmk/master/files/kustomization.yml
	[6]https://github.com/open-ness/openness-experience-kits/blob/master/roles/cmk/master/templates/rewrite_args.yml.j2
			Not found	Nothing	Test cases are missing and need to ask where the test cases are.	No
						This is standard CMK.

nfd	version: v0.4.0 Download the following files: Nfd-master.yaml.template**[1]** nfd-worker-daemonset.yaml.template**[2]** Copy following files to the same directory as nfd-master.yaml.template and nfd-worker-daemonset.yaml.template: Add_nfd_namespace.yaml**[3]** kustomization.yml**[4]** replace_cluster_role_binding_namespace.yml**[5]** replace_service_account_namespace.yml**[6]** enable_nfd_master_certs.yml.j2**[7]** enable_nfd_worker_certs.yml.j2**[8]** Run the following command to kustomize the files (nfd-master.yaml and nfd-worker-daemonset.yaml): kubectl kustomize . | kubectl apply -f - The above kustomize command will replace the namespace 'default' with 'openness', add certs to nfd-master and nfd-worker. Apply below network policy to allow the communication between nfd-master and nfd-worker: https://github.com/open-ness/edgecontroller/blob/master/kube-ovn/nfd_network_policy.yml **[1]**https://raw.githubusercontent.com/kubernetes-sigs/node-feature-discovery/v0.4.0/nfd-master.yaml.template **[2]**https://raw.githubusercontent.com/kubernetes-sigs/node-feature-discovery/v0.4.0/nfd-worker-daemonset.yaml.template **[3]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/nfd/files/add_nfd_namespace.yml **[4]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/nfd/files/kustomization.yml **[5]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/nfd/files/replace_cluster_role_binding_namespace.yml **[6]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/nfd/files/replace_service_account_namespace.yml **[7]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/nfd/templates/enable_nfd_master_certs.yml.j2 **[8]**https://github.com/open-ness/openness-experience-kits/blob/master/roles/nfd/templates/enable_nfd_worker_certs.yml.j2	nfd-master running as daemonset on kubernetes master node nfd-worker running as daemonset	Not found	Nothing	Test cases are missing and need to ask where the test cases are.	No This is standard nfd and only a few changes are applied such as namespace, certs.

Openness integration test plan for Multus, SR-IOV CNI, SR-IOV Network Device Plugin, Topology Manager, CMK, NFD

Microservice	ICN	OPENNESS	Difference	Next
MULTUS	1. Apply the bridge-network.yaml**[1]**. 2. Create Multus-deployment.yaml**[2]** with two bridge interfaces. 3. Exec follow command to check if the "net1" interface was created. kubectl exec -it $deployment_pod -- ip a [1]https://github.com/onap/multicloud-k8s/blob/9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/tests/_common.sh#L856 [2]https://github.com/onap/multicloud-k8s/blob/9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/tests/_common.sh#L873	1. Apply the macvlan-network.yaml. 2. Create a pod with macvlan annotation. 3. Verify the "net1" interface was configured in the deployed pod. Link: openness multus usage: https://github.com/open-ness/specs/blob/master/doc/enhanced-platform-awareness/openness-sriov-multiple-interfaces.md#multus-usage	• Different network types used for testing. ICN is using the 'bridge' type, OPENNESS is 'macvlan'.	• Update ICN test case with verifying macvlan network type.
SRIOV CNI SRIOV NETWORK DEVICE PLUGIN	1. Apply the sriov-network.yaml**[1]**. 2. Check if the Ethernet adapter version is equal to "XL710". 3. Create a pod**[2]** with the sriov annotation field and the sriov resource requested. 4. Verify the the deployed pod status. kubectl get pods $pod | awk 'NR==2{print $3}' 5. Check the current sriov resource allocation status**[3]**. [1]https://github.com/onap/multicloud-k8s/blob/9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/deployment_infra/playbooks/sriov-nad.yml#L1 [2]https://github.com/onap/multicloud-k8s/blob/9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/tests/sriov.sh#L32 [3]https://github.com/onap/multicloud-k8s/blob/9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/tests/sriov.sh#L68	1. Apply the sriov-network.yaml 2. Create a pod with the sriov annotation field and the sriov resource requested. 3. Verify the "net1" interface was configured in the deployed pod. Link: Openness sriov usage: https://github.com/open-ness/specs/blob/master/doc/enhanced-platform-awareness/openness-sriov-multiple-interfaces.md#usage	• Beside deploying the pod with sriov interface, ICN checks the current allocated sriov resource status.	• ICN has a more comprehensive testing and It covers openness test scope. So the ICN test case remains unchanged.

NFD	Verify NFD by setting pod.yaml with ' affinity' field.
	...
	apiVersion: v1
	kind: Pod
	metadata:
	  name: $pod_name
	spec:
	  affinity:
	  nodeAffinity:
	  requiredDuringSchedulingIgnored DuringExecution:
	  nodeSelectorTerms:
	  - matchExpressions:
	  - key: "feature.node.kubernetes. io/kernel-version.major"
	  operator: Gt
	  values:
	  - '3'
	  containers:
	  - name: with-node-affinity
	  image: gcr.io/google_containers/pause: 2.0
	...
	
	Link:
	
	KUD test script:
	https://github.com/onap/multicloud-k8s /blob/master/kud/tests/nfd.sh
		Verify NFD by setting pod.yaml with 'nodeSelector' field.
		…
		apiVersion: v1
		kind: Pod
		metadata:
		  labels:
		  env: test
		  name: golang-test
		spec:
		  containers:
		  - image: golang
		  name: go1
		  nodeSelector:
		  feature.node.kubernetes.io/cpu-pstate. turbo: 'true'
		…
		
		Link:
		
		Openness nfd usage:
		https://github.com/open-ness/specs/blob /master/doc/enhanced-platform-awareness /openness-node-feature-discovery.md#usage
			• Node affinity is conceptually similar to nodeSelector. it allows you to constrain which nodes your pod is eligible to be scheduled on, based on labels on the node.
			
			• Both tests are roughly the same like each other, ICN specifies 'affinity' to check if the NFD is effective, and OPENNESS uses the 'nodeSelector' field.
				• Add a check condition for label 'feature.node. kubernetes.io/cpu-pstate. turbo: 'true''.

CMK	NIL Link: CMK official validate solution: https://github.com/intel/CPU-Manager-for-Kubernetes/blob/master/docs/operator.md#validating-the-environment Liang's patch: https://gerrit.onap.org/r/c/multicloud/k8s/+/102311	1. Create a pod that can be used to deploy applications pinned to a core. Link: Openness CMK usage: https://github.com/open-ness/specs/blob /master/doc/enhanced-platform-awareness /openness-dedicated-core.md#usage	• CMK's integration is under way in ICN. So ICN doesn't provide a test case now.	• NIL
Topology Manager	NIL Link: Topology Manager limitation: https://kubernetes.io/docs/tasks /administer-cluster/topology-manager /#known-limitations	1. Create a pod with guaranteed(requests equal to limits) QoS class. 2. Check in kubelet's logs on you node (journalctl -xeu kubelet). Link: Openness TM usage: https://github.com/open-ness/specs/blob /master/doc/enhanced-platform-awareness /openness-topology-manager.md#usage	• Not implement Topology Manager at ICN. So ICN doesn't provide test case now.	• Dependence on k8s version.

Add more realistic test cases for platform related micro-services

Microservice	Test cases in KUD	Test cases to be added
Multus	1. Enable Multus AddOn support. 2. Verify the minion interface network type ・bridge**[1]** ・ovn4nfv**[2]** ・sr-iov**[3]** **[1]**https://github.com/onap/multicloud-k8s/blob /9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/tests/_common.sh#L856 **[2]**https://github.com/onap/multicloud-k8s/blob /9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/tests/_common.sh#L1053 **[3]**https://github.com/onap/multicloud-k8s/blob /9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/tests/sriov.sh#L32	1. Add more k8s-cni**[1]** type verification: ・macvlan ・ipvlan ・ptp **[1]** https://github.com/containernetworking /plugins
SR-IOV CNI SR-IOV Network Device Plugin	1. Enable SR-IOV cni and network device plugin AddOn support. 2. Check VF allocated status. ・Verify single VF allocated**[1]** **[1]**https://github.com/onap/multicloud-k8s/blob /9c63ce2a7b2b66b3e3fce5d1f553f327148df83f/kud/tests/sriov.sh#L47	1. Add multi-VF allocated verification ・Verify multiple VF allocated

NFD	1. Enable NFD AddOn support. 2. Using the "affinity"**[1]** field verifies If NFD is effective or not. ・ Using 'Gt' operator to check kernel version. **[1]**https://github.com/onap/multicloud-k8s/blob/master/kud/tests/nfd.sh#L27	1. Enhance the "affinity" verification method. ・Add different kinds of operator, e.g. In, Not In, Exists, DoesNotExist, Lt. ・Add multiple "matchExpressions" verification. ・Add multiple "nodeSelectorTerms" verification. 2. Add a “nodeSelector” field to verify.
CMK	CMK is not integrated into KUD yet.	It's going to be added the patch below: https://gerrit.onap.org/r/c/multicloud/k8s/+/102311

Task List

- Create Ansible scripts to create building environment, build microservices' docker images and push them to docker repository
- Create helm charts to run microservice in ONAP4K8s

Application

- TBD

ICN Requirements for adding EAA support for geo-distributed producing and consuming applications

BACKGROUND

Cloud native applications usually use microservice architecture. It means the application will contain multiple micro-serivces like Figure 1. This application consists of four micro-services (s1, s2, s3, s4). And s1 communicates with s2, s2 communicates with s3 and s3 communicates with s4. s1 is an user facing micro-service. s1 and s2 are expected to be deployed together. s2 is stateful and hence needs to communicate with other s2.

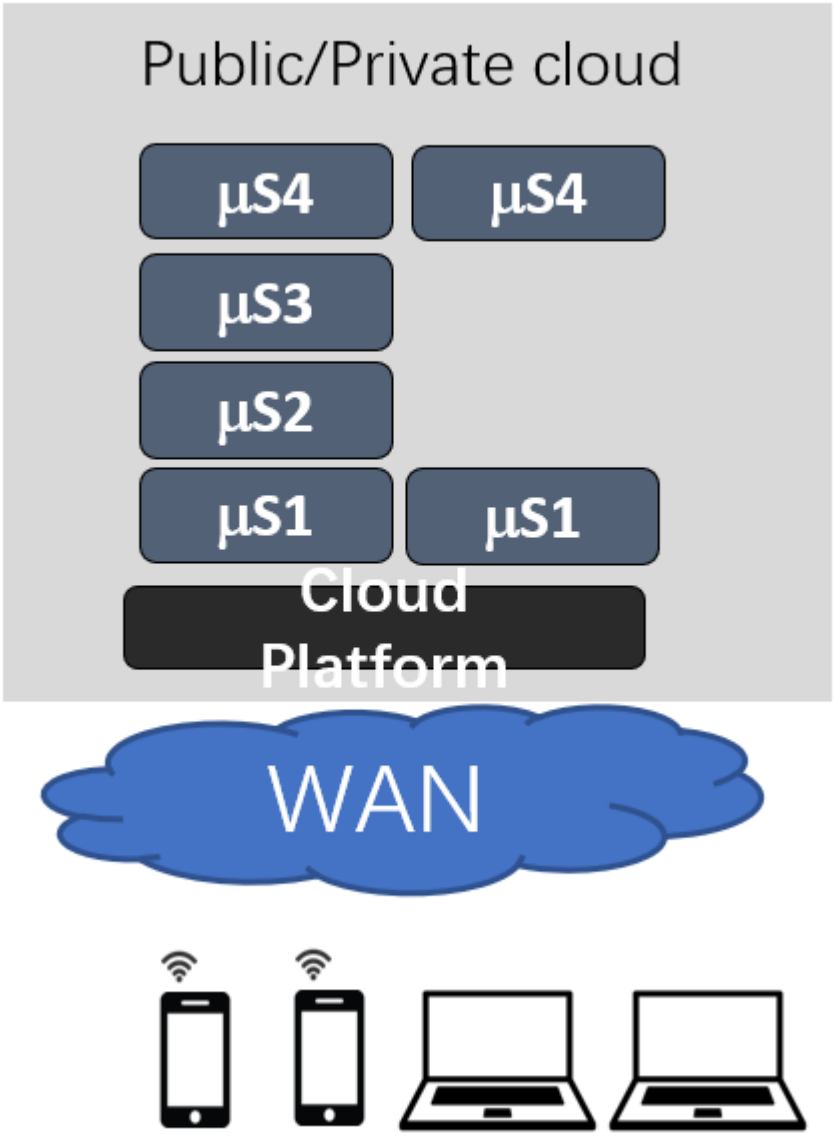

Figure 1 Centralized Application

When it comes to edge computing, some micro-services will be deployed on the edge clouds and some micro-services will be deployed on the central cloud like Figure 2. s1 and s2 are deployed on the edge cloud. s3 and s4 are deployed on the central cloud. Thus the application for edge computing is geo-distributed in nature.

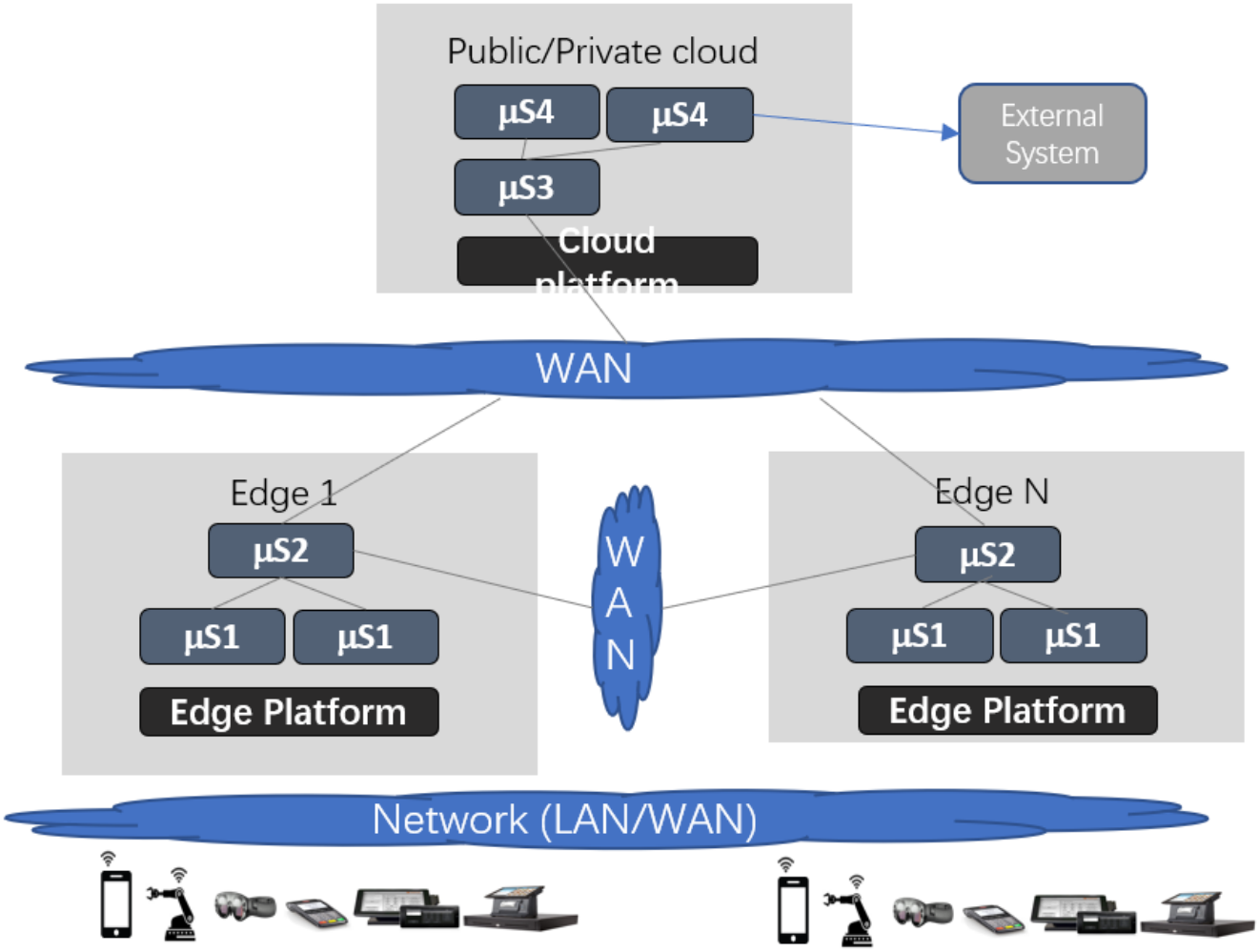

Figure 2. Distributed Application

ICN (which includes EMCO – formerly ONAP4K8s) is to show multiple clusters as one as far as the application life cycle is concerned as applications are becoming geo-distributed. In EMCO, we have a concept called 'Logical Cluster" which is an abstracted cluster across multiple K8s clusters as Figure 3.

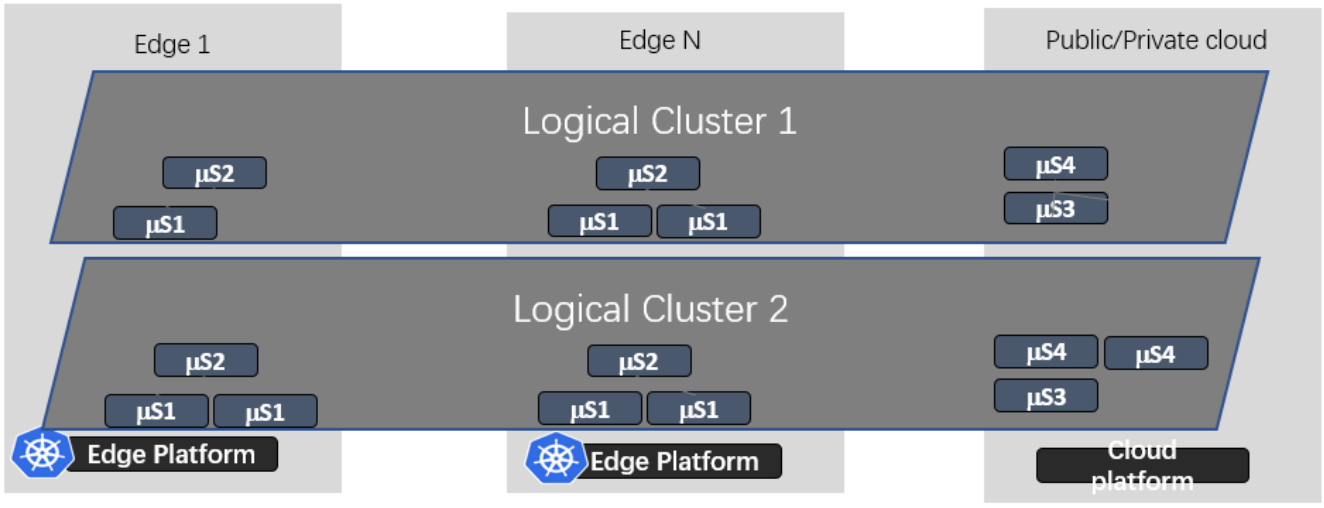

Figure 3 Logical Cluster

REQUIREMENT

EAA provides application/service registration and authentication in openness. For now eaa only supports single cluster applications and doesn't support geo-distributed, multi-cluster applications which are typically edge applications. To support geo-distributed applications, eaa needs to support application /service registration and authentication on different edge clouds which are kubernetes clusters in network edge. For example,

- If creating one EAA for every tenant (logical cluster): micro-services on different edge clouds which are kubernetes clusters should be able to communicate with each other by registering the services to the EAA and consuming the services from the EAA on different edge clouds. For example: s2 is stateful and needs to communicate with other s2 on different edge clouds to synchronize the states.
- If creating one EAA for every kubernetes cluster, EAAs need to synchronize the states because EAAs are stateful: The certs of EAAs on different edge clouds are signed by different Root CAs which are generated by Openness ansible scripts. What's more, producing application and consuming application will get certs from EAA and those certs are signed by EAA's certs. And this will cause the producing application and consuming application on different edge cloud can't communicate with each other because their certs are on different certificate chains. To solve this issue, the certs of EAAs should be signed by the same orchestrator. For example, ICN DCM (Distributed Cloud Manager) can take this role: https://wiki.onap.org/pages/viewpage.action?pageId=76875956