ICN Test Document

Introduction

ICN Pod Topology

Jenkins Information

Akraino community has a public Jenkins cluster. ICN leverages the Akriano public Jenkins to run CI jobs. While the CD jobs run in our private Jenkins cluster.

We have the following Jenkins slave nodes joined Akriano Jenkins. ICN CI jobs are supposed to be scheduled to our slave nodes by label icn-dev.

Slave Infomation			**Server Infomation**
Slave Name	Labels	Slave Root	Server Info
prd-ubuntu-dev-44c-64g	icn-dev	/home/jenkins/akraino/slave_root	pod14-node1

To add more Jenkins slave nodes, please follow the akriano jenkins guide

To setup private jenkins, please refer to the README.md under icn/ci/

The private jenkins cluster is setup on pod14-node2. We can visit the jenkins with the node ip address: http://10.10.140.22:8080/

Currently we support only AIO private Jenkins.

Akarino Test Group Information

not applicable

Overall Test Architecture

Test Architecture

We support the following jobs

CI job

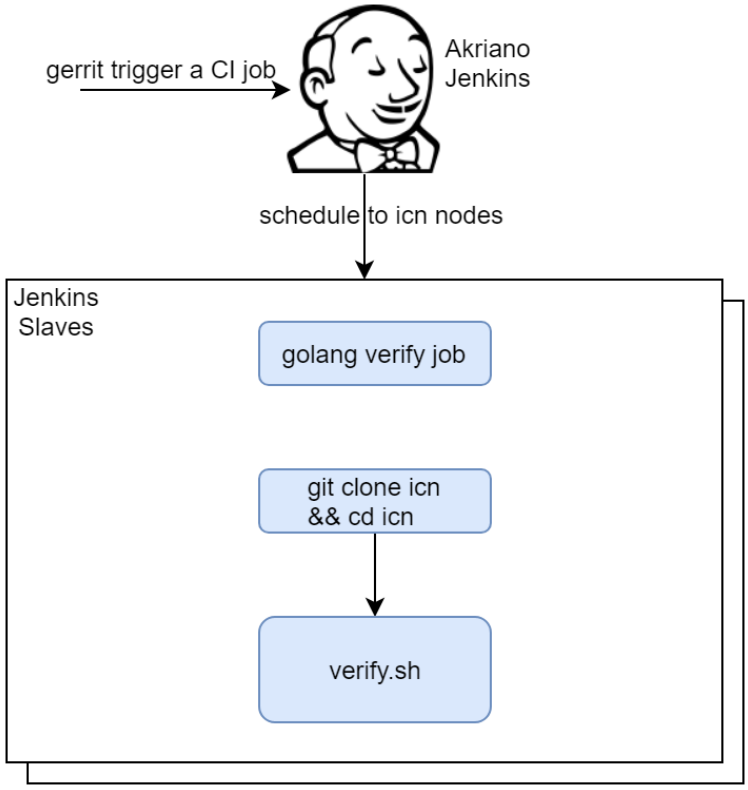
- triggered by gerrit patch creation/update.
- The job runs verify.sh under icn project. The verify.sh currently has integrated the golang test and bashate test.
- Post +1/-1 for gerrit patch if the build succeeds/fails
- Upload the job log to Nexus server in post-build actions

CD job for test

- triggered daily automatically (We can also trigger it manually)
- Run a make command, which creates VM(s) and deploys ICN components on the VM(s)
- Upload the job log to Nexus server in post-build actions

CI jobs detail

Update the verify.sh can update the CI job content.



CD job detail

We have the following steps for CD job:

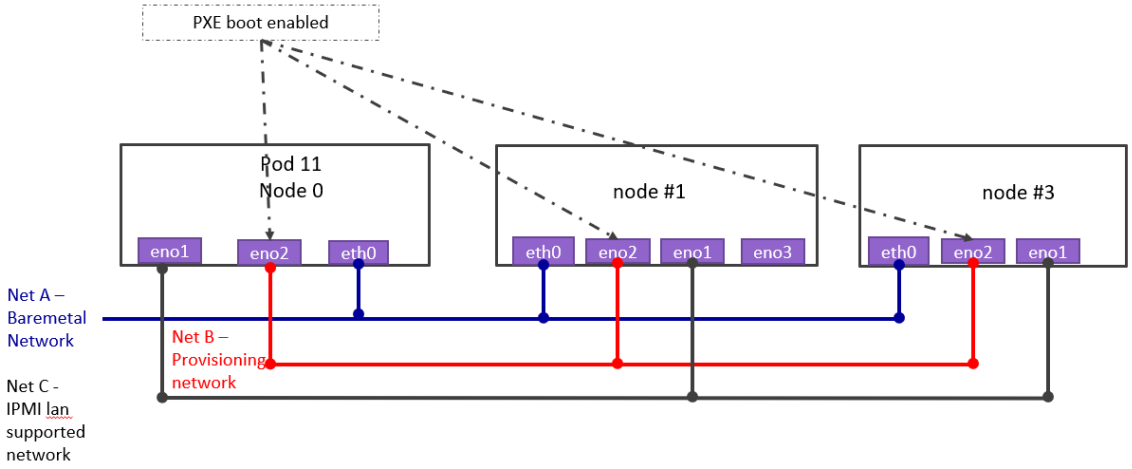
1. On our private Jenkins node, we provision a VM by vagrant. A Vagrantfile which defines the VMs properties is needed. We can define many VM properties in the Vagrantfile:
    - VM hostname
    - VM memory 64G, cpu 16, disk 300GB
2. Login to the VM and run 'make verifier' which installs the components in the VM
3. We destroy the VM as the last step of the job

Test Bed

Pod Topology

ICN Master Baremetal Deployment Verifier

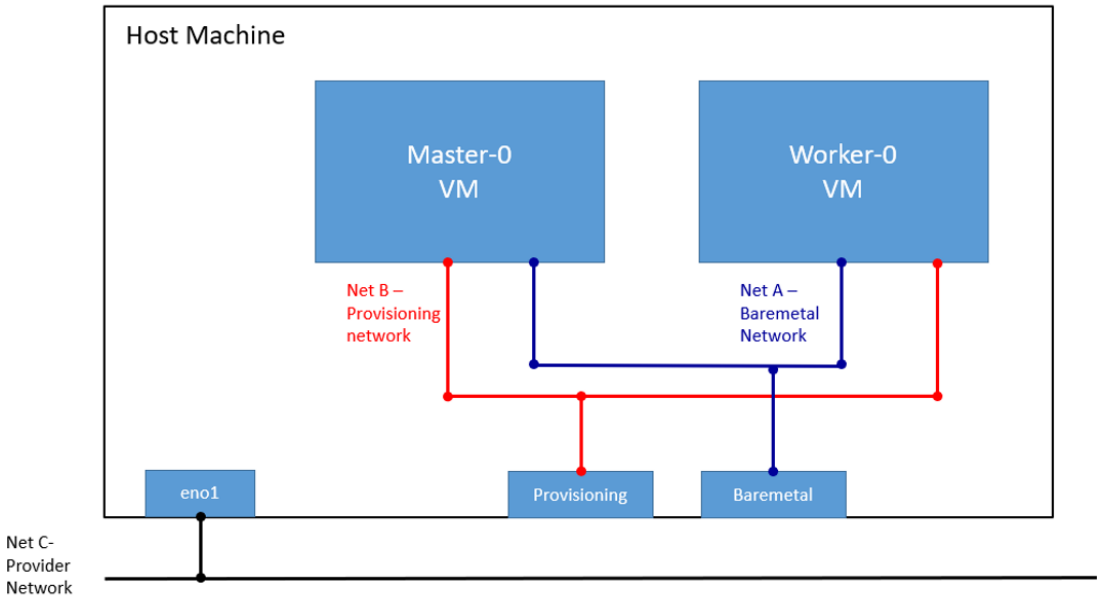
CD Topology - ICN Master Bare Metal Deployment Verifier



Net A – Baremetal Network, lab networking for SSH, It is used as the control plane for Kubernetes, used by OVN and Flannel for the overlay networking with Internet access
Net B(Internal networking) – Provisioning Networking used by Ironic to do inspection(Will be having DHCP server)
Net C(Internal networking) – IPMI lan to do IPMI protocols for the OS provisioning

ICN Master Virtual Deployment Verifier

CD Topology - ICN Master Virtual Deployment Verifier



Baremetal deployment

Hostname	CPU Model	Memory	BMC Firmware	Storage	1GbE: NIC#, VLAN, (Connected extreme 480 switch)	10GbE: NIC# VLAN, Network (Connected with IZ1 switch)	40GbE: NIC#

Jump	Intel 2xE5-2699	64GB	1.46.9995	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)	
node1	Intel 2xE5-2699	64GB	1.46.9995	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)	
node2	Intel 2xE5-2699	64GB	1.46.9995	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)	IF4: SRIOV

Virtual deployment

Hostname	CPU Model	Memory	Storage	1GbE: NIC#, VLAN, (Connected extreme 480 switch)	10GbE: NIC# VLAN, Network (Connected with IZ1 switch)
node1	Intel 2xE5-2699	64GB	3TB (Sata) 180 (SSD)	IF0: VLAN 110 (DMZ) IF1: VLAN 111 (Admin)	IF2: VLAN 112 (Private) **VLAN 114** (Management) IF3: VLAN 113 (Storage) **VLAN 1115** (Public)

Test Framework

All components are tested with end-to-end testing

Traffic Generator

Containerized packet generator is developed for traffic generator testing in cFW

Test description

Testing

CI Testing:

Bashate:

bashate test is to check the shell scripts code style. i.e. Trailing Whitespace. We find all files with suffix '.sh' and run bashate against the files. './cmd /bpa-operator/vendor/' directory is excluded.

Golang testing:

BPA Operator:

- The BPA operator has unit tests using the go framework. The unit tests check the following;
  - Job is created with the right job name for KUD installation.
  - The job metadata has the right cluster name
  - Expected error is produced when a host with the specified MAC address is not found
  - Expected error is produced when no dhcp lease is found for the specified host

BPA Rest Agent:

- Currently, automated unit tests are implemented using the Go testing framework.

CD Verifier(end-to-end testing):

All the test case are tested as follows:

Metal3:

Metal3 verifier will check all the servers are provisioned, Metal3 verifier check the status of the Baremetal servers for every 60 second for the provisioning status.

BPA Operator:

BareMetal host Provisioning

- The bpa_verifier.sh script get the MAC addresses and IP addresses of the 2 VMs provisioned by metal3, then creates a fake DHCP lease file using the IP address and MAC address information. It also creates a provisioning CR using the MAC address information
- The script the creates an ssh secret key using the ssh keys of the test host, applies the the provisioning CR
- The script busy loops till the KUD installation job completes or fails. If it completes successfully, it does a curl command using the authentication info of the new cluster to confirm if it was successful or not. On completing all the steps, it does a teardown where it deletes everything it created.

BPA Operator - Virtlet VM Provisioning

- Virtlet VM provisioning is tested as part of the 'verify_nestedk8s' testcase. K8s is first launched with Virtlet using KuD scripts after the prerequisite packages are installed.
- Next, BPA operator and multicloud-k8s docker images are built and BPA operator scripts including the provisioning_crd are deployed. Then, the Virtlet VM E2E script is launched which does the following:
- It creates a new flannel network definition for assigning mac address to VMs, creates test Virtlet VM, creates a provisioning CR for the same mac address. BPA operator then provisions the Virtlet VM by initiating the KuD installer job which installs K8s in the Virtlet VM.

BPA Rest Agent

- Test script, e2e_test.sh, creates dummy image file, creates test JSON file, checks bpa rest agent status, issues POST, GET, and PATCH requests sequentially.
- Next, e2e_test.sh checks uploaded MinIO image object size, and calls DELETE.
- If the script fails at any point then verification was unsuccessful.

Kubernetes Deployment (KuD)

KuD has test cases to verify if the add-ons are running correctly. All the test cases can be found in tests directory in the multicloud-k8s project. For each of these, we bring up the deployment that is specific to the addon, perform add-on specific actions on the pod related to the deployment

Multus:

- Multus CNI is a container network interface (CNI) plugin for Kubernetes that enables attaching multiple network interfaces to pods. This is accomplished by Multus acting as a "meta-plugin", a CNI plugin that can call multiple other CNI plugins.
- A 'NetworkAttachmentDefinition' is used to set up the network attachment, i.e. secondary interface for the pod.
- A pod is created with requesting specific network annotations with bridge CNI to create multiple interfaces. When the pod is up and running, we can attach to it to check the network interfaces on it by running *ip a* command

Virtlet:

- Virtlet is a Kubernetes runtime server which allows you to run VM workloads, based on QCOW2 images.
- We create a Virtlet VM pod-spec file adhering to the standards for virtlet to create a VM in a K8S env.
- The pod spec file is applied to bring up Virtlet deployment and make sure it is running. We attach to the pod and test to make sure the VM is running fine by connecting to it and checking details.

OVN4NFV:

- We use the Multus CNI container to create multiple ovn interfaces using OVN.
- After the pod is up and running we will be able to attach to the pod and check for multiple interfaces created inside the container.

Node feature Discovery

- Node feature discovery for Kubernetes detects hardware features available on each node in a Kubernetes cluster and advertises those features using node labels.
- Create a pod with specific label information in the case the pods are scheduled only on nodes whose Major Kernal version is 3 and above. Since the NFD Master and worker Daemonset is already running, the master has all the label information about the nodes which is collected by the worker.
- If the O.S version matches, the PoD will be scheduled and up. Otherwise, the Pod will be in a pending state in case there are no nodes with matching labels that are requested by the pod

SRIOV

- The SRIOV network device plugin is Kubernetes device plugin for discovering and advertising SRIOV network virtual functions (VFs) in a Kubernetes host.
- We first determine which hosts are SRIOV capable and install the drivers on them and run the DaemonSet and register Network attachment definition
- On an SRIOV capable hosts, we can get the resources for the node before we run the pod. When we run the test case, there is a request for a VF from the pod, therefore the number of resources for the node is increased.

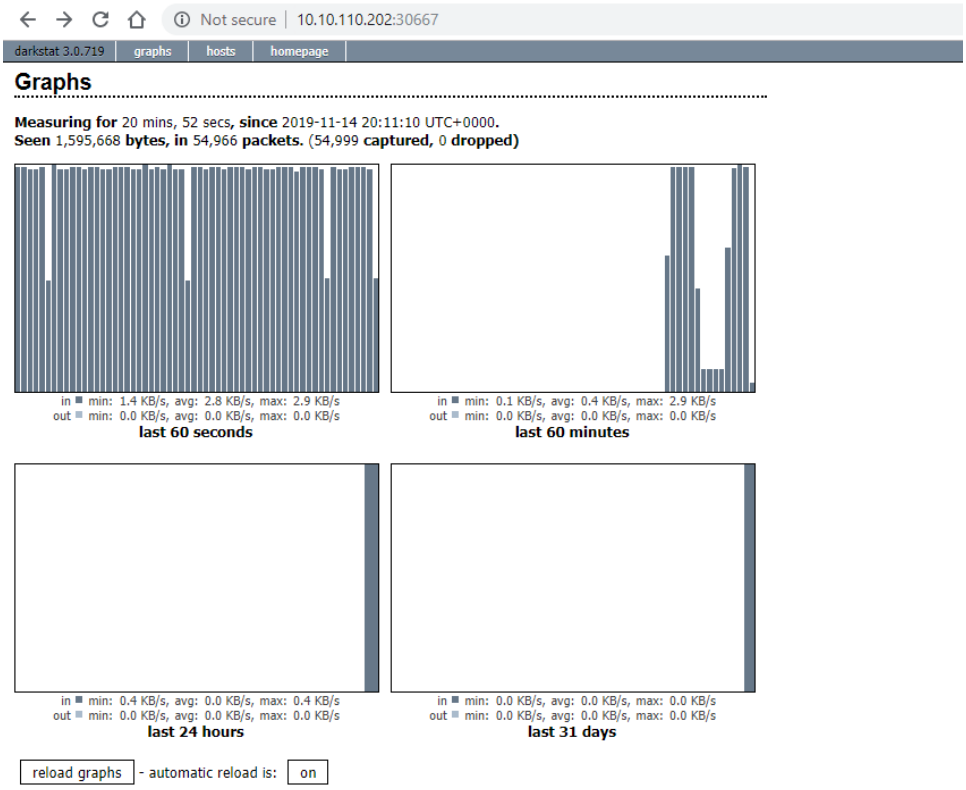
ONAP4K8s:

- ONAP4K8s testing check the health connectivity ONAP Micro service, once it is installed

cFW:

- Cloud Native FW having multiple components such packetgen generator, sink and cFW
    - Packet generator: Sends packets to the packet sink through the firewall. This includes a script that periodically generates different volumes of traffic inside the container
    - Firewall: Reports the volume of traffic passing though to the ONAP DCAE collector.

- Traffic sink: Displays the traffic volume that lands at the sink container using the link node port through your browser and enable automatic page refresh by clicking the "Off" button. You can see the traffic volume in the charts.



EdgeX Foundry:

EdgeX Foundry helm chart are installed through ONAP in the edge location. Test case ensure that all the EdgeX Framework containers are up and running

SDWAN controller:

- Create SDWAN CNF and a normal pod with additional interfaces created by OVN CNI plugin, and verify ping is workable between these 2 pods through the OVN interfaces
- Disable Allow-Ping rule of SDWAN CNF through Command Rest API call, then verify ping is not work between these 2 pods through the OVN interfaces
- Enable Allow-Ping rule of SDWAN CNF through Command Rest API call, then verify ping is workable again between these 2 pods through the OVN interfaces

CI logs:

The gerrit comments contains the CI log url. All the CI logs are under this folder ICN : https://jenkins.akraino.org/view/icn/job/icn-master-verify/

Latest CI logs

CD Logs:

ICN Master Baremetal Deployment Verifier

ICN Master Baremetal Deployment for Hardware verification

ICN Master Baremetal Deployment Virtlet nested K8s Verifier

ICN Master Virtual Deployment Verifer

ICN Master Virtual Deployment Virtlet nested K8s Verifier

Test Dashboards

All the testing results are in logs

Additional Testing

not applicable

Bottlenecks/Errata

not applicable