Kubernetes Bare-Metal Install and Configuration

This document is divided into two sections. The first section is focusing on the baremetal node installer & the second section on the Kubernetes/software installers.

Bare-Metal Node Provisioner

This section describes how to set up and configure a bare metal provisioner for RI-2. These investigations could be used as an input to the wiki of RI-2.

Scope

Provisioning is the operation of installing OS on a given infrastructure before the software stack can be installed on them. For the purpose of these investigations, a former OPNFV baremetal provisioner XCI, now referred to as Cloud Infra Automation Framework and hosted by Nordix Labs [1] is used.

This framework uses Bifrost for provisioning virtual and baremetal nodes and supports online and offline deployments. Bifrost is a set of Ansible playbooks that automates the task of deploying a base image onto a set of known hardware using ironic [2].

Lab Requirements

The lab is hosted in OPNFV and is setup in accordance with the lab principles/requirements defined in CNTT RI-2 Chapter 3.

Provisioner Requirements

Before initiating a deployment, two configuration templates, referred to as POD Descriptor File (PDF) and Installer Descriptor File (IDF) in OPNFV terminology need to be defined. Both PDF and IDF files are modelled as yaml schema.

A PDF is a hardware configuration template that includes hardware characteristics of the jumphost node & the set of compute/controller nodes. For each node, the following characteristics should be defined:

- CPU, disk & memory information
- Remote management parameters
- Network interfaces list including name, MAC address, IP address, link speed

IDF extends the PDF with the network information required by the installer. All the networks along with possible VLAN, DNS and gateway information should be defined here.

Deployment

After ensuring that the lab and provisioner requirements are met, generate SSH keypair, add user to the sudo group & have passwordless sudo enabled. After this the deployment can be initiated by cloning the repo, navigating to the engine directory & running the deploy command

> *git clone https://gerrit.nordix.org/infra/engine.git*
>
> *cd engine/engine*
>
> *./deploy.sh -o <OStype>-p file:///<pdf.yaml> -i file:///<idf.yaml> -l provision*

Currently, Ubuntu 18.04 & CentOS 7.8 are supported (default Ubuntu 18.04). Support for other system can be added as well depending on the requirements.

The engine supports offline deployment as well. <Steps to follow>

After the successful completion of the deployment, one needs to setup host networking for K8s, etc. to be able to run software provisioning tooling from CNF Testbed or Intel's BMRA playbooks to configure and install k8s (& other plugins) on the provisioned nodes (e.g. creating VLAN's, setting up internet connectivity, etc. – left to the choice of operator/vendor).

References

1. https://docs.nordix.org/submodules/infra/engine/docs/user-guide.html#framework-user-guide
2. https://github.com/openstack/bifrost

Bare-Metal Software Provisioner

CNF Testbed tooling

After provisioning nodes & configuring networking, one can run software provisioning tooling from CNF Testbed to setup and configure k8s.

Intel BMRA tooling

After provisioning nodes & configuring networking, download Intel container kit repo & update the inventory, etc with your desired configuration (refer to Prepare the BMRA software section below for more details).

Then run the following command to provision k8s & other plugins/features

> *sudo docker run --rm -v $(pwd):/bmra -v ~/.ssh/:/root/.ssh/ -ti bmra-install:centos ansible-playbook -i /bmra/inventory.ini /bmra/playbooks/cluster.yml*

The above BMRA installation works if you have provisioned your nodes using the BM provisioner described above. For pre-provisioned nodes, refer to the section below.

BMRA Installation

The following is based on configuration and installation outside of OPNFV Intel Lab.

The OS used for Jump, Master and Worker nodes is **CentOS 7.8.2003 (3.10.0-957.12.2.el7.x86_64)**

Prepare worker node

Prior to installing BMRA, log on the worker node and check the hardware configuration. This information is used when configuring BMRA later.

Start by installing pciutils, which is used by Ansible and needed when gathering information:

- `$ yum install pciutils`

CPU:

- `$ lscpu`
- Note down the number of cores and the enumeration method used for cores/threads.

Interfaces / PFs:

- `$ ip a`
  - Note down interfaces to be used with SR-IOV
- `$ lspci | grep Eth`
  - Check what PFs are available
- `$ lspci -ns <bus:device.function>`
  - Get the vendor and device IDs, e.g. 8086:1572

Prepare BMRA

Start by installing tools needed for running the BMRA playbook:

```
$ yum update
$ yum install git epel-release python36 python-netaddr
$ yum install python-pip
$ pip install ansible==2.7.16 jmespath
$ pip install jinja2 --upgrade
```

Prepare the BMRA software:

```
$ git clone https://github.com/intel/container-experience-kits.git
$ cd container-experience-kits/
$ git checkout <tag>
- v1.4.1 (If using Kubernetes 1.16)
- v1.3.1 (If using Kubernetes 1.15)
- v1.2.1 (If using Kubernetes 1.14)
$ git submodule update --init
$ cp examples/inventory.ini .
$ cp -r examples/group_vars examples/host_vars .
```

Update `inventory.ini` to match the the hardware and size of deployment. A minimal setup can look as follows:

```
[all]
master1 ansible_host=<master_ip> ip=<master_ip>
node1 ansible_host=<worker_ip> ip=<worker_ip>

[kube-master]
master1

[etcd]
master1
```

```
[kube-node]
node1

[k8s-cluster:children]
kube-master
kube-node

[calico-rr]
```

Update `host_vars/node1.yml` (and any additional files depending on `inventory.ini`):

- `sriov_enabled` - Change to true if VFs should be created during installation
- `sriov_nics` - Update with interface names (PF) from the node. Number of VFs and driver can be changed too
- `vpp_enabled` & `ovs_dpdk_enabled` - Disable one or both depending on need for a vSwitch. Using both might cause issues.
- `force_nic_drivers_update` - Set to false if SSH connection to machine uses interface with i40e or i40evf driver (otherwise connection to node is likely to be broken)
- `install_ddp_packages` - Set to false if DDP (Dynamic Device Personalization) should not be installed
- `isolcpus` - Change according to HW and need for isolated cores/treads. Also relevant for CMK configuration (see below)
- `sst_bf_configuration_enabled` - Consider setting this to false unless platform and HW supports it
- Additional changes can be done as needed

Update `group_vars/all.yml` according to hardware and capabilities:

- `cmk_hosts_list` - Update according to `inventory.ini`, e.g. only node1 for the above example
- `cmk_shared_num_cores` & `cmk_exclusive_num_cores` - Update according to available cores/threads and the number of isolated cores
  - Make sure sufficient cores/threads are isolated to support the shared and exclusive pools (see isolcpus above)
- `sriovdp_config_data` - Update according to the `sriov_nics` host_vars config (see above)
  - Updated might be needed depending on NICs. See details on Github
- `qat_dp_enabled` - Change to false if HW doesn't support QAT, either through chipset of PCI addon
- `gpu_dp_enabled` - Change to false if not supported on HW
- `tas_enabled` - Can be set to false if not needed
- `tas_create_policy` - Set to false if TAS is not enabled
- `cluster_name` - Can be changed to something more specific than "cluster.local"

Install BMRA

Once the necessary files have been updated, BMRA can be installed

- Consider creating a `tmux` or `screen` session to prevent installation issues due to disconnects
- Run the installer: `ansible-playbook -i inventory.ini playbooks/cluster.yml`

Post BMRA Install

Once installation is complete, you can decide if you will access the cluster from the master node, of from the jump/installer node.

- Using Master: SSH to the master node, and check status using `kubectl get nodes`
- Using Install/Jump: Install Kubectl, fetch Kubeconfig and set environment variable
  - Installing Kubectl: https://kubernetes.io/docs/tasks/tools/install-kubectl/
  - Fetch Kubeconfig from Master node: `scp <master_ip>:.kube/config kubeconfig`
  - Set environment: {{export KUBECONFIG=${PWD}/kubeconfig}}
- Test Kubectl using `kubectl get nodes`