**Installing NON-RT-RIC by Kubernetes**

The non-RT RIC is a type of controller. It helps control and improve RAN elements and resources. It includes an AI/ML workflow for model training and updates. It also provides policy-based guidance for applications and features in near-RT RIC.

**Primary Goals:**

* The primary goal of Non-RT RIC is to support intelligent RAN optimization by providing policy-based guidance, ML model management and enrichment information to the near-RT RIC function so that the RAN can optimize, e.g., RRM under certain conditions.
* It can also perform intelligent radio resource management function in non-real-time interval (i.e., greater than 1 second).
* Non-RT RIC can use data analytics and AI/ML training/inference to determine the RAN optimization actions for which it can leverage SMO services such as data collection and provisioning services of the O-RAN nodes.
* Non-RT-RIC will define and coordinate rApps (Non-RT-RIC applications) to perform Non-RT-RIC tasks.
* Non-RT-RIC will host the A1 interface (between NONRTRIC & near-RT RICs )
* Non-RT-RIC will also host the new R1 interface (between rApps and SMO/NONRTRIC services)

**Prerequisites:**

1. kubernetes v1.19 +
2. Docker and docker-compose (latest)
3. Helm 3
4. Minimum 8CPU, 16GB RAM and 50GB Disk.

To deploy NON-RT-RIC, first we need to prepare a perfect environment in Kubernetes. In the next step, we will show the Kubernetes installation step by step in both single node and multimode cluster.

**Kubernetes:**

Kubernetes is a powerful open-source container orchestration platform used for automating the deployment, scaling, and management of containerized applications. It was developed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes provides a way to manage containerized applications across multiple hosts, ensuring that applications are running and available to users at all times.

Kubernetes has several flavors like kubeadm, minikube, kinds and so on.

**Simple Single-node Kubernetes Cluster via kubeadm on Ubuntu 20.04 & 22.04**

**1.** **Install general dependencies**

sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl

**2.** **Install containerd:**

Although we have a few container runtimes to choose from, we’re going with containerd. Before we install containerd, we’ll create its configuration file.

curl -fsSLo containerd-config.toml \

https://gist.githubusercontent.com/oradwell/31ef858de3ca43addef68ff971f459c2/raw/5099df007eb717a11825c3890a0517892fa12dbf/containerd-config.toml

sudo mkdir /etc/containerd

sudo mv containerd-config.toml /etc/containerd/config.toml

Without delay, you can install containerd from their official GitHub repository as recommended using the following commands:

curl -fsSLo containerd-1.6.14-linux-amd64.tar.gz \

https://github.com/containerd/containerd/releases/download/v1.6.14/containerd-1.6.14-linux-amd64.tar.gz

# Extract the binaries

sudo tar Cxzvf /usr/local containerd-1.6.14-linux-amd64.tar.gz

# Install containerd as a service

sudo curl -fsSLo /etc/systemd/system/containerd.service \

https://raw.githubusercontent.com/containerd/containerd/main/containerd.service

sudo systemctl daemon-reload

sudo systemctl enable --now containerd

**3. Install runc:**

Installing runc from their official GitHub repository is the recommended way.

curl -fsSLo runc.amd64 \

https://github.com/opencontainers/runc/releases/download/v1.1.3/runc.amd64

sudo install -m 755 runc.amd64 /usr/local/sbin/runc

**4. Install CNI network plugins**

Install Container Network Interface network plugins from their official GitHub repository.

curl -fsSLo cni-plugins-linux-amd64-v1.1.1.tgz \

https://github.com/containernetworking/plugins/releases/download/v1.1.1/cni-plugins-linux-amd64-v1.1.1.tgz

sudo mkdir -p /opt/cni/bin

sudo tar Cxzvf /opt/cni/bin cni-plugins-linux-amd64-v1.1.1.tgz

**5. Forward IPv4 and let iptables see bridged network traffic**

You need to enable overlay and br\_netfilter kernel modules. Additionally, you need to allow iptables see bridged network traffic.

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

overlay

br\_netfilter

EOF

sudo modprobe -a overlay br\_netfilter

# sysctl params required by setup, params persist across reboots

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-iptables = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.ipv4.ip\_forward = 1

EOF

# Apply sysctl params without reboot

sudo sysctl –system

**6. Install kubeadm, kubelet & kubectl**

You need to ensure the versions of kubeadm, kubelet and kubectl are compatible.

# Add Kubernetes GPG key

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg \

https://packages.cloud.google.com/apt/doc/apt-key.gpg

# Add Kubernetes apt repository

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" \

| sudo tee /etc/apt/sources.list.d/kubernetes.list

# Fetch package list

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

# Prevent them from being updated automatically

sudo apt-mark hold kubelet kubeadm kubectl

**7. Ensure swap is disabled:**

You have to disable the swap feature because Kubernetes does not support it. See the GitHub issue regarding swap on Kubernetes for details.

# See if swap is enabled

swapon --show

# Turn off swap

sudo swapoff -a

# Disable swap completely

sudo sed -i -e '/swap/d' /etc/fstab

**For Single node cluster:**

Step 1-7 is common for single node and multi node cluster.

**8. Create the cluster using kubeadm**

It’s only a single command to initialise the cluster, but it won’t be very functional in single-node environments until we make some changes. Note that we’re providing “–pod-network-cidr” parameter as required by our CNI plugin (Flannel).

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

**Possible Error:** If there is any error then run

sudo systemctl daemon-reload

sudo systemctl enable --now containerd

**9. Configure kubectl**

To access the cluster, we have to configure kubectl.

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

**10. Untaint node**

We must untaint the node to allow pods to be deployed to our single-node cluster. Otherwise, your pods will be stuck in a pending state.

kubectl taint nodes --all node-role.kubernetes.io/master-

kubectl taint nodes --all node-role.kubernetes.io/control-plane-

note: If not worked

kubectl get nodes

kubectl taint nodes <ip-172-31-89-81> node-role.kubernetes.io/control-plane=:NoSchedule

**note:** change this “<….>” part according to your node ID.

**11. Install a CNI plugin**

For networking to function, you must install a Container Network Interface (CNI) plugin. With this in mind, we’re installing flannel.

Run this:

kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.ym>

**12. Install helm**

Helm is a package manager for Kubernetes. Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

To install our packages, we’re installing helm v3.

curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash

**13. Install chartmuseum**

curl https://raw.githubusercontent.com/helm/chartmuseum/main/scripts/get-chartmuseum | bash

**14. Install a CSI driver**

We need to install a Container Storage Interface (CSI) driver for the storage to work. We’ll install OpenEBS.

# Add openebs repo to helm

helm repo add openebs https://openebs.github.io/charts

kubectl create namespace openebs

helm --namespace=openebs install openebs openebs/openebs

**15. Install a test application**

To test the cluster, you can deploy WordPress. Note that we need to specify the storage class provided by our CSI.

# Add bitnami repo to helm

helm repo add bitnami https://charts.bitnami.com/bitnami

helm install wordpress bitnami/wordpress \

--set=global.storageClass=openebs-hostpath

**16. Connect master node to worker node**

Run this in master: out put of this command for joinning worker

kubeadm token create --print-join-command

output:

**example:** kubeadm join 172.31.89.81:6443 --token gnzyet.cuftjb0shinn193j --discovery-token-ca-cert-hash sha256:fdc65a3170e2caa573becf3dba31dcddaffc183e589def36a0d2a3b6e02ae3

Run that out put in worker node:

kubeadm join 172.31.89.81:6443 --token gnzyet.cuftjb0shinn193j --discovery-token-ca-cert-hash sha256:fdc65a3170e2caa573becf3dba31dcddaffc183e589def36a0d2a3b6e02ae3

**17. Open new terminal and run localhost:**

Open new terminal and run this for avoid localhost error:8080

kubectl proxy --port=8080

**Refference:** <https://kubernetes.io/docs/tasks/tools/>

**Now time to deploy NON RT-RIC**

**Preparations**

Download the the it/dep repository. At time of writing there is no branch for f-release so cloning can be made from master. Pls check if the branch exist before cloning from master.

**Clone repo:**

*git clone "https://gerrit.o-ran-sc.org/r/it/dep"*

**Configuration of components to install**

It is possible to configure which of nonrtric components to install, including the controller and a1 simulators. This configuration is made in the override for the helm package. Edit the following file

Edit override file

*<editor> dep/RECIPE\_EXAMPLE/NONRTRIC/example\_recipe.yaml*

The file shown below is a snippet from the override example\_recipe.yaml.

All parameters beginning with 'install' can be configured 'true' for enabling installation and 'false' for disabling installation.

For the parameters installNonrtricgateway and installKong, only one can be enabled.

There are many other parameters in the file that may require adaptation to fit a certain environment. For example hostname, namespace and port to message router etc. These integration details are not covered in this guide.

Editor override file

nonrtric:

installPms: true

installA1controller: true

installA1simulator: true

installControlpanel: true

installInformationservice: true

installRappcatalogueservice: true

installNonrtricgateway: true

installKong: false

installDmaapadapterservice: true

installDmaapmediatorservice: true

installHelmmanager: true

installOruclosedlooprecovery: true

installOdusliceassurance: true

volume1:

# Set the size to 0 if you do not need the volume (if you are using Dynamic Volume Provisioning)

size: 2Gi

storageClassName: pms-storage

volume2:

# Set the size to 0 if you do not need the volume (if you are using Dynamic Volume Provisioning)

size: 2Gi

storageClassName: ics-storage

volume3:

size: 1Gi

storageClassName: helmmanager-storage

...

...

...#

**Installation**

There is a script that packs and installs the components by using the helm command. The installation uses a values override file like the one shown above. This example can be run like this:

**Deploy Nonrtric**

*sudo dep/bin/deploy-nonrtric -f dep/nonrtric/RECIPE\_EXAMPLE/example\_recipe.yaml*

**Result of the installation**

The installation will create one helm release and all created kubernetes objects will be put in a namespace. This name is 'nonrtric' and cannot be changed.

Once the installation is done you can check the created kubernetes objects by using command kubectl.

Example : Deployed pods when all components are enabled:

Get Pods

>sudo kubectl get po -n nonrtric

NAME READY STATUS RESTARTS AGE

a1-sim-osc-0 1/1 Running 0 12m

a1-sim-osc-1 1/1 Running 0 10m

a1-sim-std-0 1/1 Running 0 12m

a1-sim-std-1 1/1 Running 0 10m

a1-sim-std2-0 1/1 Running 0 12m

a1-sim-std2-1 1/1 Running 0 3m57s

a1controller-64c5b7fc56-hjx6l 1/1 Running 0 12m

controlpanel-6bf7c4bf79-m6hgl 1/1 Running 0 12m

db-76d79cd769-mngm4 1/1 Running 0 12m

dmaapadapterservice-0 1/1 Running 0 12m

dmaapmediatorservice-0 1/1 Running 0 12m

helmmanager-0 1/1 Running 0 12m

informationservice-0 1/1 Running 0 12m

nonrtricgateway-677988d5c7-rfm88 1/1 Running 0 12m

odusliceassurance-cd5b6f568-q89r5 1/1 Running 0 12m

oruclosedlooprecovery-568f867b45-b6zld 1/1 Running 0 12m

policymanagementservice-0 1/1 Running 0 12m

rappcatalogueservice-687d69756c-lvwrg 1/1 Running 0 12m

After successful installation, control panel shows "No Type" as policy type as shown below.

A screenshot of a computer

Description automatically generated