**Near Real Time-RT RIC**

Requirements:

1. Ubuntu 20.04
2. 32 GB RAM, 16 CPU and minimum 30 GB Disk spaces.
3. Kubernetes -v1.16
4. Helm and base chart -v3.5.4

Steps:

1. Prepare a fresh Ubuntu 20.04 for near-RT RIC deployment.
2. Install Kubernetes, helm, and base chart for Near-RT RIC deployment.
3. Install near-RT RIC.
4. Compile and connect O-RAN E2 (e2-node) simulator from O-RAN SC simulator project
5. Deploy dms\_cli (for deploying xApps).
6. Compile, onboard and install hw-go xapp from O-RAN SC xApp project.

**1. Prepare a fresh Ubuntu 20.04 for near-RT RIC deployment:**

**2. Install Kubernetes, helm, and base chart for Near-RT RIC deployment:**

Installing Near Realtime RIC in RIC Cluster

(Note: Run all command in root user.)

# Check first which ubuntu version is ready for deployment.

#run command below.

*lsb\_release -a*

Example Output:

Getting and Preparing Deployment Scripts.

# Clone the ric-plt/dep git repository that has deployment scripts and support files on the target VM.

*git clone https://gerrit.o-ran-sc.org/r/ric-plt/ric-dep*

# Now run the command below and see ric directory added in locally.

*ls -a*

Example Output:

Deploying the Infrastructure and Platform Groups

# now Install Kubernetes, helm, Kubernetes-CNI and docker

*cd ric-dep/bin*

*./install\_k8s\_and\_helm.sh*

# It will take take times and once it done then check Kubernetes installed or not by this command. It will show Kubernetes system pods are running or not.

*Kubectl get pods -n kube-system*

Exanmple Output:

\*\*Note: make sure curl is installed. If not, then run this command to install curl: *apt install curl*

# Install chartmuseum into helm and add ric-common templates

*./install\_common\_templates\_to\_helm.sh*

Example Output:

# Now Create a txt file for separates the image pull from the public image registry and the actual installation. make sure to use same versions as in ../RECIPE\_EXAMPLE/example\_recipe\_oran\_f\_release.yaml

Run this command:

*<editor> versions.txt*

# Now, copy docker container file and save this file.

Versions.txt

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-a1:2.5.2*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-appmgr:0.5.6*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-dbaas:0.6.1*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-e2mgr:6.0.0*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-e2:6.0.0*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-rtmgr:0.9.1*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-submgr:0.9.3*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-vespamgr:0.7.5*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-o1:0.6.0*

*nexus3.o-ran-sc.org:10002/o-ran-sc/ric-plt-alarmmanager:0.5.13*

*nexus3.o-ran-sc.org:10002/o-ran-sc/it-dep-init:0.0.1*

*docker.io/prom/prometheus:v2.18.1*

*docker.io/kong/kubernetes-ingress-controller:0.7.0*

*docker.io/kong:1.4*

*docker.io/prom/alertmanager:v0.20.0*

# To see the RECIPE file run this command.

*cat ../RECIPE\_EXAMPLE/example\_recipe\_oran\_f\_release.yaml*

# Now pull the docker images from *nexus3.o-ran-sc.org*

*for i in `cat versions.txt`; do echo $i; docker pull $i; done*

**Note:** It will take time. We pulled this image before create Kubernetes pods because Kubernetes will create as following RECIPE\_EXAMPLE file. we pulled actual docker container file for near-rt ric deployment.

After the recipes are edited and helm started, the Near Realtime RIC platform is ready to be deployed, but first update the deployment recipe as per instructions in the next section.

3. Modify and Deploy near-RT RIC.

Now we can start the installation of near-RT RIC. Before that we need IP address of the particular node.

To see the IP address run the command below:

*ip a*

Example Output:

# Edit the recipe files ./RECIPE\_EXAMPLE/example\_recipe\_latest\_stable.yaml (which is a softlink that points to the latest release version). “example\_recipe\_latest\_unstable.yaml points to the latest example file that is under current development.

*vim ../RECIPE\_EXAMPLE/example\_recipe\_oran\_f\_release.yaml*

**Note**: vim is an editor

Example Output:

Deployment scripts support both helm v2 and v3. The deployment script will determine the helm version installed in cluster during the deployment.

4. Installing the RIC

After updating the recipe you can deploy the RIC with the command below. Note that generally use the latest recipe marked stable or one from a specific release.

*cd ric-dep/bin*

*./install -f ../RECIPE\_EXAMPLE/example\_recipe\_oran\_f\_release.yaml*

It will take time. After installation succeed we can see near RT RIC platform pods are actually running.

5. Checking the Deployment Status

Now check the deployment status after a short wait. Results similar to the output shown below indicate a complete and successful deployment. Check the STATUS column from both kubectl outputs to ensure that all are either “Completed” or “Running”, and that none are “Error” or “ImagePullBackOff”.

*Kubectl get pods -n ricplt*

Example Output:

helm list

Example output:

6. Checking Container Health

Check the health of the application manager platform component by querying it via the ingress controller using the following command.

*curl -v http://localhost:32080/appmgr/ric/v1/health/ready*

Example Output:

7. Compile and connect O-RAN E2 (e2-node) simulator from O-RAN SC simulator project

# First we will clone the repo of source code from git.

*git clone https://gerrit.o-ran-sc.org/r/sim/e2-interface*

# Now we will install prerequires to compile and connect E2 simulator.

*apt-get install cmake g++ libsctp-dev*

# Run this command to change the directory first and then modify Dockerfile:

*cd e2-interface/e2sim*

# Modify Docker file:

Edit CMD part and replace IP address which is E2 termination of the RIC side.

*sleep 100000000*

We execute into the container information invoke this command.

# Now start first step of the simulator compilation which is creating certain Debian packages which are then used in the docker stage of the compilation.

*mkdir build*

*cd build*

*cmake .. && make package && cmake .. -DDEV\_PKG=1 && make package*

# Copy the Debian directory.

*cp \*.deb ../e2sm\_examples/kpm\_e2sm/*

*cd ../e2sm\_examples/kpm\_e2sm/*

# Docker container creation

*docker build -t oransim:0.0.999 .*

*docker run -d --name oransim -it oransim:0.0.999*

# Now run the simulator.

*docker exec -ti oransim /bin/bash*

# Now check the simulator is running or not.

*kpm\_sim <IP> 36422*

**# Note:** Replace *<IP>* to run the simulator.

*kubectl get services -n ricplt*

Example output:

# Now copy IP from services in ricplt with 36422 port and use in previous step.

Example: *kpm\_sim 10.98.13.250 36422*

Once run that command we will see simulator is connected with near RT RIC. output will be like that:

Example output:

# To show that e2 simulator is connect with near RT RIC follow below steps:

# Copy e2 manager IP and run curl command:

Example of copying IP:

# Now run this command-

*curl -X GET http://<IP>:3800/v1/nodeb/states 2>/dev/null|jq*

Replace <IP> with e2 manager IP. Run this command in *ric-dep/bin* directory. And we will see e2 simulator connected wit near RT RIC.

**5. Deploy dms\_cli (for deploying xApps).**

The DMS component of the RIC platform is responsible for managing the Near-Real Time (NRT) data and metadata used by the platform, such as policy and configuration data. The "dms\_cli" tool is likely used to perform tasks such as querying, updating, and deleting data and metadata within the DMS using command-line instructions.

xApp onboarder provides a cli tool called dms\_cli to fecilitate xApp onboarding service to operators. It consumes the xApp descriptor and optionally additional schema file, and produces xApp helm charts.

# Before any xApp can be deployed, its Helm chart must be loaded into this private Helm repository

*docker run --rm -u 0 -it -d -p 8090:8080 -e DEBUG=1 -e STORAGE=local -e STORAGE\_LOCAL\_ROOTDIR=/charts -v $(pwd)/charts:/charts chartmuseum/chartmuseum:latest*

**Note:** run it in home directory.

# Set up the environment variables for CLI connection using the same port as used above.

*export CHART\_REPO\_URL=http://0.0.0.0:8090*

# Clone the app-manager repo

*git clone https://gerrit.o-ran-sc.org/r/ric-plt/appmgr -b f-release*

# Change the xapp\_onboarder directory.

*cd appmgr/xapp\_orchestrater/dev/xapp\_onboarder*

# If pip3 is not installed, install pip packages for python install.

*apt-get install python3-pip*

# If there is dms\_cli binary is already installed, then uninstall this first´using following command.

*pip3 uninstall xapp\_onboarder*

# Now install xapp\_onboarder.

*pip3 install ./*

# Now modify the permissions. The instruction are for python 3.6 and here we will install python 3.8

*ls -la /usr/local/lib/python3.8*

*chmod -R 755 /usr/local/lib/python3.8*

**6. Compile, onboard and install hw-go xapp from O-RAN SC xApp project.**

Here, Compile the hello world go and deploy it.

# Clone source code repo:

*git clone https://gerrit.o-ran-sc.org/r/ric-app/hw-go*

cd hw-go

# Build docker templates and *example.con:80* is a tag. It could be any url. It will run in local registry. This will be in later use.

*docker build -t example.com:80/hw-go:1.2 .*

# Now modify config-file.json for build connection with xapp onboard for deployment.

*vim config/config-file.json*

**Note:** edit config file:

“registry”: “*nexus3.o-ran-sc.org:10004”* to *“example.com:80”.*

*“*name”: *“hw-go”.*

“tag”: *“1.2”*

Example output:

Text

Description automatically generated

# Now make sure the xapp descriptor config file and the schema file at my local file system.

*dms\_cli onboard ./config/config-file.json ./config/schema.json*

Example Output:

# Install hw-go

*dms\_cli install hw-go 1.0.0 ricxapp*

Example Output:

When hw-go is installed then it will make e2 subscription to all e2 nodes that it will finds connected. Now we can see ricxapp is running and connected with e2 simulator.

# Run this command for check ricxapp pods are running or not.

*kubectl get pods -n ricxapp*

**Note:** Run it *ric-dep/bin* directory.