**Near RT RIC Installation**

Obtaining Deployment Scripts and Charts

git clone <http://gerrit.o-ran-sc.org/r/it/dep>

cd dep

git submodule update --init --recursive --remote

Generation of *cloud-init* Script:

sudo -i

cd tools/k8s/bin

./gen-cloud-init.sh

Updation of Version Combination (Optional):

Note: This step is necessary If the installation of Kubernetes, Helm, or Docker fails during the default installation procedure. Can be skipped once during first install.

* + Change directory ../etc
  + Edit the infra.rc file with “tested” version details
    - Run vi infra.rc
    - Edit the version numbers of Kubernetes, Docker and Helm as defined below:
      * Kubernetes v1.16
      * Helm v2.12
      * Docker current version (place an empty string “”)
  + Change directory to bin.
    - cd ../bin
  + Generate a new script with the version details set in the infra.rc file.
    - Run ./gen-cloud-init.sh

This operation should generate the stack install script with working versions of Kubernetes, Helm, and Docker.

Installation of Kubernetes, Helm, and Docker:

* + Execute the script that we generated in the previous steps.
    - ./k8s-1node-cloud-init-k\_1\_16-h\_2\_12-d\_cur.sh
    - Note: You may find a different .sh file name depending on the versions that were selected in the previous step.
  + The script installs all microservice components necessary to deploy the near-real-time RIC. Once the script completes successfully, the machine will reboot automatically.
  + Check that the Kubernetes pods are created after the reboot. A sudo -i operation may be necessary.
  + Initialize Helm with a stable repo URL (this is necessary to ensure the deployment uses the correct Helm charts)
    - Run helm init --stable-repo-url=https://charts.helm.sh/stable

Deploying the Near-Real-Time RIC using a Recipe:

* + Change directory to the bin directory.
    - Run cd dep/bin
  + Execute the deploy-ric-platform script with a recipe file.
    - Run ./deploy-ric-platform -f ../RECIPE\_EXAMPLE/PLATFORM/example\_recipe.yaml
  + Wait for the deployment script to complete. All platform components of the RIC will be deployed into separate Kubernetes pods. Once complete, check that the pods are up and running.
    - Run kubectl get pods -n ricplt
    - Around 16 pods such be running. Note the ricplt namespace created for deploying the near-real-time RIC components.

This marks the installation and deployment of the Near-Real-Time RIC components on a bare-metal system or a virtual machine. Once the RIC is deployed, it can help deploy various xApps in the *ricxapp* namespace.

**srsLTE Installation & Running**

ssh -XC [pratheek@128.173.223.202](mailto:pratheek@128.173.223.202) -p7005

sudo apt-get install build-essential cmake libfftw3-dev libmbedtls-dev libboost-program-options-dev libconfig++-dev libsctp-dev

Link to srsLTE xApp Installation: <https://gitlab.flux.utah.edu/powder-profiles/oran>

E2 Bindings:

<https://gitlab.flux.utah.edu/powder-profiles/oran/-/blob/master/setup-e2-bindings.sh>

export OURDIR="/home/pratheek/kumar/" depends on your installation directory

cd $OURDIR

if [ ! -e $OURDIR/E2AP-v01.00-generated-bindings.tar.gz ]; then

wget -O $OURDIR/E2AP-v01.00-generated-bindings.tar.gz \

<https://www.emulab.net/downloads/johnsond/profile-oai-oran/E2AP-v01.00-generated-bindings.tar.gz>

tar -xzvf $OURDIR/E2AP-v01.00-generated-bindings.tar.gz -C $OURDIR

fi

if [ ! -e $OURDIR/E2AP-v01.01-generated-bindings.tar.gz ]; then

wget -O $OURDIR/E2AP-v01.01-generated-bindings.tar.gz \

<https://www.emulab.net/downloads/johnsond/profile-oai-oran/E2AP-v01.01-generated-bindings.tar.gz>

tar -xzvf $OURDIR/E2AP-v01.01-generated-bindings.tar.gz -C $OURDIR

fi

if [ ! -e $OURDIR/E2SM-KPM-generated-bindings.tar.gz ]; then

wget -O $OURDIR/E2SM-KPM-generated-bindings.tar.gz \

<https://www.emulab.net/downloads/johnsond/profile-oai-oran/E2SM-KPM-generated-bindings.tar.gz>

tar -xzvf $OURDIR/E2SM-KPM-generated-bindings.tar.gz -C $OURDIR

fi

if [ ! -e $OURDIR/E2SM-NI-generated-bindings.tar.gz ]; then

wget -O $OURDIR/E2SM-NI-generated-bindings.tar.gz \

<https://www.emulab.net/downloads/johnsond/profile-oai-oran/E2SM-NI-generated-bindings.tar.gz>

tar -xzvf $OURDIR/E2SM-NI-generated-bindings.tar.gz -C $OURDIR

fi

if [ ! -e $OURDIR/E2SM-GNB-NRT-generated-bindings.tar.gz ]; then

wget -O $OURDIR/E2SM-GNB-NRT-generated-bindings.tar.gz \

<https://www.emulab.net/downloads/johnsond/profile-oai-oran/E2SM-GNB-NRT-generated-bindings.tar.gz>

tar -xzvf $OURDIR/E2SM-GNB-NRT-generated-bindings.tar.gz -C $OURDIR

fi

ASN1C Compiler:

<https://gitlab.flux.utah.edu/powder-profiles/oran/-/blob/master/setup-asn1c.sh>

cd $OURDIR

rm -rf $OURDIR/asn1c

# GIT\_SSL\_NO\_VERIFY=true

git clone <https://gitlab.eurecom.fr/oai/asn1c.git>

cd asn1c

# better to use a given commit than a branch in case the branch

# is updated and requires modifications in the source of OAI

#git checkout velichkov\_s1ap\_plus\_option\_group

git checkout f12568d617dbf48497588f8e227d70388fa217c9

autoreconf -iv

./configure

make -j`nproc`

sudo make install

cd ..

sudo ldconfig

ZERO\_MQ

<https://docs.srslte.com/en/latest/app_notes/source/zeromq/source/index.html>

sudo apt-get install libzmq3-dev

git clone <https://github.com/zeromq/libzmq.git>

cd libzmq

./autogen.sh

./configure

make

sudo make install

sudo ldconfig

cd ..

git clone <https://github.com/zeromq/czmq.git>

cd czmq

./autogen.sh

./configure

make

sudo make install

sudo ldconfig

cd ..

srsLTE\_ric Installation:

git clone <https://gitlab.flux.utah.edu/powderrenewpublic/srslte-ric>

cd srslte-ric

mkdir -p build

cd build

cmake ../ -DCMAKE\_BUILD\_TYPE=RelWithDebInfo -DRIC\_GENERATED\_E2AP\_BINDING\_DIR=$OURDIR/E2AP-v01.01 -DRIC\_GENERATED\_E2SM\_KPM\_BINDING\_DIR=$OURDIR/E2SM-KPM -DRIC\_GENERATED\_E2SM\_NI\_BINDING\_DIR=$OURDIR/E2SM-NI -DRIC\_GENERATED\_E2SM\_GNB\_NRT\_BINDING\_DIR=$OURDIR/E2SM-GNB-NRT

NCPUS=`grep proc /proc/cpuinfo | wc -l`

if [ -n "$NCPUS" ]; then

make -j$NCPUS

else

make

fi

sudo make install

sudo ./srslte\_install\_configs.sh service

ORAN Installation for srsLTE:

mkdir -p $OURDIR/oran

cd $OURDIR/oran

sudo docker login -u docker -p docker <https://nexus3.o-ran-sc.org:10004>

sudo docker login -u docker -p docker <https://nexus3.o-ran-sc.org:10002>

export CURRENTIMAGE="nexus3.o-ran-sc.org:10002/o-ran-sc/bldr-ubuntu18-c-go:1.9.0"

export OLDIMAGES="nexus3.o-ran-sc.org:10004/o-ran-sc/bldr-ubuntu18-c-go:1.9.0 nexus3.o-ran-sc.org:10004/o-ran-sc/bldr-ubuntu18-c-go:9-u18.04 nexus3.o-ran-sc.org:10004/o-ran-sc/bldr-ubuntu18-c-go:8-u18.04"

sudo docker pull $CURRENTIMAGE

for oi in $OLDIMAGES ; do

sudo docker tag $CURRENTIMAGE $oi

done

<https://docs.docker.com/registry/deploying/>

git clone <https://gitlab.flux.utah.edu/powderrenewpublic/e2>

cd e2

cd RIC-E2-TERMINATION

sudo docker run -d -p 5000:5000 --restart=always --name registry registry:2

sudo docker build -f Dockerfile -t localhost:5000/e2term:5.4.8 .

sudo docker push localhost:5000/e2term:5.4.8

cd ../../

git clone <http://gerrit.o-ran-sc.org/r/it/dep>

cd dep

sudo git submodule update --init --recursive --remote

sudo helm init --client-only --stable-repo-url "<https://charts.helm.sh/stable>"

cd dep/bin

Four changes in example\_recipe.yaml file. (3 in e2term & 1 in submgr)

e2term:

  alpha:

    image:

      registry: "localhost:5000"

      name: e2term

      tag: 5.4.8

    privilegedmode: false

    hostnetworkmode: false

    env:

      print: "1"

      messagecollectorfile: "/data/outgoing/"

    dataVolSize: 100Mi

    storageClassName: local-storage

    pizpub:

      enabled: false

submgr:

  image:

    registry: "nexus3.o-ran-sc.org:10002/o-ran-sc"

    name: ric-plt-submgr

    tag: 0.5.8

sudo ./deploy-ric-platform -f /home/pratheek/kumar/oran/dep/RECIPE\_EXAMPLE/PLATFORM/example\_recipe.yaml

Line number profile.py (312) follow

Execution to run srsLTE:

sudo /home/pratheek/kumar/srslte-ric/build/srsepc/src/srsepc --spgw.sgi\_if\_addr=192.168.0.1

export E2TERM\_SCTP=`sudo kubectl get svc -n ricplt --field-selector metadata.name=service-ricplt-e2term-sctp-alpha -o jsonpath='{.items[0].spec.clusterIP}'`

sudo /home/pratheek/kumar/srslte-ric/build/srsenb/src/srsenb --enb.n\_prb=15 --enb.name=enb1 --enb.enb\_id=0x19B --rf.device\_name=zmq --rf.device\_args="fail\_on\_disconnect=true,id=enb,base\_srate=23.04e6,tx\_port=tcp://\*:2000,rx\_port=tcp://localhost:2001" --ric.agent.remote\_ipv4\_addr=${E2TERM\_SCTP} --log.all\_level=warn --ric.agent.log\_level=debug --log.filename=stdout --slicer.enable=1 --slicer.workshare=0

**NexRAN Slicing xApp Installation & running in Kubernetes**

export KONG\_PROXY=`sudo kubectl get svc -n ricplt -l app.kubernetes.io/name=kong -o jsonpath='{.items[0].spec.clusterIP}'`

export E2MGR\_HTTP=`sudo kubectl get svc -n ricplt --field-selector metadata.name=service-ricplt-e2mgr-http -o jsonpath='{.items[0].spec.clusterIP}'`

export APPMGR\_HTTP=`sudo kubectl get svc -n ricplt --field-selector metadata.name=service-ricplt-appmgr-http -o jsonpath='{.items[0].spec.clusterIP}'`

export E2TERM\_SCTP=`sudo kubectl get svc -n ricplt --field-selector metadata.name=service-ricplt-e2term-sctp-alpha -o jsonpath='{.items[0].spec.clusterIP}'`

export ONBOARDER\_HTTP=`sudo kubectl get svc -n ricplt --field-selector metadata.name=service-ricplt-xapp-onboarder-http -o jsonpath='{.items[0].spec.clusterIP}'`

export RTMGR\_HTTP=`sudo kubectl get svc -n ricplt --field-selector metadata.name=service-ricplt-rtmgr-http -o jsonpath='{.items[0].spec.clusterIP}'`

sudo docker build -f Dockerfile -t docker-repository.local:8080/nexran .

sudo docker push docker-repository.local:8080/nexran

sudo docker tag localhost:8080/nexran docker-repository.local:8080/nexran

Create nexran-config file (copy from nexran.sh in powder code)

{"config-file.json\_url":"https://cdn-123.anonfiles.com/Rfn702Bdu5/6ebc2dd4-1628643253/nexran-config-file.json"}

Create nexran-onboard.url file & copy ""

<https://cdn-123.anonfiles.com/Rfn702Bdu5/6ebc2dd4-1628643253/nexran-config-file.json> (copy link nexran-onboard.url as per nexran.sh file (as per line 91))

curl -L -X POST \

"<http://$KONG_PROXY:32080/onboard/api/v1/onboard/download>" \

--header 'Content-Type: application/json' \

--data-binary "@nexran-onboard.url"

curl -L-XGET \ "<http://$KONG_PROXY:32080/onboard/api/v1/charts>"

curl -L-XPOST \        "<http://$KONG_PROXY:32080/appmgr/ric/v1/xapps>"\

--header'Content-Type: application/json'\

--data-raw'{"xappName": "nexran"}'

**srsLTE nexRAN Running Procedure**

sudo /home/pratheek/kumar/srslte-ric/build/srsepc/src/srsepc --spgw.sgi\_if\_addr=192.168.0.1

export E2TERM\_SCTP=`sudo kubectl get svc -n ricplt --field-selector metadata.name=service-ricplt-e2term-sctp-alpha -o jsonpath='{.items[0].spec.clusterIP}'`

sudo /home/pratheek/kumar/srslte-ric/build/srsenb/src/srsenb \

--enb.n\_prb=15 --enb.name=enb1 --enb.enb\_id=0x19B --rf.device\_name=zmq \

--rf.device\_args="fail\_on\_disconnect=true,id=enb,base\_srate=23.04e6,tx\_port=tcp://\*:2000,rx\_port=tcp://localhost:2001" \

--ric.agent.remote\_ipv4\_addr=${E2TERM\_SCTP} --log.all\_level=warn --ric.agent.log\_level=debug --log.filename=stdout \

--slicer.enable=1 --slicer.workshare=0

sudo ip netns add ue1

sudo /home/pratheek/kumar/srslte-ric/build/srsue/src/srsue \

--rf.device\_name=zmq --rf.device\_args="tx\_port=tcp://\*:2001,rx\_port=tcp://localhost:2000,id=ue,base\_srate=23.04e6" \

--usim.algo=xor --usim.imsi=001010123456789 --usim.k=00112233445566778899aabbccddeeff --usim.imei=353490069873310 \

--log.all\_level=warn --log.filename=stdout --gw.netns=ue1

kubectl logs -f -n ricxapp -l app=ricxapp-nexran

export NEXRAN\_XAPP=`kubectl get svc -n ricxapp --field-selector metadata.name=service-ricxapp-nexran-rmr -o jsonpath='{.items[0].spec.clusterIP}'` ; echo $NEXRAN\_XAPP

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/version ; echo ; echo

sudo ip netns exec ue1 iperf -s -p 5010 -i 4 -t 36000

iperf -c 192.168.0.2 -p 5010 -i 4 -t 36000

#!/bin/sh

SLEEPINT=4

export NEXRAN\_XAPP=`kubectl get svc -n ricxapp --field-selector metadata.name=service-ricxapp-nexran-rmr -o jsonpath='{.items[0].spec.clusterIP}'`

echo NEXRAN\_XAPP=$NEXRAN\_XAPP ; echo

echo Listing NodeBs: ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/nodebs ; echo ; echo

echo Listing Slices: ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/slices ; echo ; echo

echo Listing Ues: ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/ues ; echo ; echo

sleep $SLEEPINT

echo "Creating NodeB (id=1):" ; echo

curl -i -X POST -H "Content-type: application/json" -d '{"type":"eNB","id":411,"mcc":"001","mnc":"01"}' http://${NEXRAN\_XAPP}:8000/v1/nodebs ; echo ; echo

echo Listing NodeBs: ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/nodebs ; echo ; echo

sleep $SLEEPINT

echo "Creating Slice (name=fast)": ; echo

curl -i -X POST -H "Content-type: application/json" -d '{"name":"fast","allocation\_policy":{"type":"proportional","share":1024}}' http://${NEXRAN\_XAPP}:8000/v1/slices ; echo ; echo

echo Listing Slices: ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/slices ; echo ; echo

sleep $SLEEPINT

echo "Creating Slice (name=slow)": ; echo

curl -i -X POST -H "Content-type: application/json" -d '{"name":"slow","allocation\_policy":{"type":"proportional","share":256}}' http://${NEXRAN\_XAPP}:8000/v1/slices ; echo ; echo

echo Listing Slices: ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/slices ; echo ; echo

sleep $SLEEPINT

echo "Binding Slice to NodeB (name=fast):" ; echo

curl -i -X POST http://${NEXRAN\_XAPP}:8000/v1/nodebs/enB\_macro\_001\_001\_0019b0/slices/fast ; echo ; echo

echo "Getting NodeB (name=enB\_macro\_001\_001\_0019b0):" ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/nodebs/enB\_macro\_001\_001\_0019b0 ; echo ; echo

sleep $SLEEPINT

echo "Binding Slice to NodeB (name=slow):" ; echo

curl -i -X POST http://${NEXRAN\_XAPP}:8000/v1/nodebs/enB\_macro\_001\_001\_0019b0/slices/slow ; echo ; echo

echo "Getting NodeB (name=enB\_macro\_001\_001\_0019b0):" ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/nodebs/enB\_macro\_001\_001\_0019b0 ; echo ; echo

sleep $SLEEPINT

echo "Creating Ue (ue=001010123456789)" ; echo

curl -i -X POST -H "Content-type: application/json" -d '{"imsi":"001010123456789"}' http://${NEXRAN\_XAPP}:8000/v1/ues ; echo ; echo

echo Listing Ues: ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/ues ; echo ; echo

sleep $SLEEPINT

echo "Binding Ue to Slice fast (imsi=001010123456789):" ; echo

curl -i -X POST http://${NEXRAN\_XAPP}:8000/v1/slices/fast/ues/001010123456789 ; echo ; echo

echo "Getting Slice (name=fast):" ; echo

curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/slices/fast ; echo ; echo

#echo "Creating Ue (ue=001010123456788)" ; echo

#curl -i -X POST -H "Content-type: application/json" -d '{"imsi":"001010123456788"}' http://${NEXRAN\_XAPP}:8000/v1/ues ; echo ; echo

#echo Listing Ues: ; echo

#curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/ues ; echo ; echo

#echo "Binding Ue (imsi=001010123456788):" ; echo

#curl -i -X POST http://${NEXRAN\_XAPP}:8000/v1/slices/slow/ues/001010123456788 ; echo ; echo

#echo "Getting Slice (name=slow):" ; echo

#curl -i -X GET http://${NEXRAN\_XAPP}:8000/v1/slices/slow ; echo ; echo

#sleep $SLEEPINT

#echo "Inverting priority of fast and slow slices:" ; echo

#curl -i -X PUT -H "Content-type: application/json" -d '{"allocation\_policy":{"type":"proportional","share":1024}}' http://${NEXRAN\_XAPP}:8000/v1/slices/slow ; echo ; echo ;

#sleep $SLEEPINT

#curl -i -X PUT -H "Content-type: application/json" -d '{"allocation\_policy":{"type":"proportional","share":256}}' http://${NEXRAN\_XAPP}:8000/v1/slices/fast ; echo ; echo

curl -i -X PUT -H "Content-type: application/json" -d '{"allocation\_policy":{"type":"proportional","share":1024}}' http://${NEXRAN\_XAPP}:8000/v1/slices/slow ; echo ; echo ;

curl -i -X PUT -H "Content-type: application/json" -d '{"allocation\_policy":{"type":"proportional","share":256}}' http://${NEXRAN\_XAPP}:8000/v1/slices/fast ; echo ; echo

curl -i -X PUT -H "Content-type: application/json" -d '{"allocation\_policy":{"type":"proportional","share":1024}}' http://${NEXRAN\_XAPP}:8000/v1/slices/fast ; echo ; echo