O-RAN Near-Real Time RIC Installation Guide

# System Requirements

* OS: [Ubuntu 18.04 LTS (Bionic Beaver)](https://en.wikipedia.org/wiki/Ubuntu_version_history#Ubuntu_18.04_LTS_(Bionic_Beaver))
* CPU(s): 2-4 vCPUs
* RAM: 6-16 GB
* Storage: 20-160 GB

# O-RAN Near-Real Time RIC Software Architecture

Text, whiteboard

Description automatically generated

Step 3

Step 2

Step 1

# Near-Real Time RIC Installation Procedure Step 1: Install and configure an Ubuntu Host Machine/ Virtual Machine (VM)

The near-real time RIC can be run on a host machine or a VM as per your preference (I would recommend a VM if your system is powerful enough to support multiple VMs).

In this instruction set I am assuming the VM/Linux host system is already configured with the specified system requirements. If you need help with VM installation on Windows 10, refer []. Refer [] for help with VM configuration. This completes step 1.

## Step 2: Install Kubernetes, Docker, and Helm

### Background Information [You can either go through it now, or come back as and when required]

#### Docker:

Docker is used to run and manage apps side-by-side in isolated environments called containers that contain everything needed to run the application to get better compute density. So, you do not need to rely on what is currently installed on the host. The isolation and security allow you to run many containers simultaneously on a given host. Docker is used to build agile software delivery pipelines to ship new features faster, more securely and with repeatability for both Linux and Windows Server apps.

##### Some of the core components of docker

##### Diagram Description automatically generated

Images:

Containers:

Registry:

Docker Networking:

#### Kubernetes:

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure. It is an automated platform that enables auto-placement, auto-restart, auto-replication, auto-scaling of application containers.

Kubernetes manages a cluster of Linux machines (might be cloud VM like AWS EC2 or physical servers), on each host machine, Kubernetes runs any number of Pods, in each Pod there can be any number of containers. User’s application is running in one of those containers.

#### Helm:

The installation of Near Realtime RAN Intelligent Controller is spread onto two separate Kubernetes clusters. The first cluster is used for deploying the Near Realtime RIC (platform and applications), and the other is for deploying other auxiliary functions. They are referred to as **RIC cluster** and **AUX cluster** respectively [1].

The **RIC cluster** consists of 3 major Kubernetes Systems. Each of them is separated by their specified namespaces (kube-system ns, ricinfra ns, ricplt ns):

**Kube-system ns**: The underlying Kubernetes application which provides the basic framework for deployment and maintenance of pods.

### Commands to install near-real time RIC:

*# Enter root*

sudo -i

*# Clone the repository (“dep”) containing deployment scripts, pre generated helm charts for each of the RIC components. This repository also contains some “demo” scripts which can be run after complete installation.*

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

cd dep *#Change directory to dep*

*# Check out the latest version of every dependent submodule within the “dep” repository.*

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

cd tools/k8s/bin

# *This directory contains tools for generating a simple script that can help us set up a* ***one-node Kubernetes cluster*** *(OSC also supports a 3 node Master slave Kubernetes configuration, but I do not cover that here). The scripts automatically read in parameters (version specifications, setting up private containers/registries) from the following files:*

*k8s/etc/infra.rc: specifies the docker host, Kubernetes, and Kubernetes CNI (Cluster Networking Interfaces) versions. If left unspecified, the default version is installed.*

*k8s/etc/env.rc: Normally no change needed for this file. Can specify special/custom Kubernetes Cluster components, such as running private Docker registry with self-signed certificates, or hostnames that can be only resolved via private /etc/hosts entries.*

*etc/openstack.rc: (Relevant only for Open Stack VMs) If the Kubernetes cluster is deployed on Open Stack VMs, this file specifies parameters for accessing the APIs of the Open Stack installation.*

***For a simple installation there is no need to modify any of the above files. The files give flexibility to define our own custom Kubernetes environment if we ever need to.***

# *Run the script which will generate the Kubernetes* stack install script. Executing the below command will output a shell script called k8s-1node-cloud-init-k\_1\_16-h\_2\_12-d\_cur.sh.

./gen-cloud-init.sh

# *Executing the generated script will* install Kubernetes, Docker and Helm with version specified in the k8s/etc/infra.c. This also installs some pods which help cluster creation, service creation and internetworking between services. Running this script will replace any existing installation of Docker host, Kubernetes, and Helm on the VM. The script will reboot the machine upon successful completion. This will take some time (approx. 15-20 mins).

./k8s-1node-cloud-init-k\_1\_16-h\_2\_12-d\_cur.sh

# *Login to root again*

sudo -i

# *Check if all the pods in the newly installed Kubernetes Cluster are in “Running” state using,*

kubectl get pods -A

*There should be a total of 9 pods up and running in the cluster. These pods serve as the Kubernetes Framework which will be helpful in deploying the RIC platform. Here, I list each of the pods’ functionality (Most of which help in networking between Kubernetes nodes) [].*

* *CoreDNS* : DNS server that serves as the Kubernetes cluster DNS. This is a replacement for the default kube-dns service.
* *Flannel*: Flannel is a basic **overlay network** that works by assigning a range of subnet addresses (usually IPv4). To facilitate inter-container connectivity across nodes, flannel is used. Flannel does not control how containers are networked to the host, only how the traffic is transported between hosts. Flannel uses etcd to maintain a mapping between allocated subnets and real host IP addresses. **For example, this is very useful when the RAN is trying to communicate with the RIC since they are both different/separate nodes.**
* *Etcd server*: Consistent and highly available key value store used as Kubernetes' backing store for all cluster data. Example : Used by Flannel to register its container’s IP. etcd server stores a key-value mapping of each container with its IP.
* *Kube-APIserver*: A control plane module that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane. The Kubernetes API server validates and configures data for the api objects which include pods, services, replication controllers, and others. For example, it uses etcd server as a service to get the IP mappings and assign service IPs accordingly.
* *Kube-proxy*: Creates iptables rules and allocates static endpoints and load balancing. Basically, this means, in case the node goes down or the pod restarts it will get a new local IP, but the service IP created by kubernetes will remain the same enabling kubernetes to route traffic to correct set of pods.

See [], [], [] for more details on networking in docker and Kubernetes.

* *Kube-scheduler*: Control plane component that watches for newly created Pods with no assigned node and selects a node for them to run on. The scheduler determines which Nodes are valid placements for each Pod in the scheduling queue according to constraints and available resources. Constraints include collective resource requirements, hardware/software/policy constraints, inter-workload interference, and deadlines.
* *Kube-controller-manager*: Control plane component that runs controller processes. Some examples of controller processes include node controller (Responsible for noticing and responding when nodes go down), job controller (Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion) etc.
* *Tiller-deploy*: the server portion of Helm, typically runs inside the Kubernetes cluster. Tiller is the service that communicates with the Kubernetes API to manage our RIC components’ Helm packages. Discontinued since Helm v3 since it was seen as a security risk. But in our deployments, we are still using Helm v2, so tiller is essential.

## Step 3: Deploy the near-Real Time RIC

Influx db

Edits to helm charts

Recipe is an important concept for Near Realtime RIC deployment. Each deployment group has its own recipe. Recipe provides a customized specification for the components of a deployment group for a specific deployment site. The RECIPE\_EXAMPLE directory contains the example recipes for the three deployment groups (bronze, cherry, dawn). The benefit of using “recipe files” is that changing over from one release to another is seamless requiring the just the execution of a single script without having to perform “Step 2” all over again.

**If by chance, you encounter any issues while following the instructions visit the confluence website maintained by O-RAN Software Community for possible fixes and troubleshooting advice. (https://wiki.o-ran-sc.org/display/GS/Near+Realtime+RIC+Installation)**

# References:

1. <https://www.youtube.com/watch?v=x5MhydijWmc>
2. <https://docs.o-ran-sc.org/projects/o-ran-sc-it-dep/en/latest/installation-guides.html#one-node-kubernetes-cluster>
3. <https://kubernetes.io/docs/concepts/overview/components/>
4. <https://www.velotio.com/engineering-blog/flannel-a-network-fabric-for-containers>
5. <https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/>
6. <https://kubernetes.io/docs/concepts/cluster-administration/networking/>