KubeCon | CloudNativeCon
North America 2019

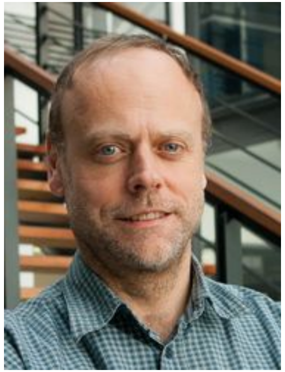
BYO Private 5G Network on Kubernetes

Frank Zdarsky, Red Hat
Raymond Knopp, Eurecom

$> whoami

Frank Zdarsky

- Sr Principal Software Engineer, Office of the CTO, Red Hat
- Responsible for Edge Computing
- Contributor to OpenShift, OpenStack, OAI, ONAP, Akraino



Raymond Knopp

- Professor in Communication Systems @ EURECOM
- President of the OpenAirInterface Software Alliance
- Expert in Radio-Access Networks

What's This Fuss about 5G?

- **Ultra flexible radio access = 5G *New Radio* (NR)**
  - Higher bandwidth and spectral efficiency (bits/s/Hz/m^2)
  - Bandwidth parts (tailor bandwidth to UE class)
  - New abstractions for service classification down to Layer 1 (slicing)
- **Radio Access Network compatibility with 4G and 5G cores**
  - 5G dual-connectivity (non-standalone operation)
  - Interconnection of evolved 4G eNodeB (ng-eNB) with 5G core
- **5G core cloud-native architecture, and evolutionary path to cloud-native radio-access, too**

New Verticals

- **5G is 3GPP's answer for enabling new use cases**
  - beyond smartphones and traditional IoT applications
  - industrial AR, remote control of drones/farming vehicles, support for new vehicular services (NR-V2X), etc.
- **Address requirements not satisfied by today's WiFi or 4G-based solutions**
  - network density/scalability, resilience for critical communications
- **Private 5G**
  - evolution of private LTE exploiting the new 5G features (i.e. low-latency and ultra-reliable transmission, service classification through slicing)
  - licensed bands for non-public applications (e.g. Industry 4.0)
  - Customized IT + Radio solution (OS+Servers+Radio+Edge Computing)

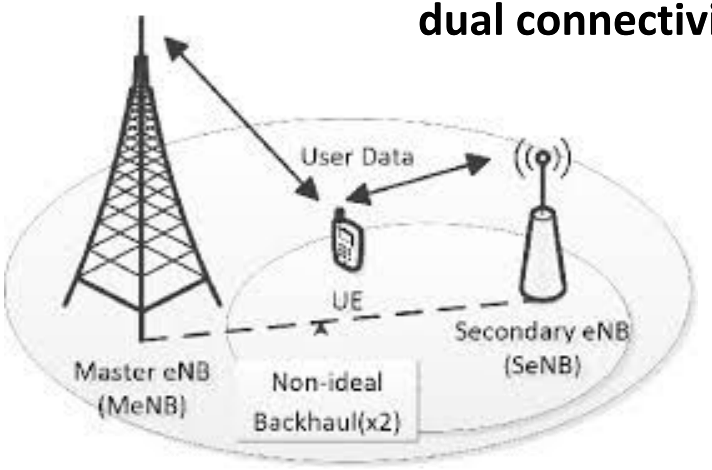
5G, small-cells and dual-connectivity

KubeCon | CloudNativeCon North America 2019

broadband low-latency → **more bandwdith** → **higher frequency** → **shorter range**

↓

dual connectivity

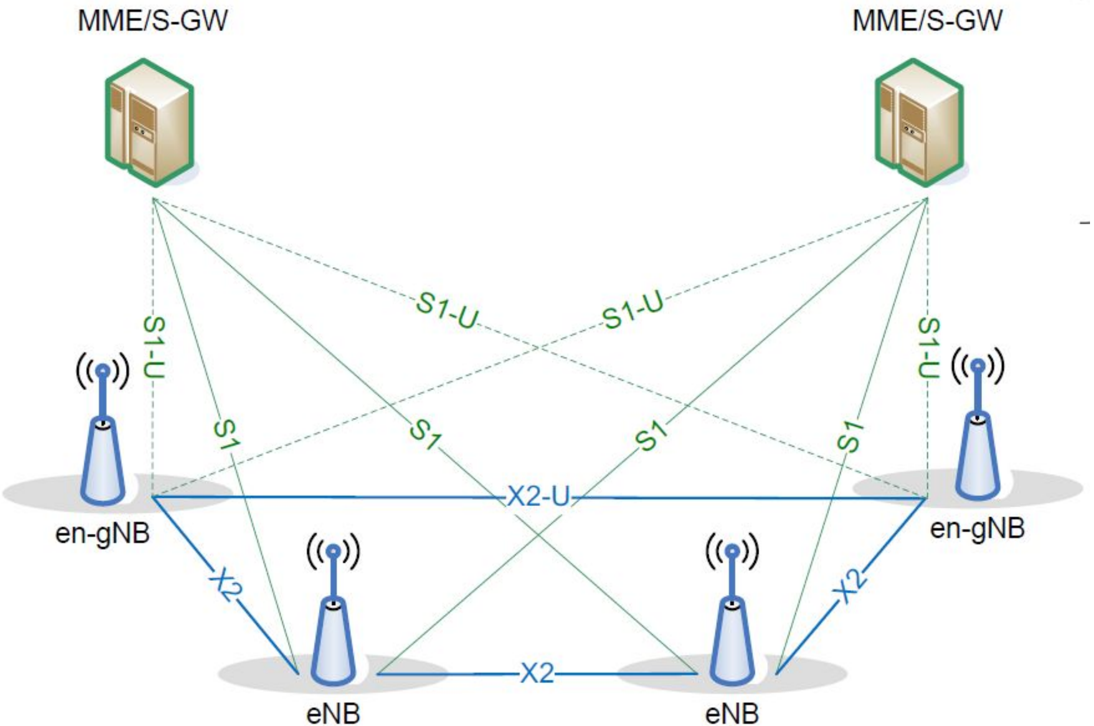


user-plane from both
control-plane only from Master

→ almost always in coverage (4G),
high-speed (5G) when near the SgNB)

What Does a 5G Network Look Like?

Elements of *today's* 5G network : implements dual connectivity with 4G core => non-standalone (NSA) operation



primary 4G CORE Network Entities

Radio-Access Network Entities

OAI and Friends

It's become feasible to put a fully compliant 4G/5G eNodeB/gNodeB and EPC/5GC in a commodity x86 box. Even major vendors adopt this approach

- Types of software for *run-of-the-mill* users
  - Amarisoft (closed, commercial)
  - OAI (open-source, 3GPP-friendly), O-RAN (partially open-source, 3GPP-friendly)
  - srsLTE/openLTE (open-source, 3GPP-unfriendly)
- Emergence of "radio-hackers" and development/user communities experimenting with 3GPP software implementations

  → Democratization of radio-access through open source SW and open HW

About OAI Alliance

- Founded in 2014 as a “Fond de Dotation” = Endowment Fund
- 3GPP strategic members (users/contributors)

orange TCL NOKIA FUJITSU INTERDIGITAL QUALCOMM PAWR Project Office

- Many associate members from industry and academia
- Donations are to maintain an engineering support team
  - CI/CD
  - Community management/building
  - Industry relations

Let’s Build This!

support from

5G PPP
PUBLIC-PRIVATE PARTNERSHIP

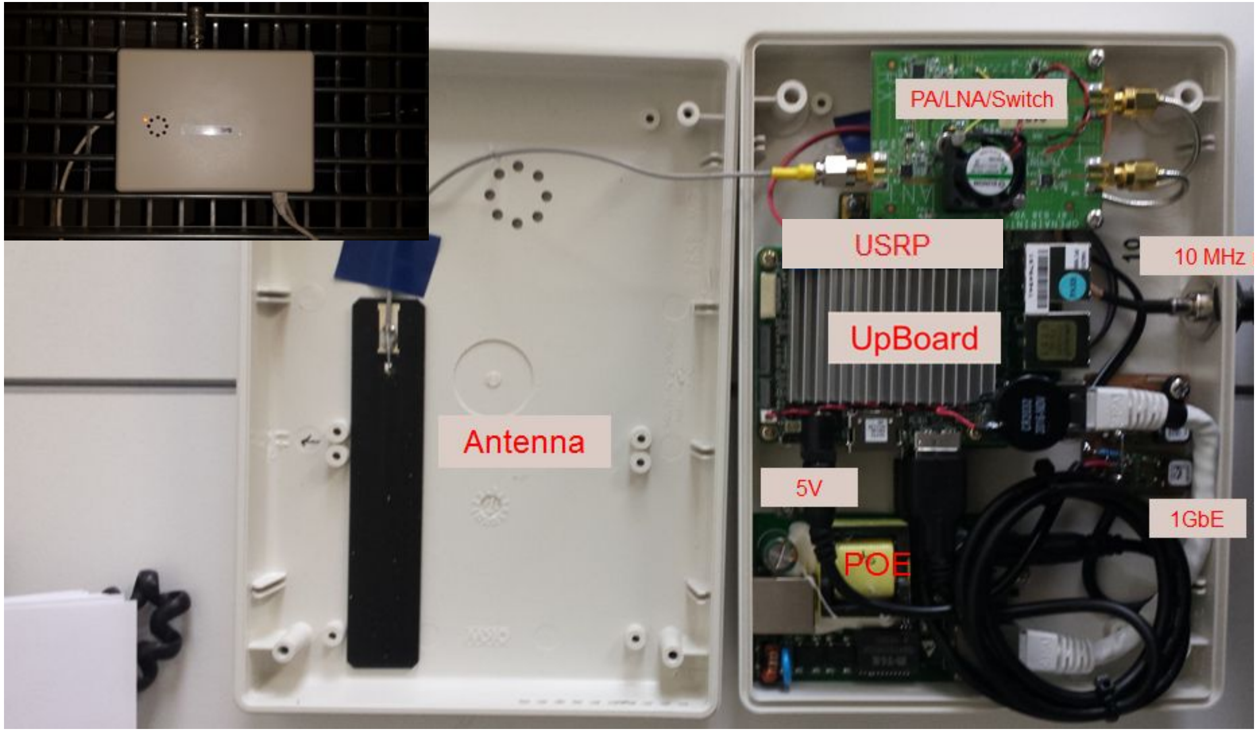
5G EVE

5G VICTORI

Red Hat | PARTNER CONNECT

Low-End Prototyping Hardware

< 50 MHz BW

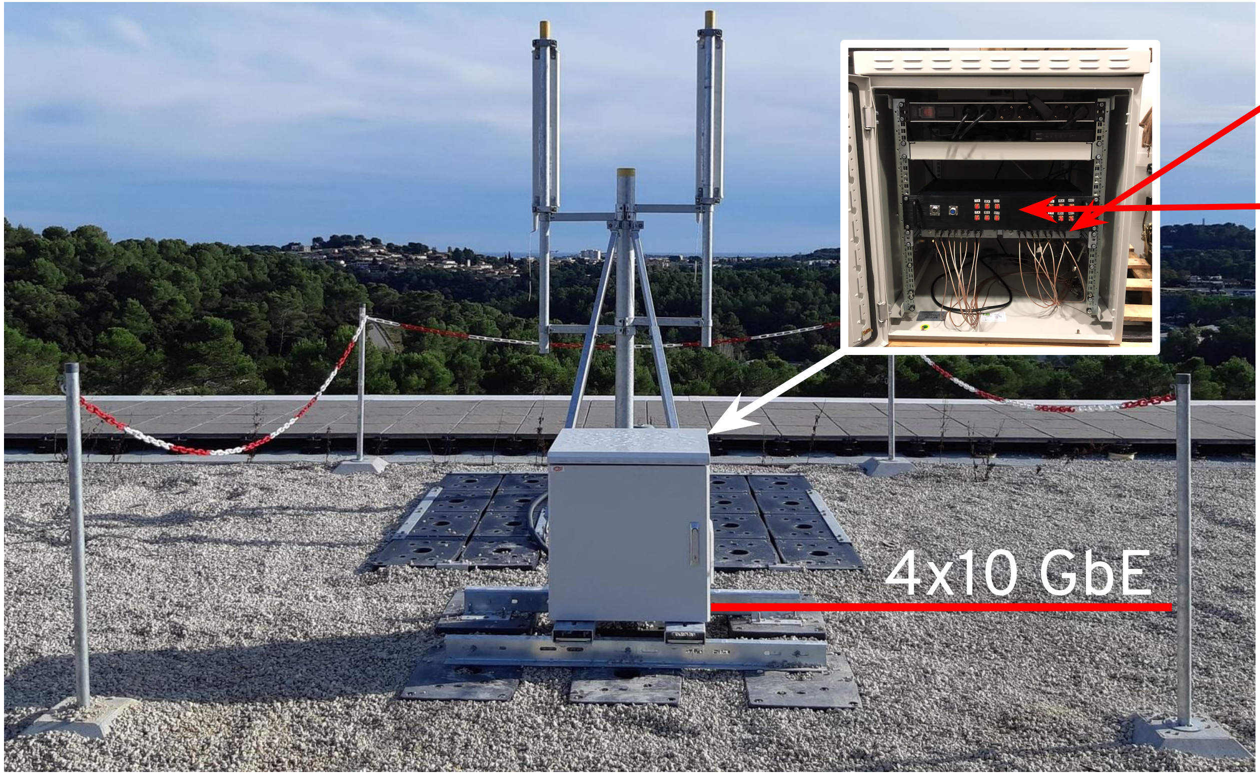


Shopping List:

- USRP B200-mini ($500)
  - up to 50 MHz BW
- custom 20 dBm PA/LNA/Switch ($300) - band 38, 42/43, n38/n77-78
- Upboard/Upboard2 (low-end $90 PC)
- GbE fronthaul POE+
- Antenna
- optional GPSDO

High-End Prototyping Hardware

8 antenna, 100 MHz (FR1)



Shopping List:

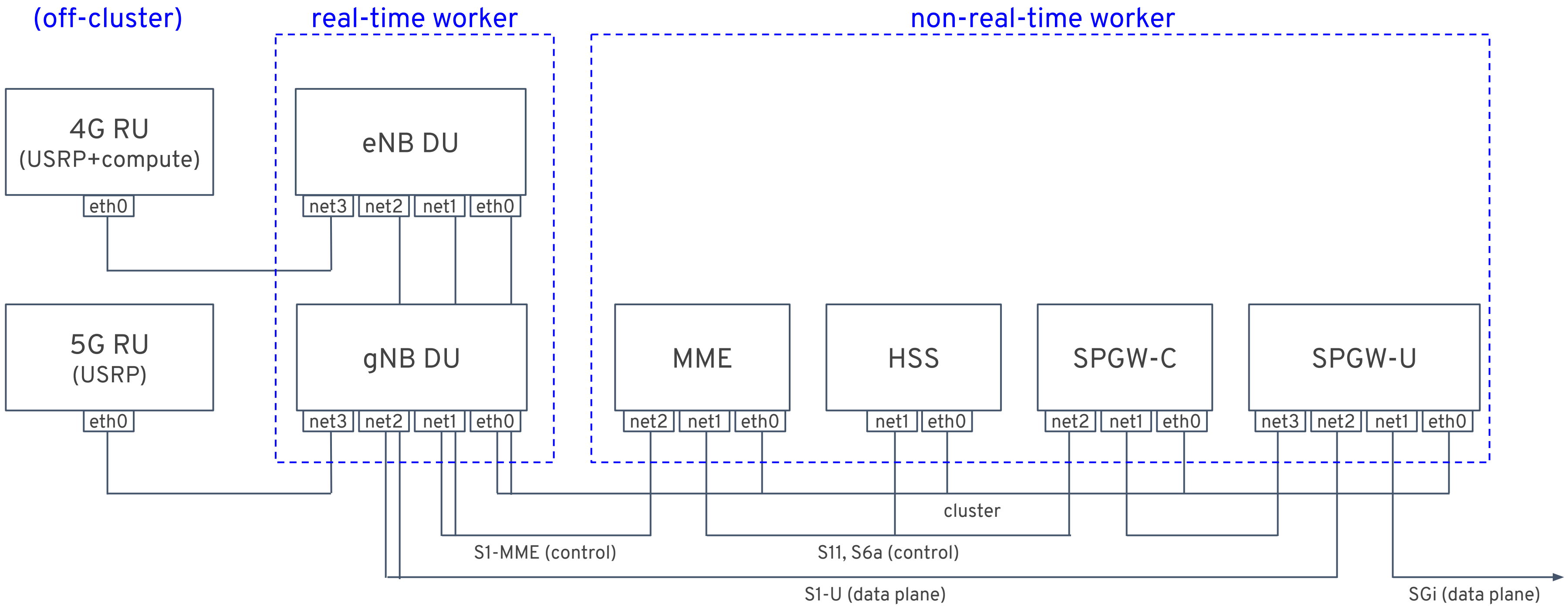
- two USRP N310 (~$20000) - up to 100 MHz BW, 8 antennas in total
- eight 2W PA/LNA/Switch (~$2500) - 2.6 or 3.5 GHz bands, e.g. **www.zhixun-wireless.top**
- 10 GbE optical fronthaul
- two 4-port Kathrein Antennas
- GPS antenna for N310s

Production-Level Hardware

Commercial radio units

- eCPRI/O-RAN Ethernet-based fronthaul solutions can be **commodity** and **whitebox** for standard interconnection with switching fabric
  http://aw2s.com/RRU.html
  https://benetel.com/product/ran-remote-radio-unit-rru/
- High-power (43 dBm); can cost less than high-end prototype described earlier.
- Less generic/flexible but complete product ready for deployment in specific bands.

Deployment Architecture

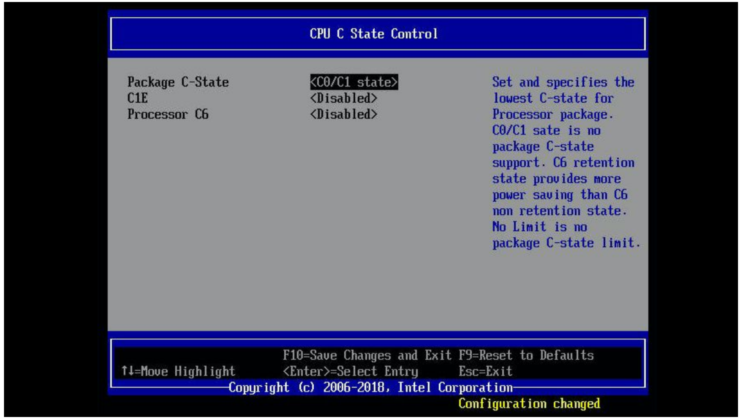


BIOS Configuration

Eliminate HW- and firmware-level sources of non-determinism

- disable C-states (CPU power save)
- disable P-states (CPU freq. scaling)
- disable EDAC (ECC memory scans)
- don't touch SMIs!

Run `hwlatdetect` for 24h to detect HW/firmware-induced latency spikes; no OS-level tuning can fix these!




```
[core@babylon ~]$ sudo hwlatdetect --duration=1d
hwlatdetect:   test duration 86400 seconds
   detector: tracer
   parameters:
        Latency threshold: 10us
        Sample window:     1000000us
        Sample width:      500000us
     Non-sampling period:  500000us
        Output File:       None

Starting test
test finished
Max Latency: Below threshold
Samples recorded: 0
Samples exceeding threshold: 0
```

Host OS Configuration

Use real-time pre-empt kernel.

Specify huge pages and isolcpus cores in /etc/tuned/realtime-variables.conf, use ‘`tuned-adm profile realtime`’ to auto-configure system parameters.

Verify OS indeed shows p- and c-states disabled.

Verify latency bounds using `cyclictest`.

```
[core@babylon ~]# sudo echo “isolated_cores=4-23” >>
/etc/tuned/realtime-variables.conf
[core@babylon ~]# sudo tuned-adm profile realtime && sudo reboot

[core@babylon ~]$ uname -a
Linux babylon 4.18.0-80.11.2.rt9.157.el8_0.x86_64 #1 SMP PREEMPT RT
Mon Sep 16 15:45:17 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux

[core@babylon ~]$ cat /proc/cmdline
BOOT_IMAGE=[...] skew_tick=1 isolcpus=4-23 intel_pstate=disable
nosoftlockup nmi_watchdog=0 audit=0 mce=off kthread_cpus=0
irqaffinity=0 skew_tick=1 processor.max_cstate=1 idle=poll
intel_idle.max_cstate=0 intel_iommu=on iommu=pt hugepagesz=1G
default_hugepagesz=1G hugepages=10 nohz=on nohz_full=4-24
rcu_nocbs=4-24

[core@babylon ~]$ sudo cpupower monitor
                  |Nehalem                   || Mperf
PKG |CORE|CPU | C3   | C6   | PC3  | PC6  || C0   | Cx   | Freq
   0|   0|   0|  0.00|  0.00|  0.00|  0.00|| 99.97|  0.03|  3690
   0|   1|   4|  0.00|  0.00|  0.00|  0.00|| 99.97|  0.03|  3690
   0|   2|   8|  0.00|  0.00|  0.00|  0.00|| 99.97|  0.03|  3690
   0|   3|   6|  0.00|  0.00|  0.00|  0.00|| 99.97|  0.03|  3690
[...]
```

K8s Config: Machines

OpenShift enables declarative management of machines and host OS via the Machine API.

To apply the RT-config automatically,

- define a MachineConfig for RT-tuning via a one-shot systemd unit

```yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker-rt
  name: machine-config-worker-rt
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;base64,[...]
          filesystem: root
          mode: 0777
          path: /opt/setup_rt.sh
    systemd:
      units:
        - contents: |
            [Unit]
            After=network-online.target
            ConditionPathExists=!/opt/rt_executed
            [Service]
            Type=oneshot
            ExecStart=/opt/setup_rt.sh
            [Install]
            WantedBy=multi-user.target
          enabled: true
          name: install_realtime.service
```

K8s Config: Machines

OpenShift enables declarative management of machines and host OS via the Machine API.

To apply the RT-config automatically,

- define a MachineConfig for RT-tuning via a one-shot systemd unit
- select it to the MachineConfigPool applied to all nodes labeled with the node-role "worker-rt"

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker-rt
  name: machine-config-worker-rt
spec:
  confi
    ig
    st
    sy
          [Service]
          Type=oneshot
          ExecStart=/opt/setup_rt.sh
          [Install]
          WantedBy=multi-user.target
        enabled: true
        name: install_realtime.service
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-rt
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role,
operator: In, values: [worker,worker-rt]}
  maxUnavailable: null
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-rt: ""
  paused: false
```

K8s Config: CPU Resource Mgmt.

To place a real-time workload (e.g. `cyclictest`) on isolated cores on the worker-rt node:

- configure the "static" cpuManagerPolicy on the Kubelet
- set resource requests and limits for *both CPU and memory* resources

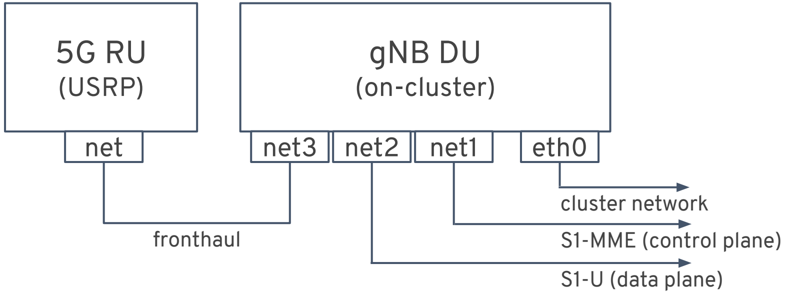
```
apiVersion: v1
kind: Pod
metadata:
  name: cyclictest
spec:
  containers:
  - name: cyclictest
    image:  docker.io/cscojianzhan/cyclictest
    resources:
      limits:
        cpu: 4
        memory: "400Mi"
      requests:
        cpu: 4
        memory: "400Mi"
    securityContext:
      capabilities:
        add:
          - SYS_NICE
          - SYS_RAWIO
          - IPC_LOCK
    volumeMounts:
    - mountPath: /dev/cpu_d
      name: cstate
  nodeSelector:
    node-role.kubernetes.io/worker-rt: ""
  volumes:
  - name: cstate
    hostPath:
      path: /dev/cpu_dma_latency
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpumanager-enabled
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: cpumanager-enabled
  kubeletConfig:
     cpuManagerPolicy: static
     cpuManagerReconcilePeriod: 5s
     kubeReserved:
       cpu: "1"
```

K8s Config: Networking

We use Multus CNI to segregate the 3GPP control and data plane networks from the cluster net-work (used for mgmt.).

eNB and gNB pods are connected to the USRP software-defined radios via dedicated interfaces.




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oai-gnb
  labels:
    app: oai-gnb
spec:
  selector:
    matchLabels:
      app: oai-enb
  template:
    metadata:
      labels:
        app: oai-gnb
      annotations:
        k8s.v1.cni.cncf.io/networks:
control, data, fronthaul
    spec:
      [...]
```

```yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: fronthaul
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "macvlan",
    "master": "bond0",
    "mode": "bridge",
    "ipam": {
      "type": "static",
      "addresses": [
        {"address": "192.168.18.206/24",
         "gateway": "192.168.18.1"}
      ]
    }
  }'
```

Deploying OAI

Deploy vRAN-ready cluster, e.g. using the Akraino KNI for vRAN blueprint.[0]

Clone openair-k8s Github repo.[1]

On a RHEL host, build OAI images and push to local cluster registry:

```
hack/build_images
hack/push_images $your_cluster_registry
```

Adapt config to your deployment.

Deploy:

```
kustomize build manifests/$component | kubectl apply -f -
```

Let’s Demo This!

USRP N310
TX
RX
ANT
ANT

KubeCon | CloudNativeCon

North America 2019

References

[0] https://wiki.akraino.org/display/AK/Provider+Access+Edge+%28PAE%29+Blueprint
[1] https://github.com/openairinterface/openair-k8s
[2] https://github.com/OPENAIRINTERFACE/openair-cn
[3] https://github.com/OPENAIRINTERFACE/openair-cn-cups
[4] https://gitlab.eurecom.fr/oai/openairinterface5g
[5] https://https://5g-ppp.eu
[6] https://5g-ppp.eu/5g-eve
[7] https://5g-ppp.eu/5g-victori