Is There a Place for Performance Sensitive Workloads in Kubernetes?

Levente Kálé, Gergely Csatári

whoarewe

Levente Kálé

Nokia
TelCo edge cloud architect

: levovar-045/

: @Levovar

Gergely Csatári

Nokia
Open Source Specialist

: gergely-csatári-76591020/

: @CsatariGergely

What do we do?

Surprise: Kubernetes-based TelCo grade clouds and containerized TelCo applications ☺

Clouds

- Private edge, central, and public clouds
- Kubernetes-only or multi-orchestrator

Apps

- *5G Radio Access Network (DU, CU, RIC)*
- Mobile core
- Fixed network
- IT/Enterprise

Open source entanglements

- CNCF TUG
- CNF Testbed
- CNTT
- SIG-Node / WG-Resource Management
- DANM
- CPU-Pooler
- Akraino

Is There a Place for Performance Sensitive Workloads in Kubernetes?

How does K8s perform when it is used to run the **worst possible** workloads?

Our worst possible is running compute heavy workloads:

- ***Using*** all the available CPU of a Node without much idling
- ***With*** high network throughput needs: 25Gbps+
- ***With*** real-time latency requirements: <1µs RTT
- ***Facing*** challenging high-availability ("five-nines") and legal (government laws) requirements

Is There a Place for Performance Sensitive Workloads in Kubernetes?

Y E S!!!

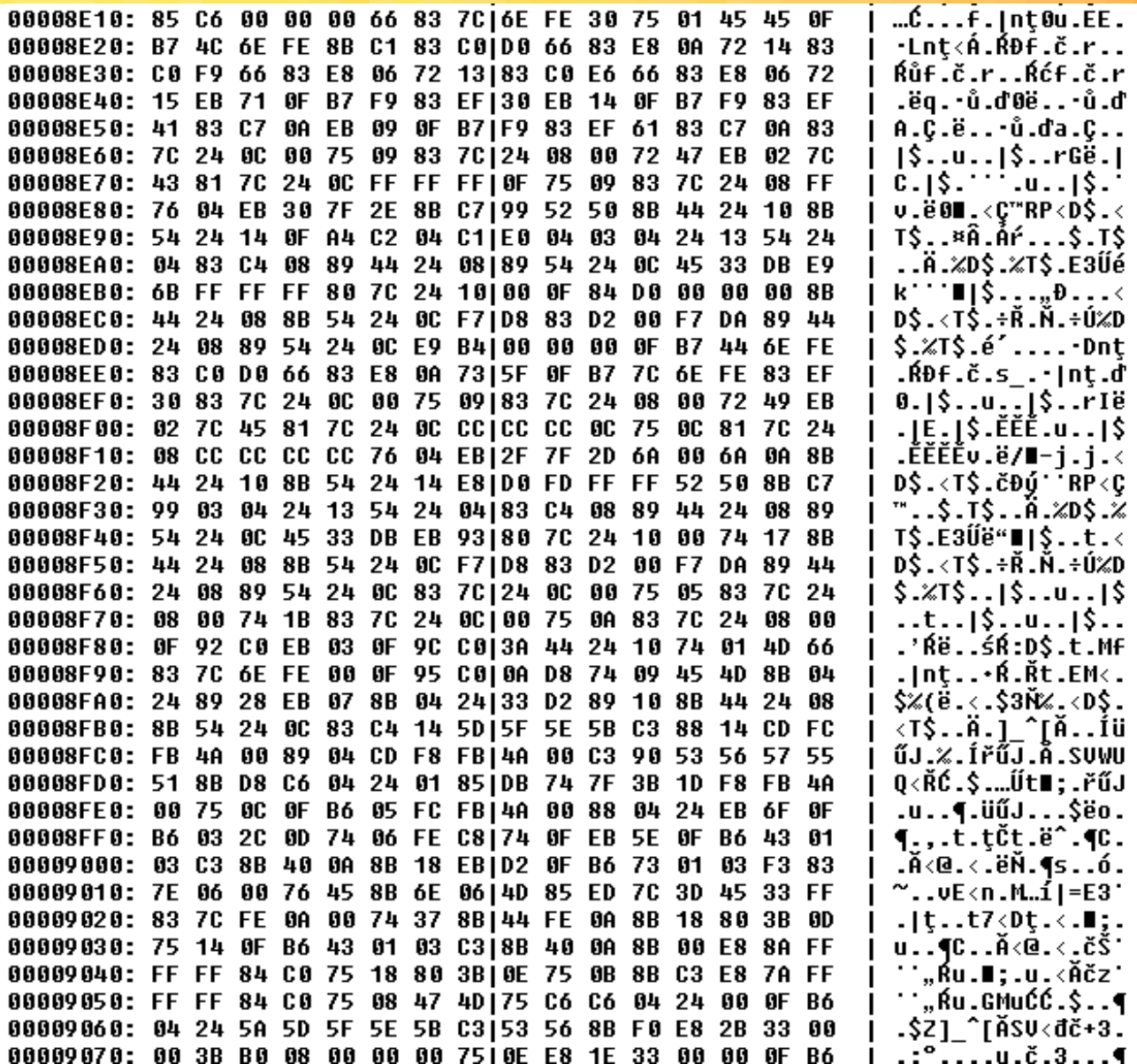
B U T...

"Solutions" out-of-scope and question

Before deep-dive, some things You should also say no to:

- Proprietary hardware
- proprietary operating systems
- special kernel patches (including preemptive RT kernel)
- proprietary kernel drivers
- privileged applications

With a few native enhancements even the worst applications can be kept cloud-native*!
(*ish) ☺



Realtime Scorecard for K8s 1.16

CPU management ○○○○○

HW acceleration ○○○○○

Memory management ○○○○○

Network management ○○○○○

Topology awareness ○○○○○

CPU management baseline

Kubernetes has a CPU manager!

The Good:

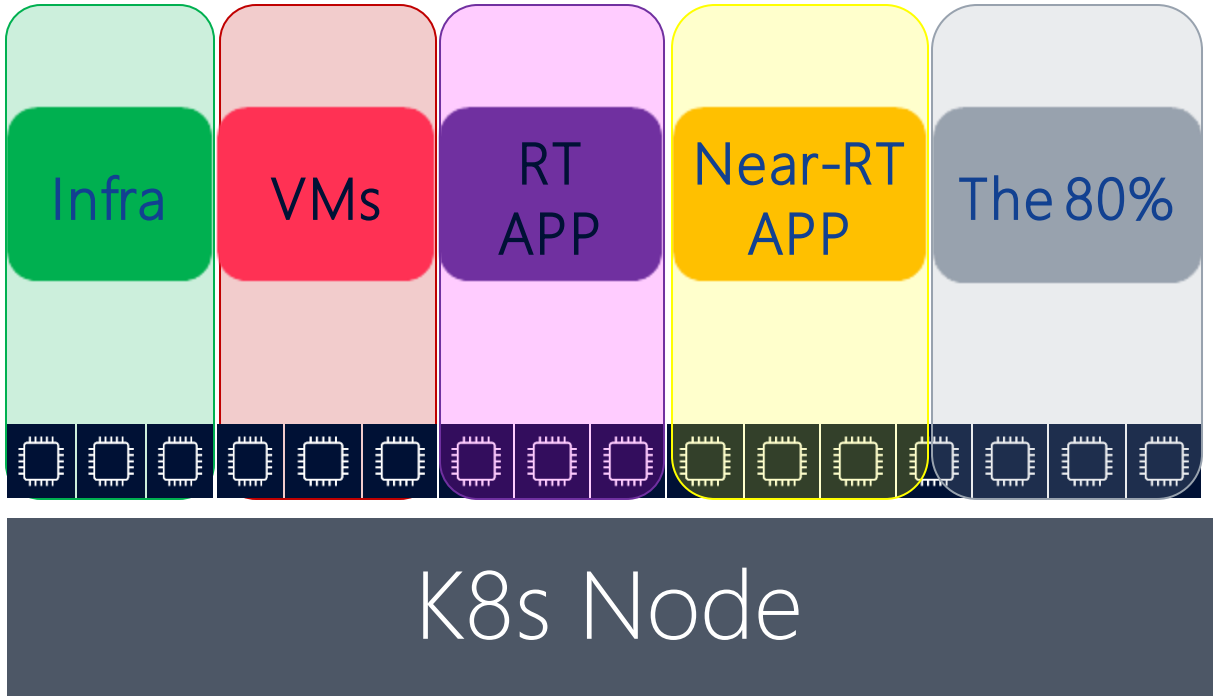
- Uses cpusets
- Supports multiple policies: none, static
- Possible to allocate exclusive CPU cores

The Bad:

- No interworking with non-K8s managed processes
- Only supports node-level separation

CPU management – Need moar pools!

Reservoir pools



Mr. Green, The Platfrom

Mr. Red, The Virtual Machine

Mr. Purple, The Exclusive

Mr. Orange, The Shared

Mr. Grey, The Default

CPU management – CPU-Pooler

https://github.com/nokia/CPU-Pooler

Native extension to CPU Manager

- Uses core resources API
- Uses same kernel features

Just supports more pools – and makes it possible to tune them differently

CPU management – Pooler in action


```
poolconfig-compute-1.yaml:
---
nodeSelector:
  nodename: caas_worker1
pools:
  default:
    cpus: 1,13-14,29,41-42
  exclusive_caas:
    cpus: 9-12,24-27,37-40,52-55
  shared_caas:
    cpus: 2-8,15-23,30-36,43-51
```

```
[cloudadmin@controller-1 ~]$ kubectl exec cpu-pooling-demo-769fb5fb44-vftqx -c default-test cat /proc/1/status | grep Cpus_allowed_list
Cpus_allowed_list:      1,13-14,29,41-42
```

```
[cloudadmin@controller-1 ~]$ kubectl exec cpu-pooling-demo-769fb5fb44-vftqx -c exclusive-test cat /proc/1/status | grep Cpus_allowed_list
Cpus_allowed_list:      10
```

```
[cloudadmin@controller-1 ~]$ kubectl exec cpu-pooling-demo-769fb5fb44-vftqx -c shared-test cat /proc/1/status | grep Cpus_allowed_list
Cpus_allowed_list:      2-8,15-23,30-36,43-51
```

CPU management enhanced

Support non-K8s managed pools by excluding cores from its own

CRI resource manager

Why not take the best of both worlds -> ***dynamic pooling!***

- Natively support sub-node pools like CPU-Pooler
- But allocate cores to a pool dynamically, on-demand like CPU Manager

A „CPU Pool" would become an abstraction, describing a **set of characteristics** or **configurations** CPU Manager would apply ***when a workload is instantiated!***

HW acceleration baseline

HW acceleration: delegate recurring and costly computing operations to specialized hardware
Examples: (SR-IOV,) FPGA, GPU

The Good:

- Flexible and extensible (gRPC)
- Device Plugin API to plugin HW device managers

The Bad:

- Absence of fine-grained control

HW acceleration devices

F(ield)P(rogrammable)G(ate)A(rray): better for single-thread/serialized, high-volume, low-complexity computations

- FEC computation in L1 of RAN (https://en.wikipedia.org/wiki/Forward_error_correction)
- Offloading network, crypto(encryption/decryption), storage management etc.

GPU: better for multi-threaded computations

- (Predictive) rendering: cloud gaming, AR/VR
- High performance general purpose computing (CUDA/GPGPU): AI/ML, image recognition, neural networks, physical simulations, cryptocurrency

HW acceleration enhanced

„Release” DPAPI call

Passing parameters to Allocate() DPAPI call

Sharing the same physical device (e.g. queue of the same FPGA, GPU cores/lanes of the same card etc.)

Memory management baseline

The Good:

- Native support for allocating normal memory
- Supports huge memory pages
- Supports different sizes (2M, 1Gi)

The Bad:

- Need better isolation
- Lack of topology awareness hurts performance

Memory management enhanced

Topology aware accounting of hugepages

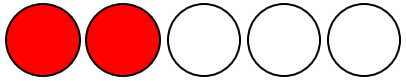
Topology aware resource management of hugepages

Manage hugepages on the container level

Topology aware allocation of RAM

Isolating memory and hugepages from non-K8s managed workloads?

Network management baseline



"Network management" in vanilla Kubernetes == CNI

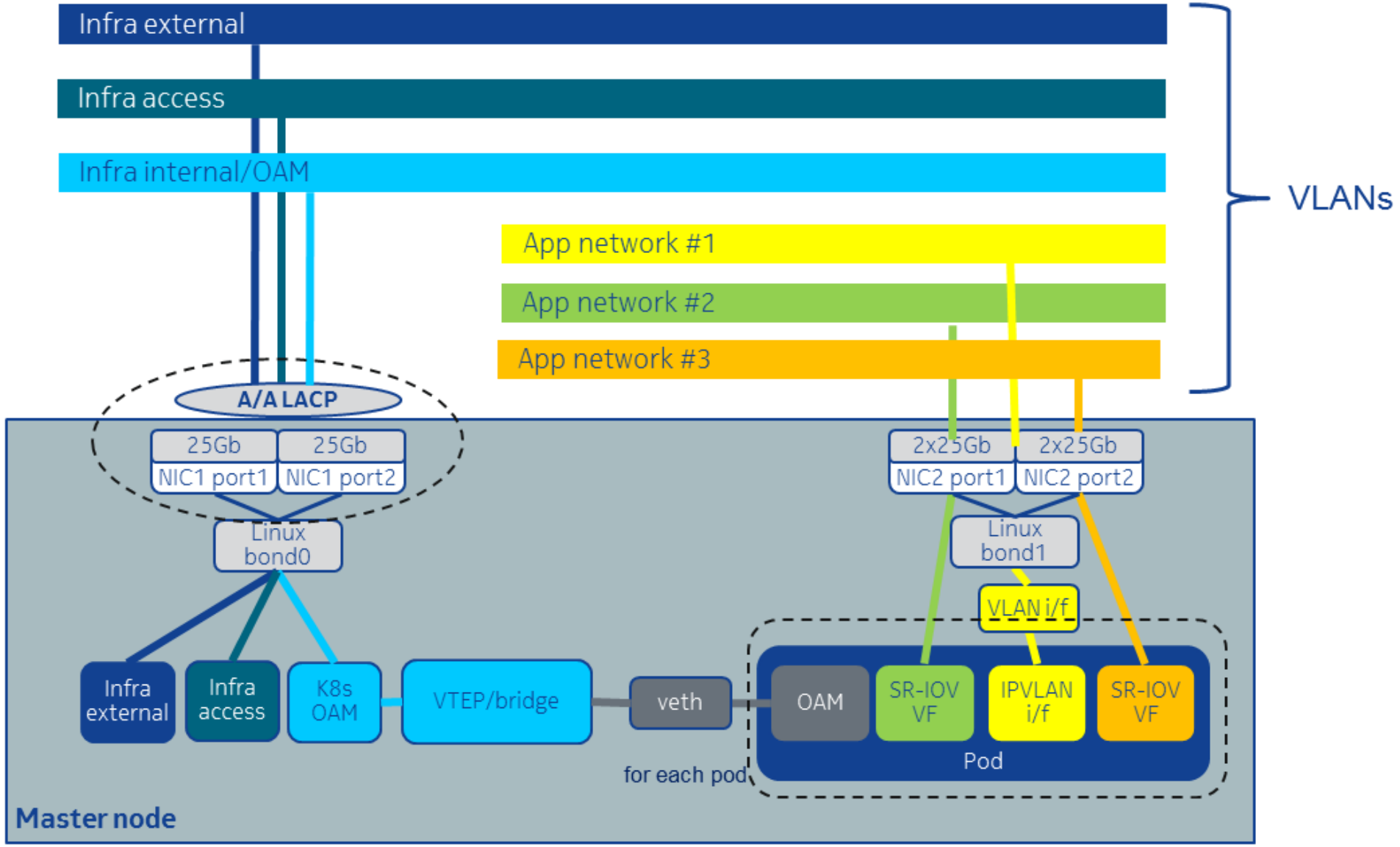
The Good:

- Perfect for small, single tenant, IT apps

The Bad:

- Perfect for *small*, *single* tenant, *IT* apps
- Cannot satisfy TelCo functional, standard, and legal requirements

Network management requirements in production



Network management eco-system

Industry consensus: K8s needs a centralized network manager in production
To quote NSM documentation: „An API should exist that allows the **VNF*** to specify its networking intent through an abstract API dynamically."

Same problem description, different solutions:

DANM (presenter's choice): hides –but implements- CNI, provides multi-tenant, ***user role specific*** network management APIs to **operators** and **applications**, implements common features in a backend-agnostic manner

Multus: directly exposes CNI to operators

Network Service Mesh: works independently from normal CNI, extra interfaces provisioned post-deployment, targeting specific use-cases

Networking enhanced

https://github.com/nokia/danm

API-driven, multi-tenant, multi-role network management integrated to bare-metal cloud underlay network fabric ("host-to-leaf")

Supports multiple, physically separated interfaces

Natively supports multiple, varying network provisioning backends with different characteristics

Centralized, API-driven features (e.g. K8s Services, IPAM, IP routes, VLAN, VxLAN etc.) extended for all interfaces, agnostic of their type

Don't believe the hype

DANM User Guide

Akraino REC

Multiple networks for Kubernetes workloads

Topology awareness baseline

There is now a Topology Manager in K8s!

The Good:

- Aligns CPUs, and Devices
- Multiple policies available from the get-go ("strict" included)

The Bad:

- Unnecessary restriction on alignment
- Does not align hugepages
- No topology aware *scheduling*

Topology awareness enhanced

Fortunately Topology Manager is a major community focus, so most of the shortcomings are already being addressed!

History and future

1.17 issue tracker

Notable and important enhancements:

Removing restriction of when alignment can happen (1.17)

Hugepage support (1.18)

Topology aware scheduling (under discussion)

The final, definitive, never-to-be-changed K8s 1.16 realtime scorecard is...

References

- DANM: https://github.com/nokia/danm/
- CPU Pooler: https://github.com/nokia/CPU-Pooler
- CNTT: https://github.com/cntt-n/CNTT
- CNCF TUG: https://github.com/cncf/telecom-user-group
- REC: https://www.lfedge.org/projects/akraino/release-1/telco-appliance-radio-edge-cloud/

Q&A