An Analysis of Lightweight Virtualization Technologies for NFV

Sriram Natarajan, Deutsche Telekom Inc.
Ramki Krishnan, Dell Inc.
Anoop Ghanwani, Dell Inc.
Dilip Krishnaswamy, IBM Research
Peter Willis, BT Plc
Ashay Chaudhary, Verizon
Felipe Huici, NEC – new co-author

Recap of first version

- VM based network functions
  - Key Challenges
- Container-based network functions
  - Key Benefits, Challenges and Potential Solutions

Latest version summary

- Beyond Container-based network functions
- Include other lightweight virtualization technologies Unikernels and OS Tinyfication
- Detailed benchmarking experiments
- Comparison discussion
- Next Steps

Benchmark Setup (1)

- All tests are run on an x86-64 server with an Intel Xeon E5-1630 v3 3.7GHz CPU (4 cores) and 32GB RAM.

- For the hypervisors we use KVM running on Linux 4.4.1 and Xen version 4.6.0. The virtual machines running on KVM and Xen are of three types:
  - Unikernels, on top of the minimalistic operating systems OSv and MiniOS for KVM and Xen, respectively. The only application built into them is iperf. To denote them we use the shorthand unikernel.osv.kvm or unikernels.minios.xen.
  - Tinyfied Linux (a.k.a. Tinyx), consisting of a Linux kernel version 4.4.1 with only a reduced set of drivers (ext4, and netfront/blkfront for Xen), and a distribution containing only busybox, an ssh server for convenience, and iperf. We use the shorthand tinyx.kvm and tinyx.xen.
  - Standard VM, consisting of a Debian distribution including iperf and Linux version 4.4.1. We use the shorthand standardvm.kvm and standardvm.xen for it.

- For containers, we use Docker version 1.11 running on Linux 4.4.1.

Benchmark Setup (2)

- Numbers reported here for virtual machines (whether standard, Tinyx or unikernels) include the following optimizations to the underlying virtualization technologies.
  - For Xen, we use the optimized Xenstore, toolstack and hotplug scripts reported in [SUPERFLUIDITY] as well as the accelerated packet I/O derived from persistent grants (for Tx) [PGRANTS].
  - For KVM, we remove the creation of a tap device from the VM's boot process and use a pre-created tap device instead.

- Choice of iperf for VNF workloads
  - Basic TCP-stack based throughput/delay measurements to find out what the fundamental limits of the underlying virtualization technology are *before* adding any VNFs.
  - TCP stack data plane processing - cover a broad range of data plane VNFs such as firewall, load balancer, IDS/IPS etc.

Benchmark - Instantiation Times (Service Agility/Elasticity)

Technology Type	Time (msecs)
standardvm.xen	6500
standardvm.kvm	2988
Container	1711
tinyx.kvm	1081
tinyx.xen	431
unikernel.osv.kvm	330
unikernels.minios.xen	31

- Test Methodology
  - Carry out a SYN flood from an external server and measure the time it takes for the container/VM to respond with a RST packet during instantiation
- Result
  - Unikernel the fastest to come up

Benchmark - Throughput

Technology Type	Throughput (Gb/s) Tx	Throughput (Gb/s) Rx
standardvm.xen	23.1	24.5
standardvm.kvm	20.1	38.9
Container	45.1	43.8
tinyx.kvm	21.5	37.9
tinyx.xen	28.6	24.9
unikernel.osv.kvm	47.9	47.7
unikernels.minios.xen	49.5	32.6

- Test Methodology
  - TCP traffic using guest IPPERF application between the guest and the host -- there are no NICs involved so that rates are not bound by physical medium limitations.
- Result
  - Containers and unikernels (at least for Tx and for Tx/Rx for KVM) are fairly equally matched and perform best, with unikernels having a slight edge.

Benchmark - RTT

Technology Type	Time (msecs)
standardvm.xen	34
standardvm.kvm	18
Container	4
tinyx.kvm	19
tinyx.xen	15
unikernel.osv.kvm	9
unikernels.minios.xen	5

- Test Methodology
  - To measure round-trip time (RTT) from an external server to the VM/container we carry out a ping flood and report the average RTT.
- Result
  - Containers are the best, unikernels with mini OS follow.

Benchmark - Image Size

Technology Type	Size (MBs)
standardvm.xen	913
standardvm.kvm	913
Container	61
tinyx.kvm	3.5
tinyx.xen	3.7
unikernel.osv.kvm	12
unikernels.minios.xen	2

- Test Methodology
  - Measure image size using standard “ls” tool.
- Result
  - Largest image size is standard VM
  - The smallest image is the one based on MiniOS/Xen with 2MB.

Benchmark - Memory Usage

Technology Type	Usage (MBs)
standardvm.xen	112
standardvm.kvm	82
Container	3.8
tinyx.kvm	30
tinyx.xen	31
unikernel.osv.kvm	52
unikernels.minios.xen	8

- Test Methodology
  - Measure memory usage using standard tools such as “top” and “xl” (Xen’s management tool).
- Result
  - Largest memory usage – standard VM
  - Smallest memory usage – container

Comparison Discussion

- Service Agility/Elasticity
  - Unikernels and Containers are the best
- Memory Usage
  - Containers are the best, share host resources
- Management Frameworks
  - Standard VMs and Containers have well supported open source communities
  - Unikernels might need open source glue
- Security/Isolation
  - VM based solutions are robust
  - Containers are evolving
    - Kernel security modules SELinux, Apparmor can cover some aspects
- Application Compatibility
  - Standard VMs are most flexible
  - Linux OS has the best container support – container feature set is catching up in other OS such as Windows, BSD etc.
  - Unikernels are specialized
- Overall choice
  - Not black and white – use case and vendor solution availability dependent

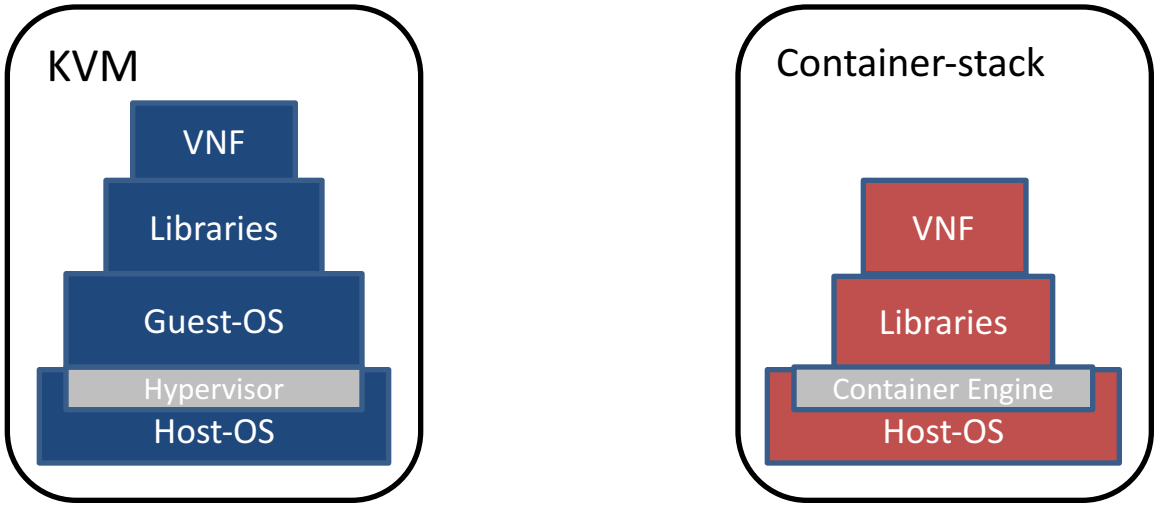
Next Steps

- Read the draft?

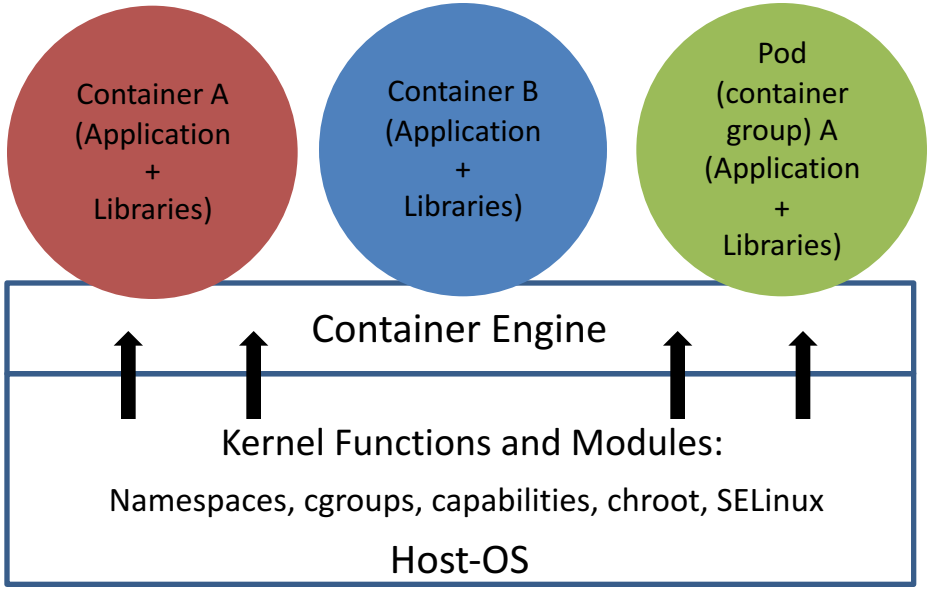
- Requesting for research group adoption over the list

Backup (material from previous version)

Virtual Machine vs. Container Stack



- **Lightweight footprint**: Very small images with API-based control to automate the management of services
- **Resource Overhead**: Lower use of system resources (CPU, memory, etc.) by eliminating hypervisor & guest OS overhead



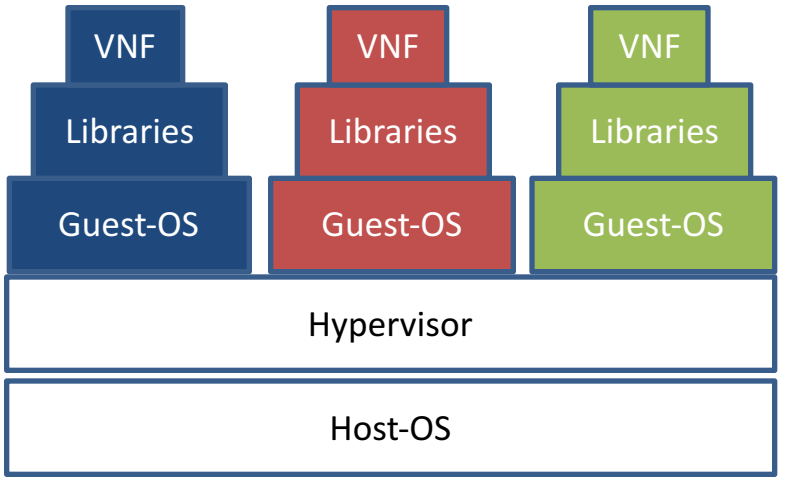
- **Deployment time**: Rapidly deploy applications with minimal run-time requirements
- **Updates**: Depending on requirements, updates, failures or scaling apps can be achieved by scaling containers up/down

VM based Network Functions

Key Challenges

Service Agility/Performance

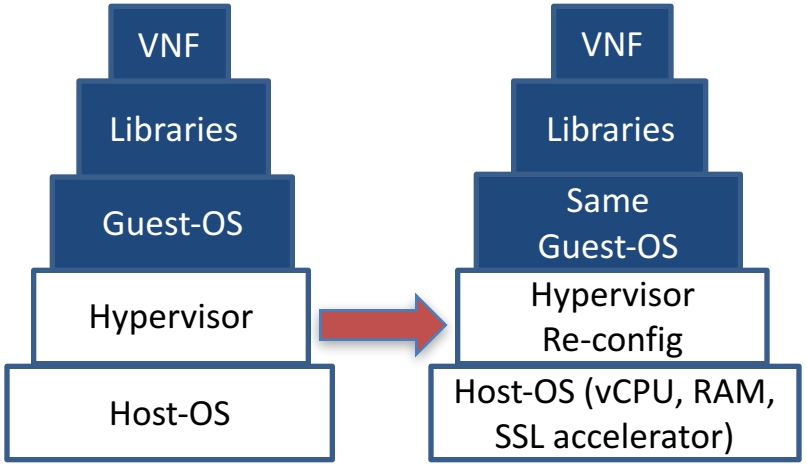
- Provisioning time:
  - Hypervisor configuration
  - Spin-up guest OS
  - Align dependencies between Guest-OS & VNFs



- Runtime performance overhead:
  - Performance proportional to resource allocated to individual VMs (throughput, line rate, concurrent sessions, etc.)
  - Overhead stems from components other than VNF process (e.g. guest OS)
  - Need for inter-VM networking solution
  - Meeting SLAs requires dynamic fine tuning or instantiation of additive features, which is complex in a VM environment

Portability/ Elasticity/Scalability

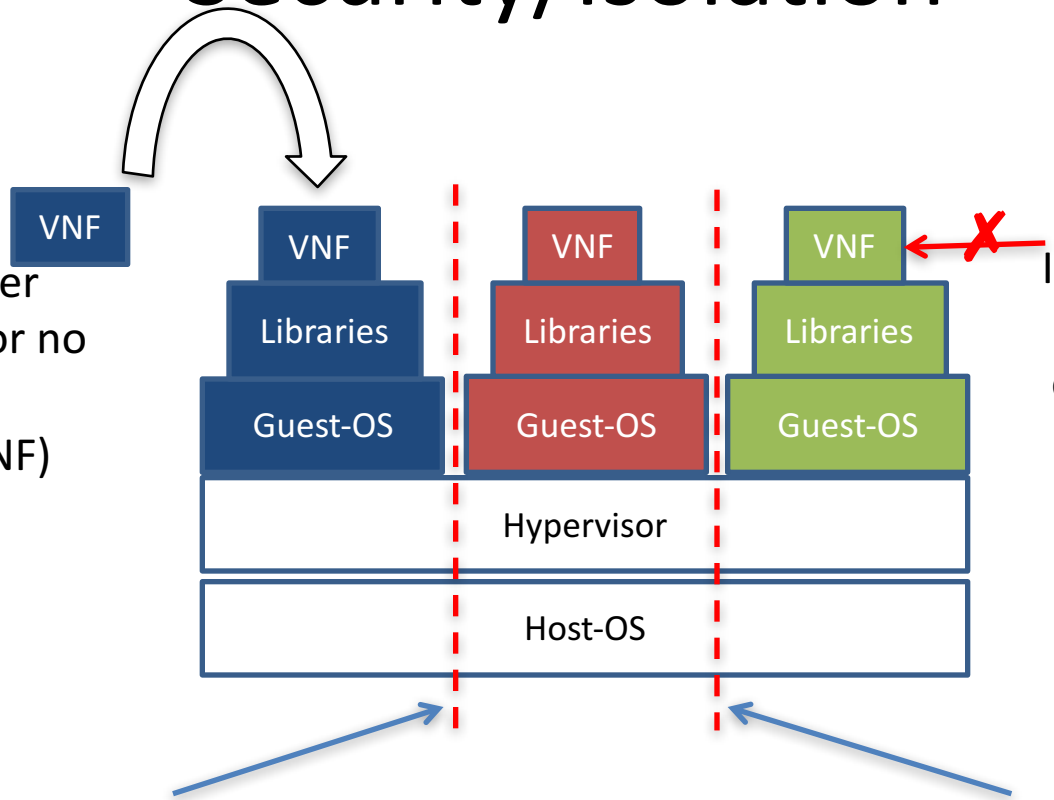
- Porting VNFs require:
  - Identifying suitable nodes for new VNF instances (or re-locating existing instances). For example, resource types, available capacity, guest OS images, hypervisor configs, HW/SW accelerators, etc.)
  - Allocating required resources for new instances
  - Provisioning configs to components in the guest OS, libraries and VNF



- Elastic scalability needs are driven by workloads on the VNF instances, and stateful VNFs increase the latency to spin up new instances to fully working state.

Security/Isolation

Securely recover with minimal or no downtime (reschedule VNF)



If VNF is compromised (misconfiguration, etc.), how to securely quarantine the VNF, but ensure continuity of other VNFs?

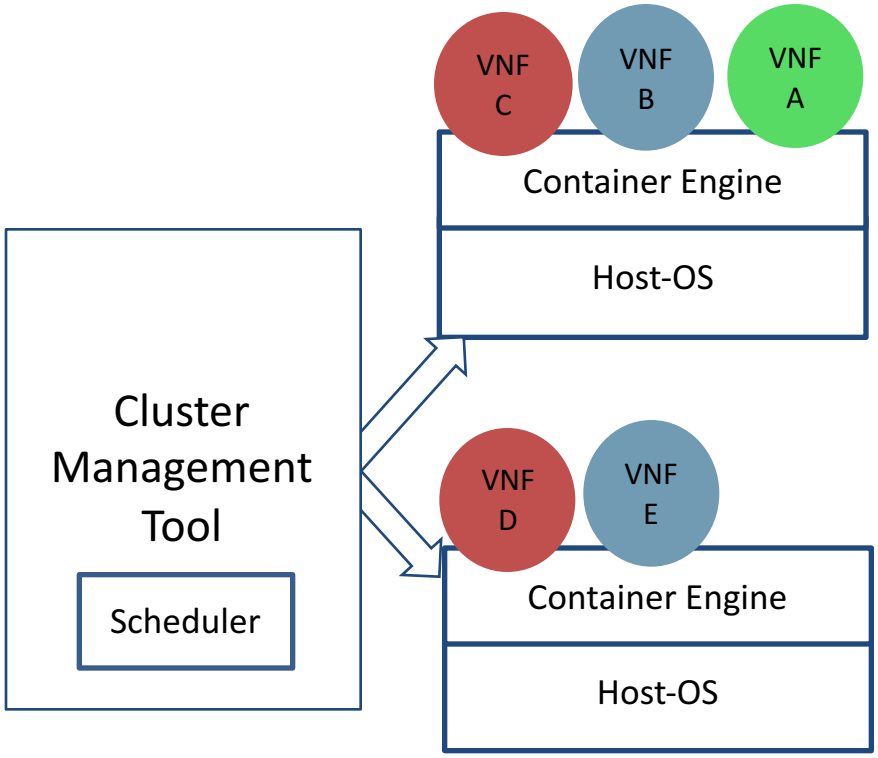
Guarantee complete isolation across resource entities (hardware units, hypervisor, protection of shared resource, isolation of virtual networks, L3 cache, QPI, etc.)

Resource hungry VNF can starve the shared resources (noisy neighbor effect) that are allocated to other VNFs; Need to monitor and cut-off hungry VNF usage

Containerized Network Functions

Key Benefits, Challenges and Potential Solutions

Service Agility/Performance/Isolation (1)

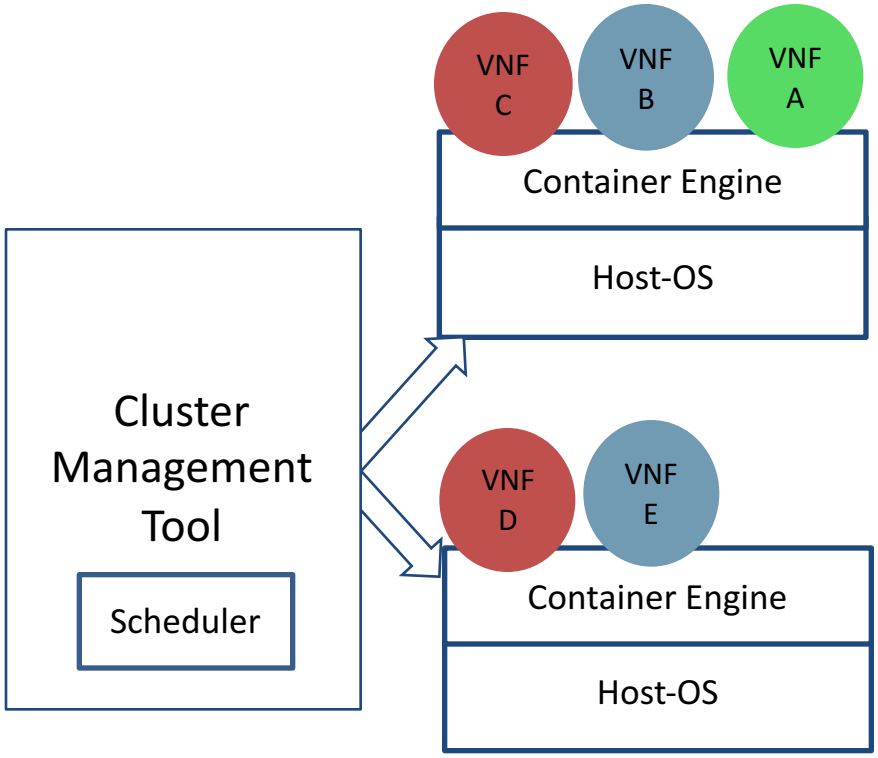


Key Benefits:

- Containers can provide better service agility (e.g. dynamically provision VNFs for offering on-demand services), and performance as it allows us to run the VNF process directly in the host environment

- Inter-VNF communication latency depends on inter-process communication option (when hosted in the same host)

Service Agility/Performance/Isolation (2)



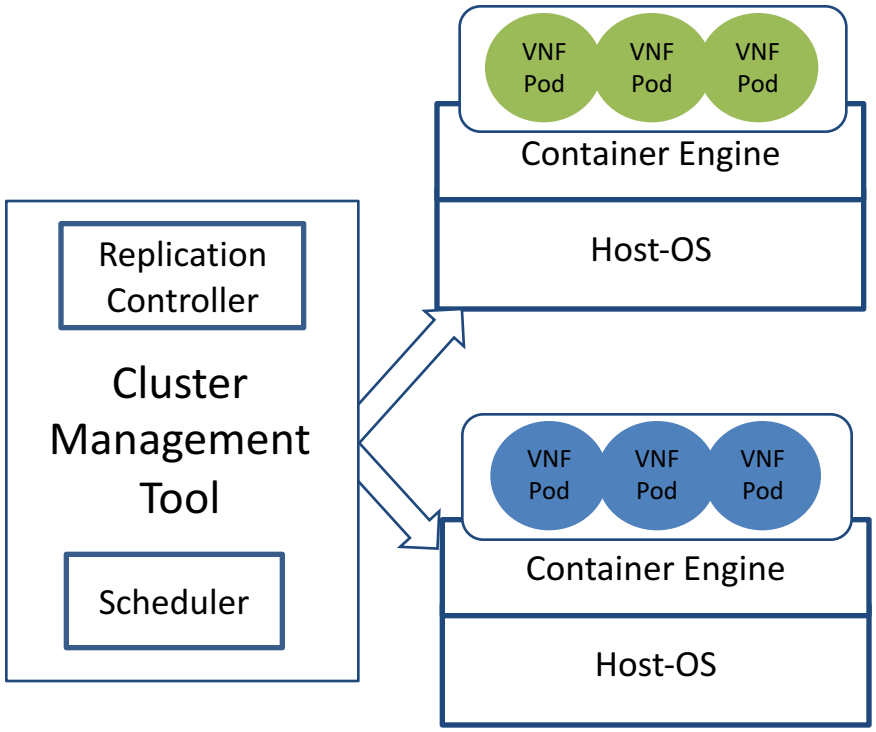
Key Challenges:

- Isolation: Containers create a slice of the underlying host using techniques like namespaces, cgroups, chroot etc.; several other kernel features that are not completely isolated.
- Resource Mgmt: Containers do not provide a mechanism to quota manage the resources and hence susceptible to the “noisy neighbor” challenge.

Potential Solutions:

- Kernel Security Modules: SElinux, AppArmor
- Resource Mgmt: Kubernetes
- Platform Awareness: ClearLinux

Elasticity & Resilience

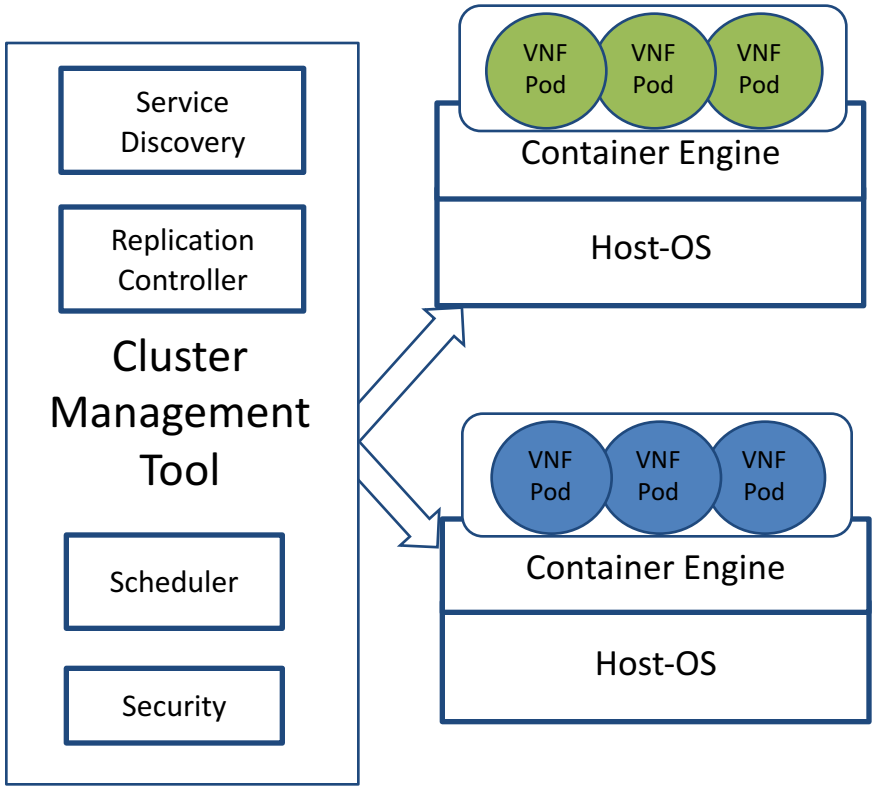


Key Benefits:

- Auto-scaling VNFs or achieving service elasticity in runtime can be simplified by the use of container based VNFs due to the lightweight resource usage of containers (e.g. Mesosphere/Kubernetes)

- Container management solutions (e.g. Kubernetes) provide self-healing features such as auto-placement, restart, and replacement by using service discovery and continuous monitoring

Operations & Management



Key Challenges:

- Containers are supported in selective operating systems such as Linux, Windows and Solaris
- In the current range of VNFs, many don't support Linux OS or other OSes such as Windows and Solaris

Potential Solutions:

- Hybrid deployment with VMs and containers can be envisioned, e.g. leverage ideas from Aptible technology currently used for applications

Conclusion and Future Work

Conclusion and Future Work

- Use of containers for VNFs appears to have significant advantages compared to using VMs and hypervisors, especially for efficiency and performance
  - "Virtual Customer CPE Container Performance White Paper," http://info.ixiacom.com/rs/098-FRB-840/images/Calsoft-Labs-CaseStudy2015.pdf
    - Test Setup:
      - COTS server with Intel Xeon E5-2680 v2 processor
      - Virtual CPE VNFs (Firewall etc.) fast path optimized using Intel DPDK
      - Measured L2-L3 TCP traffic throughput per core
    - VM (KVM) environment with SRIOV -- 5.8Gbps
    - Containers (LXC) environment -- 7.2Gbps
      - **~25% PERFROMANCE IMPROVEMENT OVER VMs**
- Opportunistic areas for future work
  - Distributed micro-service network functions
  - VNF Controller discovery/management/etc. standardization
  - etc.

Call for Action

- Address aforementioned challenges
- Further research to identify currently unknown challenges
- Vendors to consider developing container based solutions – especially to support proof of concepts and field trials
- Reach consensus on a common framework for use of containers for NFV
- Field trial container-based VNFs