# OpenRetriever Next Gen VIM & Edge Computing Scheduler Requirements Document

Created by the OPNFV OpenRetriever Team

Amar Kapadia

Wassim Haddad  
Heikki Mahkonen

Srinivasa Addepalli

v1.0 5/3/17

v1.1 5/16/17

V1.2 7/26/17

## Motivation

The OpenRetriever team believes that existing and new NFV workloads can benefit from a new VIM placement and scheduling component. We further believe that these same requirements will be very useful for edge computing scheduling. This document aims to document requirements for this effort.

By placement and scheduling, we mean:

* Choose which hardware node to run the VNF on factors such as AAA, ML prediction or MANO
* Start the VNF(s) depending on a trigger e.g. receiving requests such as DHCP, DNS or upon data packet or NULL trigger

We use the generic term “scheduler” to refer to the placement and scheduling component in the rest of this document. We are not including lifecycle management of the VNF in our definition of the scheduler.

At a high level, we believe the VIM scheduler must:

* Support virtual machines, containers and unikernels
* Support legacy and event-driven scheduling
  + By legacy scheduling we mean scheduling without any trigger (see above) i.e. the current technique used by schedulers such as OpenStack Nova.
  + By event-driven scheduling we mean scheduling with a trigger (see above). We do not mean that the unikernel or container that is going to run the VNF is already running . The instance is started and torn-down in response to traffic. The two step process is transparent to the user.
  + More specialized higher level schedulers and orchestration systems may be run on top e.g. FaaS (similar to AWS Lambda) etc.

|  |
| --- |
| Serverless vs. FaaS vs. Event-Driven Terminology  Serverless: By serverless, we mean a general PaaS concept where the user does not have to specify which physical or virtual compute resource their code snippet or function will run on. The code snippet/function is executed in response to an event.  FaaS: We use this term synonymously with serverless.  Event-Driven: By event-driven, we mean an entire microservice or service (as opposed a code snippet) is executed in response to an event. |

* Work in distributed edge environments

Please provide your inputs. Once we have a comprehensive list of requirements, we will investigate what the right open source solution should be, and how to influence that particular project.

## Use cases

A number of NFV use cases can benefit from a new VIM scheduler:

### vCPE

vCPE can benefit from a new scheduler in two ways:

1. uCPE devices have very few cores (4-8 typical). Running statically scheduled VMs is inefficient. An event-driven scheduler would help optimize the hardware resources and increase capacity.
2. vCPE is a bursty NFV use case, where services are not “on” all the time. Legacy provisioning of virtual machines for each VNF significantly reduces resource utilization, which in turn negatively impacts the total-cost-of-ownership (TCO). Recent Intel studies have shown, in certain cases, vCPE saves 30-40% TCO over physical functions. This number is hardly compelling, we believe it needs to be significantly higher to be of any interest. This can be accomplished by increasing utilization, which in turn can be achieved through event-driven scheduling.

### IOT/ MEC

IOT & multi-access edge computing ([MEC](http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing)) share many of the same characteristics as the uCPE. Though serverless functions increase the resource utilization, it does not provide ability for application developers to introduce traditional security functions. Serverless services that can be brought up on-demand basis provide increases resource utilization as well as ability to introduce security functions within the service. Additionally, there is need for low latency and high security as well. A new scheduler can help with these needs.

### 5G

5G brings with it a number of above requirements, but perhaps the one that stands out the most is price/ performance. By using containers and unikernels, the price/ performance ratio can be significantly improved. (Containers or unikernels result in ~10x density with Legacy scheduling; higher density is possible with event-driven scheduling.) 5G will also bring MEC and IOT needs from the prior use case.

### Security

Many traditional services are always-on. Always-on services provide enough time for attackers to find vulnerabilities and exploit them. By bringing up workloads on demand basis and terminating them upon completion of its usage, closes the time advantage attackers have. For example, in three tier architecture of “Web”, “App” and “DB”, following on demand bring up would reduce the attack surface

* On demand bring up of “DB” service upon “APP” layer request.
* On demand bringup of “APP” service upon “Web” layer authenticates the user.
* On demand bring up of “Web” service upon “DNS” request or upon seeing “SYN” packet

Workloads can be brought down upon inactivity or using some application specific methods. Thin services (implemented using unikernels & Clear containers) and fast schedulers are required to enable this kind of security.

## Detailed Requirements

### Multiple compute types

|  |  |
| --- | --- |
| Requirement | Details |
| Support for virtual machines | VMs are the most common form of VNFs, and are not going away anytime soon. A scheduler must be able to support VMs. In theory, the MANO software could use two VIMs: one for VMs and another for containers/ unikernels. However, we believe this is a suboptimal solution since the operational complexity doubles - now the ops team has to deal with two VIM software layers. Also, networking coordination between the two VIM layers becomes complex.  NOTE: Bare-metal server scheduling, e.g. OpenStack Ironic, is out-of-scope for this document. |
| Support containers | This need is clear, the future of VNFs seems to be containerized VNFs. Containers are 10x more dense than VMs and boot 10x faster. Containers will also accelerate the move to cloud-native VNFs. Some users may want nested scheduling e.g. containers in VMs or containers in containers. Nested scheduling is out-of-scope for this document. We will only focus on one layer of scheduling problem and expect the other layer of scheduler to be distinct and separate. |
| Support unikernels | Unikernels are lightweight VMs; with the same density of containers, but faster boot times than containers. Since unikernels are VMs and incredible small surface area, they have rock-solid security characteristics. Unikernels are also higher performance than VMs. For these reasons, unikernels could play an important role in NFV. The downsides with unikernels are i) they are new, ii) often tied to a programming language and iii) they require a software recompile. Unikernels are an ideal fit for micro-VNFs. More specifically:   * Need VNFs to be highly secure by reducing significantly the attack surface * Need to be able to schedule to NFVI with high performance OVS-less services chaining (e.g. through shared memory) that can significantly improve performance |
| Colocation | We need support for affinity/anti-affinity constraints on VNF compute type (i.e. VM, unikernel, container). This will make colocation of different types of VNF compute types on the same host possible, if needed. |
| Support all compute types on one SFC | Since VNFs are procured from different vendors, it is possible to get a mix of compute types: VMs, containers, unikernels; and it should be possible to construct a service function chain from heterogeneous compute types. |
| Unified API for all compute types | Even though it is theoretically possible to have different APIs for different compute types and push the problem to the MANO layer, this increases the overall complexity for the solution. For this reason, the API needs to be unified and consistent for different compute types. |
| Hardware awareness | Ability to place workloads with specific hardware or underlying infrastructure capabilities (e.g. Intel EPA[[1]](#footnote-0), FD.io, Smart NICs, Trusted Execution Environment, shared memory switching etc.) |
| Rich networking | The new VIM scheduler needs to be supported by rich networking features currently available to OpenStack Nova through OpenStack Neutron (See document outlining K8s [networking](https://docs.google.com/document/d/1TW3P4c8auWwYy-w_5afIPDcGNLK3LZf0m14943eVfVg/edit?ts=5901ec88) requirements as an example):   * Ability to create multiple IP addresses/ VNF * Networks not having cluster-wide connectivity; not having visibility to each other * Multi-tenancy: i) support traffic isolation between compute entities belonging to different tenants, ii) support overlapping IP addresses across VNFs. * Limit services such as load balancing, service discovery etc. on certain network interfaces (see additional [document](https://docs.google.com/document/d/1mNZZ2lL6PERBbt653y_hnck3O4TkQhrlIzW1cIc8dJI/edit)). * L2 and L3 connectivity (?) * Service Discovery |
| Image repository & shared storage | * Centralized/distributed image repository * Support shared storage (e.g. OpenStack Cinder, K8s volumes etc.) |

[OPEN QUESTION] What subset of the Neutron functionality is required here?

### Multiple scheduling techniques

|  |  |
| --- | --- |
| Requirement | Details |
| Legacy scheduling | This is the current technique used by OpenStack Nova and container orchestration engines. Legacy scheduling needs to be supported as-is. |
| Event-driven scheduling | This applies only to unikernels, since unikernels are the only compute type that can boot at packet RTT. Thus, the requirement is to be able to schedule and boot unikernel instances in response to events with <30ms of ms (e.g., event-driven type of scheduling) as a must-have and <10ms as a nice-to-have. |
| Distributed Scheduling | Since services need to be brought up at packet RTT, there could be requirement to distribute the scheduling across compute nodes. |
| Multi Stage scheduling | To enable scheduling of services at packet RTT, there is a need to divide the scheduling to at least two stages - Initial stage where multiple service images are uploaded to candidate compute nodes and second stage where distributed scheduler bring up the service using the locally cached images. |

[OPEN QUESTION] What subset of the rich scheduler feature-set is required here? (e.g. affinity, anti-affinity, understanding of dataplane acceleration etc.)

### Highly distributed environments

There are two possibilities here. A) The entire VIM will be in an edge device and the MANO software will have to deal with 10s or 100s of thousands of VIM instances. B) The alternative is that the VIM itself will manage edge devices, i.e. the MANO software will deal with a limited number of VIM instances. Both scenarios are captured below.

|  |  |
| --- | --- |
| Requirement | Details |
| Small footprint | It should be possible to run the VIM scheduler in 1-2 cores. |
| Nodes across WAN | It should be possible to distribute the VIM scheduler across nodes separated by long RTT delays (i.e. WAN). |

## Software Survey Candidates

Once the survey is complete, we will evaluate the following software stacks against those requirements. Each survey, either conducted in person and/or via documentation review, will consist of:

1. Architecture overview
2. Pros
3. Cons
4. Gap analysis
5. How gaps can be addressed

Each survey is expected to take 3-4 weeks.

|  |  |
| --- | --- |
| CNCF K8s | Srini (talk to Xuan, Frederic, study gap analysis) |
| Docker Swarm |  |
| VMware Photon | Srikanth |
| Intel Clear Container | Srini |
| Intel Ciao | Srini |
| OpenStack Nova |  |
| Mesos | Srikanth |
| Virtlet (VM scheduling by K8s) | Amar |
| Kubelet (VM scheduling by K8s) | Amar |
| Kuryr (K8s to Neutron interface) | Prem |
| RunV (like RunC) - can it support a VM |  |
| Nelson distributed container framework |  |
| Nomad |  |

## Additional Points to Revisit

* Guidance on how to create immutable infrastructure with complete configuration, and benefits to performance and security
* Guidance on API - VNFM vs. VIM

1. Intel EPA includes DPDK, SR-IOV, CPU and NUMA pinning, Huge Pages etc. [↑](#footnote-ref-0)