Network Service Mesh

An Introduction
Cloud 1.0 - Slap a ‘v’ in front of it

It turns out, most physical concepts are painful overkill in the cloud, bringing only needless pain and suffering.

None of the ‘v’ things were designed to evolve at speed of cloud.
Cloud 1.0 NFV

- Big
- Complex
- Heavy
- Expensive
- Lift and Shift Mentality
Cloud-native NFV: Step 1: Lift & Shift Management Plane

- Relatively simple
- Management Plane look like Apps
- Put into containers
- Only half way to Cloud-native
- Full transition will require architectural rethink
  - Blurred boundaries
Cloud-native NFV: Step 1: Lift & Shift Management Plane

Cloud 1.0 VNFs Trap us in Past

VNF

VNF

VNF

VNF

NFVi

Virtual Compute

Virtual Storage

Virtual Networking

Virtualization Layer

Hardware Resources

Hardware Compute

Hardware Storage

Hardware Networking

VIM

VNF Managers

Orchestrator

EMS

OSS/BSS
Cloud-native NFV: Step 2: VNF -> CNF

- **VNF**
  - VM Based
  - Big
  - Heavy
  - Magic Kernel based dataplane

- **CNF (Cloud-native Network Function)**
  - Containerized
  - Small
  - Light
  - Userspace dataplane
Cloud-native NFV: Anatomy of a CNF

Cloud-native management agent

Userspace Dataplane

CNF

VPP Agent

VPP

fd.io

ligato.io
Cloud-native NFV: Step 3: The Wiring Problem: **Network Service Mesh**

- How do we wire CNFs together into Service Function Chains to provide services to users
- In Cloud 1.0, we used heavy complex orchestration
- Cloud-native NFV needs a more Cloud-native approach

**Take inspiration from Service Mesh**
What is Network Service Mesh (NSM)?

“Network Service Mesh is a novel approach solving complicated L2/L3 use cases in Kubernetes that are tricky to address with the existing Kubernetes Network Model. **Inspired by Istio**, Network Service Mesh maps the concept of a service mesh to L2/L3 payloads.”

Translation for NFV folks. NSM provides a simple Cloud-native approach to:
- Local/Remote cross connects between CNFs
- Multi-interface
- Hardware NICs/SRIOV
- Service Function Chaining (SFC)
Network Service Mesh: Missing Layers

Service Mesh
- L7 (HTTP/2 etc)
- L4 (Streams(TCP)/Datagrams(UDP))

Payloads handled by Kubernetes Services/Istio/ServiceMesh

Network Service Mesh
- L3 (IP)
- L2 (Ethernet, MPLS, etc)

Payloads needed by NFV
Temptation: Port in Cloud 1.0 Solution

We know how to solve this in Cloud 1.0:
  - vInterface
  - vSubnet
  - Etc
Deep temptation to simply follow the old patterns, add User Facing APIs for:
  - Multi-interface per Pod
  - ‘Networks’
  - etc

Solution:
- Invent Cloud Native constructs to solve NFV Problems

Cloud Native (Kubernetes/Cloud Native/Service Mesh) folks do not want to regress by bringing back the Cloud 1.0 constructs

Opposition

Solution
Sarah

and...

secure-intranet-connectivity
Meet Sarah

Sarah is writing a Kubernetes app to be deployed in the public cloud

One of the Pods in Sarah’s app needs **secure access to her corporate intranet**
From Sarah's point of view her needs look like this
Sarah's Pod

L2/L3 connection

Corporate Intranet

Security goes here...

Sarah still wants her normal Kubernetes Networking...
But she also wants to send and receive traffic to her corporate intranet ...
How do I find out what subnet this connects to?
Sarah’s definition of hell...

Wait, who defines the subnet?

- Sarah’s Pod
  - interface
- Subnet
  - interface
- VPN Gateway Pod
  - interface
- VPN Concentrator
  - Corporate Intranet
Sarah’s definition of hell...

What if the subnet is too small in the future?
Sarah’s definition of hell...

I have to define the interface on this end too?
Sarah’s definition of hell...

And insert routes for all my corporate IPs into *my* Pod?
Sarah’s definition of hell...

I need more replica’s, the subnet is too small now. I have to do this all over again!
Sarah’s definition of hell...

What if the subnet changes? Do I have to change all this stuff too?
Sarah’s definition of hell...

Arrrg... my Enterprise Network guys say I chose a subnet that is incompatible with the corporate intranet. I have to redo all of this again?
Sarah’s definition of hell...

Wait… my network guys re-IPed *something* in the intranet and now my subnet is incompatible again… this is hell!
Sarah’s definition of hell...

Corporate added a new CIDR to the corporate network. Time to update the routes… why God! Why?
Then Sarah’s IT guys decide ‘secure’ also includes a Firewall Pod

Do I need a new subnet for this? Why am I doing network design with 1990s concepts?
Maybe I can help?
Who are you?

I’m the Ariadne the NSM (Network Service Mesh) Spider
Sarah: What is Network Service Mesh? Is that like Istio?

Sort of. You know all the cool things Istio does for you with TCP connections and HTTP?
Sarah: Yeah, it's awesome!

I do that for IP, Ethernet, and other L2/L3 protocols. Tell me about your problems.
I have a Pod and I just want to securely connect it to my corporate intranet.
Cool, this is how you would do it with NSM.

```yaml
kind: NetworkService
apiVersion: V1
metadata:
  name: secure-intranet-connectivity
spec:
  payload: IP
```
That looks a lot like a Service Resource in K8s.

Yep! But instead of the ‘payload’ being TCP streams, its L2/L3 payloads like 'ethernet' or ‘ip’
So I just need this and a Deployment for the VPN-GW Pod?

```
kind: NetworkService
apiVersion: V1
metadata:  
    name: secure-intranet-connectivity
spec:  
    payload: IP
```

Close! You would also want to insert our standard Network Service Mesh init-container and a Config Map telling it what Network Service you want to connect to into your Pod. That’s it.
No interfaces? No subnets? No routes? How does all this magic work?

```yaml
kind: NetworkService
apiVersion: V1
metadata:
  name: secure-intranet-connectivity
spec:
  payload: IP
```
Network Service Mesh has three basic concepts. You’ve already met the first: Network Service (NS).

It’s something you send L2/L3 payloads to and from and it does something you want.
Like giving me secure connectivity to my intranet?

Exactly!

Network Service
Example: secure-intranet-connectivity
The Second concept is a Network Service Endpoint (NSE).

It’s a Pod that is providing the Network Service you want.

Network Service

Example: secure-intranet-connectivity

Network Service Endpoint
Is that like Endpoints for Services?

Very much. We tried to use familiar concepts in Network Service Mesh.

Network Service
Example: secure-intranet-connectivity

Network Service Endpoint
So the VPN Gateway Pod is a Network Service Endpoint?

You're getting it! :)

Network Service
Example: secure-intranet-connectivity

Network Service Endpoint
Example: VPN Gateway Pod
The Third concept is the L2/L3 ‘connection’ between your Pod and the NSE.
Is that an interface?

Sarah

Usually it is instantiated as a kernel interface in your Pod.

There are NSM users with more complex use cases who want more exotic things like memif or vhost user, and we can give that to them.

You probably want a kernel interface.
What about subnets?

L2/L3 connections are point to point cross connects between your Pod, and the Network Service you want. You don’t have to think about subnets.

If you want a Bridge Domain, that’s a Network Service itself.
And routes? My network guys added an intranet CIDR last month and I had to update the routes on *all* of my Pods. It sucked!

Generally, **addresses and routes** for an L2/L3 connection come from the **Network Service Endpoint**, like your VPN Gateway. They are in a better position to know what they should be.

Advanced use cases can be done with more flexibility, but that’s probably not what you want here.
What about the new Firewall Pod they want to get traffic before the VPN Gateway Pod?

Network Service Mesh is a **Mesh**.

The Firewall Pod and the VPN Gateway Pod work **together** to give you the Network Service you want:

**secure-intranet-connectivity**
How does that work?

You know VirtualServices in Istio?
Network Service Mesh extends the Network Service with a very similar concept. In Istio VirtualServices help select what Endpoint to connect to for a Service. In Network Service Mesh we do something quite similar for L2/L3 Connections.

Sarah

Yeah, they are super useful.
When a NetworkServiceEndpoint exposes a Network Service, it can attach ‘Labels’ (key=value pairs). We call these Destination Labels. In this example “Firewall Pod” would have DestinationLabel “app=firewall” and “VPN Gateway Pod” would have label “app=vpn-gateway”.

Yeah, they are super useful.
When a Pod requests an L2/L3 Connection to a NetworkService, it can attach ‘Labels’ (key=value pairs) to that request. We call these SourceLabels. So in this example, when the Firewall Pod asks for an L2/L3 connection to the secure-intranet-connectivity Network Service, it uses SourceLabel ‘app=firewall’.

Yeah, they are super useful.
We can then enhance our Network Service to tell it that when it gets a request for an L2/L3 Connection with SourceLabels `app=firewall`...
kind: NetworkService
apiVersion: V1
metadata:
  name: secure-intranet-connectivity
spec:
  payload: IP
  matches:
    - match:
        sourceSelector:
          app: firewall
        route:
          - destination:
              destinationSelector:
                app: vpn-gateway

... It should route them to a Network Service Endpoint with Destination Label 'app=firewall'
OK. How does my Pod get to the Firewall Pod?

```yaml
kind: NetworkService
apiVersion: v1
metadata:
  name: secure-intranet-connectivity
spec:
  payload: IP
  matches:
  - match:
      sourceSelector:
        app: firewall
      route:
        - destination:
            destinationSelector:
              app: vpn-gateway
```
kind: NetworkService
metadata:
  name: secure-intranet-connectivity
spec:
  payload: IP
  matches:
  - match:
      sourceSelector:
        app: firewall
      route:
      - destination:
        destinationSelector:
          app: vpn-gateway
    - match:
      route:
      - destination:
        destinationSelector:
          app: firewall

We add an ‘empty’ match which matches any connection request...
And route them to Network Service Endpoints with DestinationLabel 'app=firewall'.

```yaml
kind: NetworkService
metadata:
  name: secure-intranet-connectivity
spec:
  payload: IP
  matches:
  - match:
      sourceSelector:
        app: firewall
      route:
      - destination:
        destinationSelector:
          app: vpn-gateway
      - match:
          route:
          - destination:
            destinationSelector:
              app: firewall
```
So when IT decides to put something else in there for more security, I just add a Deployment for that and update the NetworkService?

Yep!
Sarah: No interfaces, no IPs, no subnets, no routes?

Nope!
Network Service Mesh is completely orthogonal to normal Kubernetes Networking. It doesn't use CNI, and you can use it with your existing Kubernetes and CNI.

Do I need a new version of Kubernetes?
Do I have to use a specific CNI plugin?

- K8s upgrades
- Special CNI Plugin
Sarah: You know how Istio Service Mesh's are often described as handling things like Service Routing etc?

Computer A

Service A  Business Logic
Sidecar  Service Discovery/Routing

Connection Protocol Stack
Networking Stack

Computer B

Sidecar  Service Discovery/Routing
Service B  Business Logic

Connection Protocol Stack
Networking Stack

That's awesome, how does it work?

You know how Istio Service Mesh's are often described as handling things like Service Routing etc?
Network Service Mesh has something like the Envoy Proxy that does the Service Discovery and Routing too. It’s called the:

**Network Service Manager (nsmd)**

It runs as a DaemonSet so you have one on each Node.

Yep! I’ve read [Phil Calçado](https://example.com)'s paper.
The NSM InitContainer reads your Config Map, and sends a GRPC call across a unix file socket to the nsmd to Request an L2/L3 Connection to the secure-intranet-connectivity Network Service.
Request Connection has any information needed to be clear about how you want the connection to look to your Pod locally, like that you want it to be a kernel interface, the interface name you’d prefer if you care, etc.
Let’s talk first about the case where your VPN Gateway Pod is on the same Node.

1. Request (Connection)
The nsmd sends a Request Connection to the VPN Gateway Pod, which sends back an Reply.
The nsmd creates an injects an interface into the VPN Gateway Pod
The nsmd creates an injects an interface into Sarah’s Pod.
The nsmd then cross connects the two interfaces.
And finally sends back a Reply to the NSM InitContainer. The NSM InitContainer then exits, and the Pod proceeds normally.

You are ready to go
So the NSM does Service Discovery and sets up the connection?

Yep.

For L2/L3 connections we don’t have a universal mechanism like TCP in the kernel, so we have the nsmd do it.
What if the VPN Gateway Pod is on a different Node.
If the VPN Gateway Pod is on a different Node, it looks exactly the same to your Pod.

You send a Request Connection request to NSM using GRPC over a unix file socket.
If the VPN Gateway Pod is on a different Node, it looks exactly the same to your Pod.

You send a Request Connection request to NSM using GRPC over a unix file socket.
nsmd1 then looks up (or more likely caches) NetworkServiceEndpoints and NetworkService from the K8s API Server and selects a Network Service Endpoint for the secure-intranet-connectivity Network Service.

From that NetworkServiceEndpoint resource it learns how to reach nsmd2.
nsmd1 sends a Request Connection GRPC call to nsmd2, listing out what its preferences are for tunnel mechanism (VXLAN, GRE, etc) and its preferences for parameters (VNI, etc.)
nsmd2 sends a Request Connection to the VPN Gateway which sends back a Reply.
nsmd2 creates and injects an interface into the VPN Gateway Pod for the connection.
nsmd2 sets up its end of the tunnel.

1. Request Container
2. Select based on NetworkServiceEndpoint
3. Response Container
4. Request Container
5. Reply
6. Create & Inject Interface
7. Create Tunnel
nsmd2 send an Reply back to nsmd1 with the selected tunnel mechanism and tunnel parameters.
nsmd1 creates and injects an interface for the connection into Sarah’s Pod.
nsmd1 creates its end of the tunnel and completes the cross connect.
Finally nsmd1 sends an Accept back to the InitContainer your Pod.
Sarah: Cool! This looks so much easier than trying to manually string together interfaces and subnets myself! Thank you!

You are welcome! Have fun!
Cloud-native NFV: Step 3: CNFs + Network Service Mesh

Cloud 1.0 VNFs Trap us in Past

VNF
VNF
VNF

NFVi

Virtual Compute
Virtual Storage
Virtual Networking

Virtualization Layer

Hardware Resources

Hardware Compute
Hardware Storage
Hardware Networking

VIM

VNF Managers

Orchestrator

EMS

OSS/BSS
Cloud-native NFV: Step 3: CNFs + Network Service Mesh

- VNF Managers
- Orchestrator
- EMS
- OSS/BSS

Hardware Resources:
- Hardware Compute
- Hardware Storage
- Hardware Networking

Network Service Mesh of CNFs
Cool things we haven’t covered here

- How Network Service Mesh lets you interact cross-cluster and with external physical providers of Network Services
  - External Network Service Managers (eNSMs)
- How Network Service Mesh handles physical NICs and SRIOV
- How Network Service Mesh allows global hinting
  - Proxy Network Service Managers (pNSMs)
How to get involved with Network Service Mesh

- networkservicemesh.io
  - Broad open source community working on Network Service Mesh
  - Lots of Network Service Mesh happenings at Kubecon:
    - https://www.networkservicemesh.io/events/kubeconna2018/