Network Service Mesh

A Narrative Introduction
Frederick Kautz and Kyle Mestery
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
- Security goes here...
- Corporate Intranet

K8s interface
Sarah still wants her normal Kubernetes Networking...
But she also wants to send and receive traffic to her corporate intranet...
Sarah’s definition of hell...

How do I find out what subnet this connects to?
Sarah’s definition of hell...

Wait, who defines the subnet?
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?

K8s interface

Sarah's Pod

Subnet

VPN Gateway Pod

VPN Concentrator

Corporate Intranet
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!
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?

Sarah
Corporate added a new CIDR to the corporate network. Time to update the routes... why God! Why?
Maybe I can help?
Sarah

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!

Me: 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.
Sarah

Cool, this is how you would do it with NSM.

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

Yep! You use selectors on Pods to find the Pods providing the Network Service, and it exposes ‘payloads’ instead of ports and tcp.

```yaml
kind: NetworkService
apiVersion: V1
metadata:  
  name: secure-intranet-connectivity
spec:  
  selector:  
    app: secure-intranet-gateway
  payload: IP
```
So I just need this and a Deployment for the VPN-GW Pod?

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.

```yaml
kind: NetworkService
apiVersion: v1
metadata:
  name: secure-intranet-connectivity
spec:
  selector:
    app: secure-intranet-gateway
  payload: IP
```
No interfaces? No subnets? No routes? How does all this magic work?

```yaml
kind: NetworkService
apiVersion: V1
metadata:
  name: secure-intranet-connectivity
spec:
  selector:
    app: secure-intranet-gateway
  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.
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

Sarah
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.

Example: secure-intranet-connectivity

Network Service Endpoint Example: VPN Gateway Pod
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.
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 VirtualHosts/RouteRules in Istio?
Network Service Mesh has an analogous concept, we call them:

**Network Service Wirings**

Sarah's Pod

---

Yeah, they are super useful.

---

**Network Service**

secure-intranet-connectivity

- L2/L3 connection
- Firewall Pod
- L2/L3 connection
- VPN Gateway Pod
The ‘secure-intranet-connectivity-wiring-1’ Network Service Wiring says:

‘target: secure-intranet-connectivity’

That means it applies to L2/L3 connections trying to reach the secure-intranet-connectivity Network Service.

```
kind: NetworkServiceWiring
groupVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-1
spec:
  target:
    - secure-intranet-connectivity
  qualifiers:
    source:
      sourceService:
        !secure-intranet-connectivity
  action:
    route:
      -destination:
        podSelector:
          firewall=true
```
The ‘secure-intranet-connectivity-wiring-1’ **Network Service Wiring** then has a qualifier that only matches things **not** providing the secure-intranet-connectivity service themselves.

Things like your Pod.

```yaml
kind: NetworkServiceWiring
apiVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-1
spec:
  target:
    - secure-intranet-connectivity
  qualifiers:
    source:
      sourceService:
        !secure-intranet-connectivity
  action:
    route:
      -destination:
        podSelector:
          firewall=true
```
And it has a ‘route’ that sends those connections to one of the Network Service Endpoints with label ‘firewall=true’... like the Firewall Pod.

```yaml
kind: NetworkServiceWiring
apiVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-1
spec:
  target:
    - secure-intranet-connectivity
  qualifiers:
    source:
      sourceService:
        !secure-intranet-connectivity
  action:
    route:
      -destination:
        podSelector:
          firewall=true
```
When your Pod tries to connect to the secure-intranet-connectivity Network Service, it matches the Network Service Wiring:

```
kind: NetworkServiceWiring
apiVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-1
spec:
  target:
    - secure-intranet-connectivity
  qualifiers:
    source:
      sourceService:
        - secure-intranet-connectivity
  action:
    route:
      -destination:
        podSelector:
          firewall=true
```
kind: NetworkServiceWiring
apiVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-1
spec:
target:
  - secure-intranet-connectivity
qualifiers:
  source:
    sourceService:
      !secure-intranet-connectivity
action:
  route:
    -destination:
      podSelector:
        firewall=true

Network Service Wiring
secure-intranet-connectivity-wiring-1
Causes your connection to be routed to the Firewall Pod
kind: NetworkServiceWiring
apiVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-2
spec:
  target:
    - secure-intranet-connectivity
  qualifiers:
    source:
      podSelector:
        firewall=true
  action:
    route:
      -destination:
        podSelector:
          vpngateway=true

OK. How does the Firewall Pod get connected to the VPN Gateway Pod?
That has a qualifier that matches sources that have label firewall=true

```yaml
kind: NetworkServiceWiring
apiVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-2
spec:
  target:
    - secure-intranet-connectivity
  qualifiers:
    source:
      podSelector:
        firewall=true
  action:
    route:
      -destination:
        podSelector:
          vpngateway=true
```
kind: NetworkServiceWiring
apiVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-2
spec:
  target:
    - secure-intranet-connectivity
  qualifiers:
    source:
    podSelector:
      firewall=true
  action:
    route:
      -destination:
        podSelector:
          vpngateway=true
The Firewall Pod does its Firewall thing, but doesn’t know anything about how to connect to your intranet. So it opens a connection to secure-intranet-connectivity itself.

The Firewall Pod’s connection matches Network Service Wiring:

```yaml
kind: NetworkServiceWiring
apiVersion: v1
metadata:
  name: secure-intranet-connectivity-wiring-2
spec:
  target:
    - secure-intranet-connectivity
  qualifiers:
    source:
      podSelector:
        firewall=true
  action:
    route:
      -destination:
        podSelector:
          vpngateway=true
```

Network Service

secure-intranet-connectivity

- Sarah's Pod
  - L2/L3 connection
  - Firewall Pod
    - firewall=true
  - L2/L3 connection
  - VPN Gateway Pod
    - vpngateway=true
kind: NetworkServiceWiring
apiVersion: V1
metadata:
  name: secure-intranet-connectivity-wiring-2
spec:
target:
  - secure-intranet-connectivity
qualifiers:
  source:
    podSelector:
      firewall=true
action:
  route:
    -destination:
      podSelector:
        vpngateway=true

causes the Firewall Pod's connection to be routed to the VPN Gateway Pod
So when IT decides to put something else in there for more security, I just add a Deployment for that and a couple of Network Service Wiring Resources? 

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
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 (NSM)**

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

Yep! I’ve read [Phil Calçado](https://www.philcalacao.com/)’s paper.
The NSM InitContainer reads your Config Map, and sends a GRPC call across a unix file socket to the NSM 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 NSM sends a Request Connection to the VPN Gateway Pod, which sends back an Accept.
The NSM creates an interface for the connection for the VPN Gateway and injects it into the VPN Gateway Pod.
The NSM creates and injects an interface into Sarah’s Pod.
The NSM then cross connects the two interfaces in the dataplane.
Finally the NSM responds to your Pod’s NSMInitContainer with an Accept.

Then you are ready to go.

1. Request Connection
2. Req Con
3. Accept
4. Create & Inject Interface
5. Create & Inject Interface
6. Cross connect
7. Accept
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 NSM do it.
What if the VPN Gateway Pod is on a different Node.

1. Request Connection
2. Req Con
3. Accept
4. Create & Inject Interface
5. Create & Inject Interface
6. Cross connect
7. Accept

Dataplane (kernel/vswitch)
cross-connect
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.
NSM1 then looks up (or more likely caches) NetworkServiceEndpoints and NetworkServiceWiring 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 NSM2.
NSM1 sends a Request Connection GRPC call to NSM2, listing out what its preferences are for tunnel mechanism (VXLAN, GRE, etc) and its preferences for parameters (VNI, etc.)
NSM2 sends a Request Connection to the VPN Gateway which sends back an Accept.
NSM2 creates and injects an interface into the VPN Gateway Pod for the connection.
NSM2 sets up its end of the tunnel.
1. Req Con

2. Select based on NetworkServiceEndpoint
   NetworkServiceWiring

3. Req Con

4. Req Con

5. Accept

6. Create & Inject Interface

7. Create tunnel

8. Accept

NSM2 send an Accept back to NSM1 with the selected tunnel mechanism and tunnel parameters.
NSM1 creates and injects an interface for the connection into Sarah's Pod.
NSM1 creates its end of the tunnel and completes the cross connect.
1. Request connection from the Pod to NSM1.
2. Select based on NetworkServiceEndpoint NetworkServiceWiring.
3. Request connection from the InitContainer to NSM1.
4. Request connection from NSM1 to NSM2.
5. Accept connection from NSM2 to NSM1.
6. Create and inject interface from NSM2 to the Dataplane (kernel/vswitch).
7. Create tunnel from NSM2 to NSM1.
8. Accept connection from NSM1 to NSM2.
9. Create and inject interface from NSM1 to the Dataplane (kernel/vswitch).
10. Create tunnel from NSM1 to NSM2.
11. Accept connection from NSM2 to NSM1.

Finally, NSM1 sends an Accept back to the InitContainer your Pod.
How do the NetworkEndpoint Resources get into the API Server? Do I have to put them there?

Nope, let's talk through that.
When the VPN Gateway Pod comes up, it exposes to the NSM1 that it has a secure-intranet-connectivity Network Service willing to accept connections.
Then the NSM creates a NetworkServiceEndpoint CRD for it in the K8s API server.
Cool! This looks so much easier than trying to manually string together interfaces and subnets myself! Thank you!

You are welcome! Have fun!
Thank you!