# Cloud-Native Networking, Home Edition

## Build and connect your VPCs with the Open Network Fabric

Quentin Monnet
<quentin@hedgehog.cloud>

FOSDEM – 2025-02-02

# Open Network Fabric

Make on-premises cloud infrastructure **easy**
by abstracting the network away

**Agenda**

- Can I have a cloud at home, please?
- What's *Open Network Fabric*?
- What does it look like?

# Can I have a cloud at home, please?

# The cloud: back to the origins*

In the beginning…

- There were binaries and libraries and messy dependencies

Let there be virtualization!

- … And there were virtual machines
- Too many resources, too slow

Let there be containers!

- … And there was Docker
- But how to orchestrate the containers?

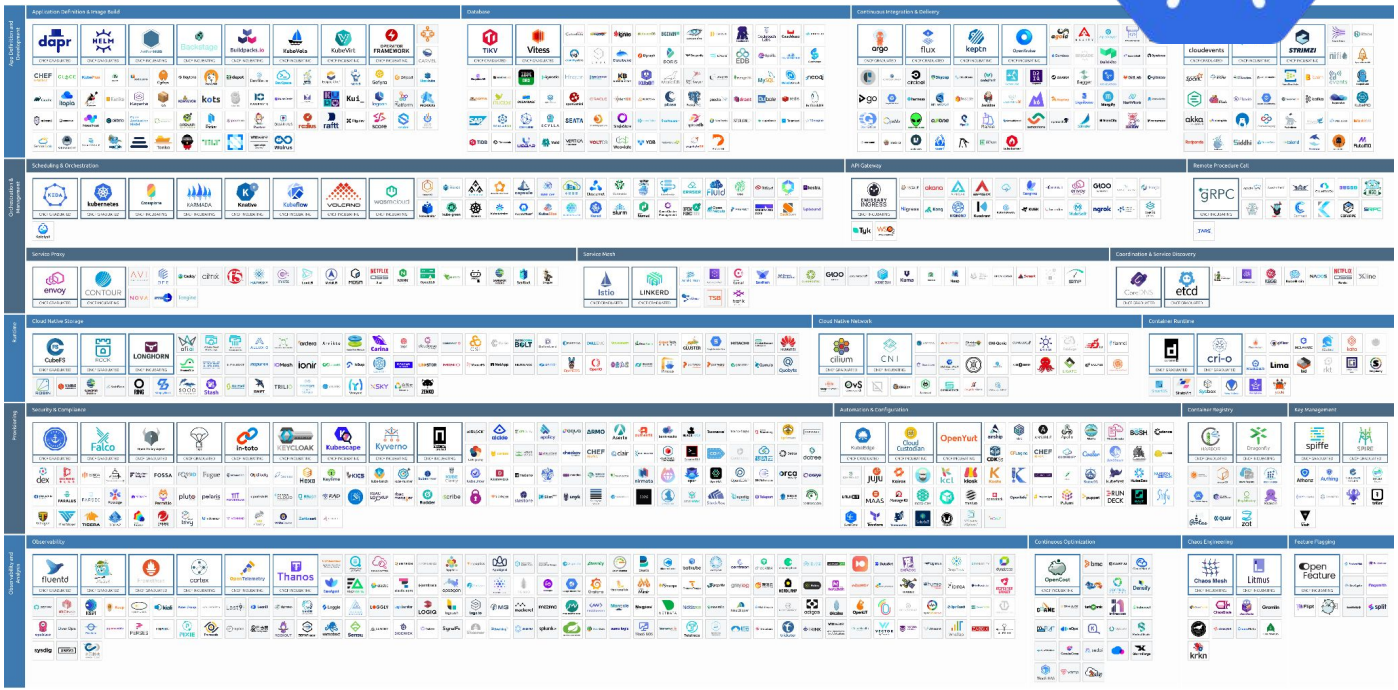
* Warning: Oversimplification detected. This recollection of events may not be 100% accurate.

# “We need a ship’s wheel with 7 handles”

… And all that comes with it.

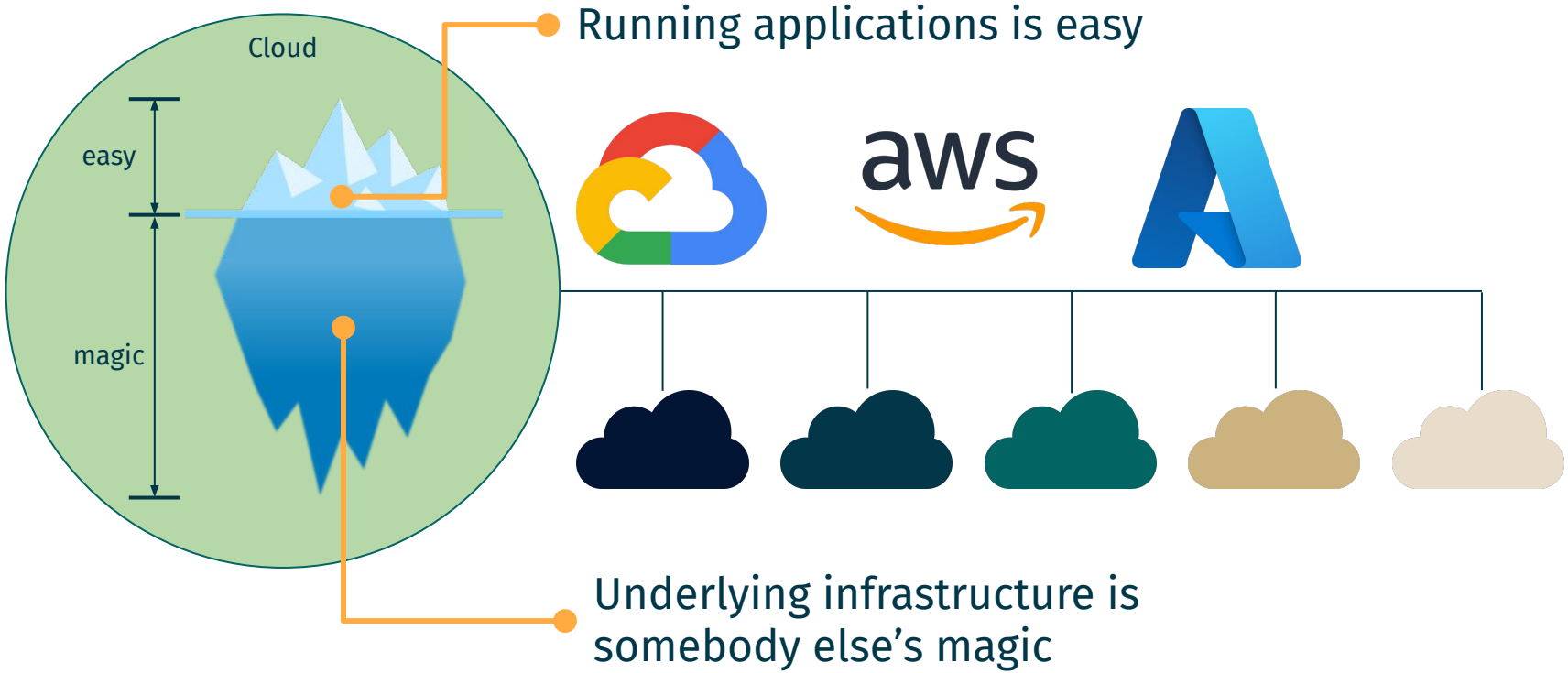
# “We need a ship’s wheel with 7 handles”

… And all that comes with it.

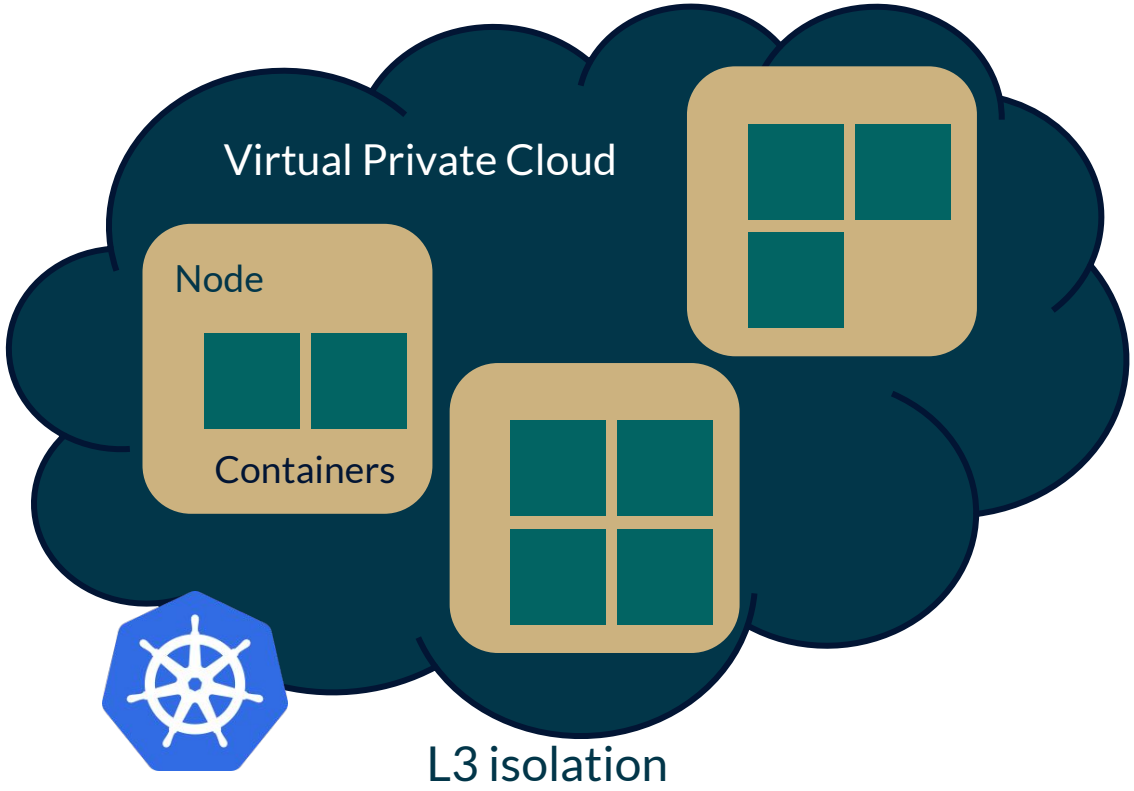


And all became immediately simpler.

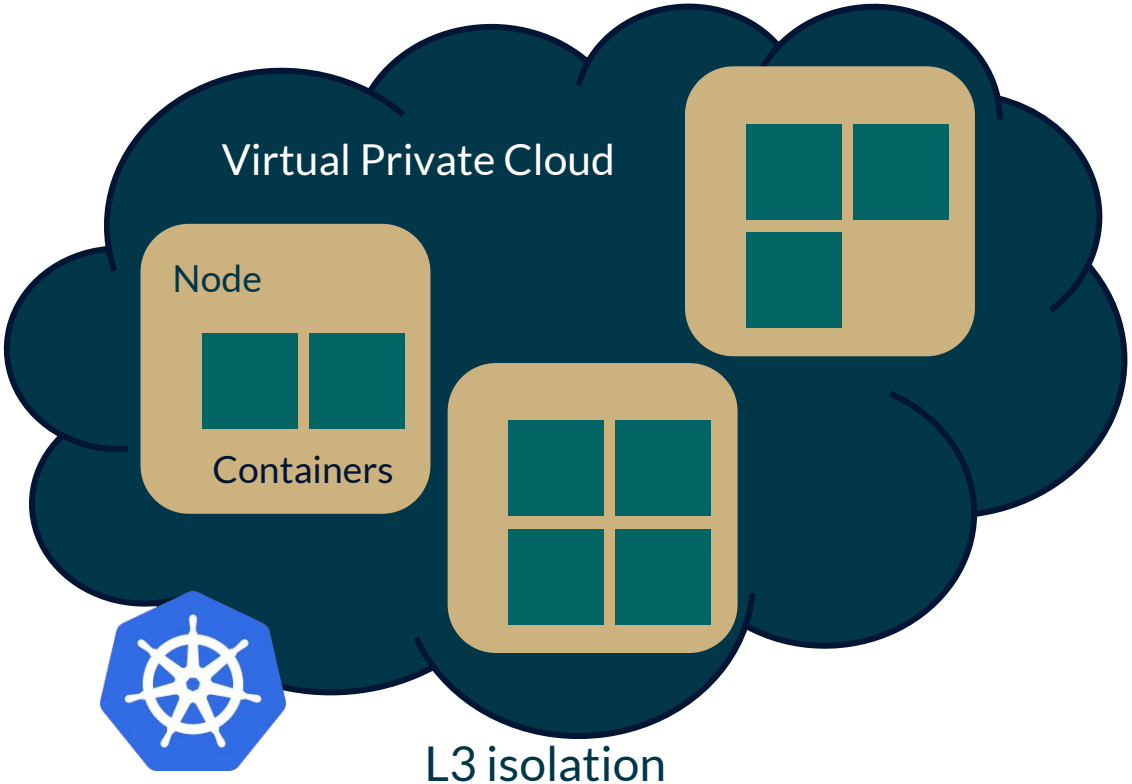
# Cloud providers to the rescue



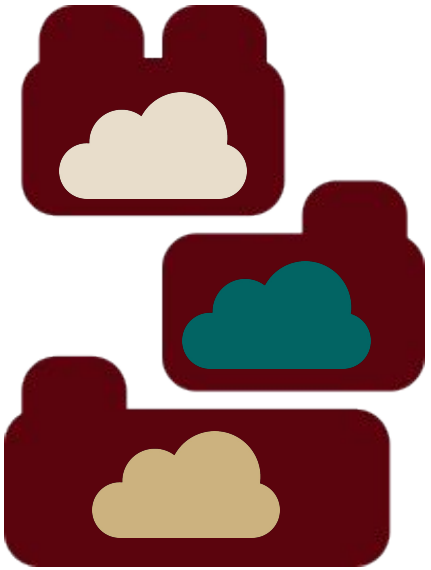
# Virtual Private Clouds



# Virtual Private Clouds




L3 isolation



Building blocks for a cloud infrastructure

# What if I want a cloud on my own hardware?

Motivations

- Costs: buying GPUs can be more interesting than renting them
- Latency: Edge Computing to move compute closer to source of data
- Compliance: on-prems processing required

Use cases

- Cloud decentralisation (specialised, distributed clouds)
- Edge computing (smart city, industrial IoT, 5G edge)
- What else? 🤔

# What if I want a cloud on my own hardware?

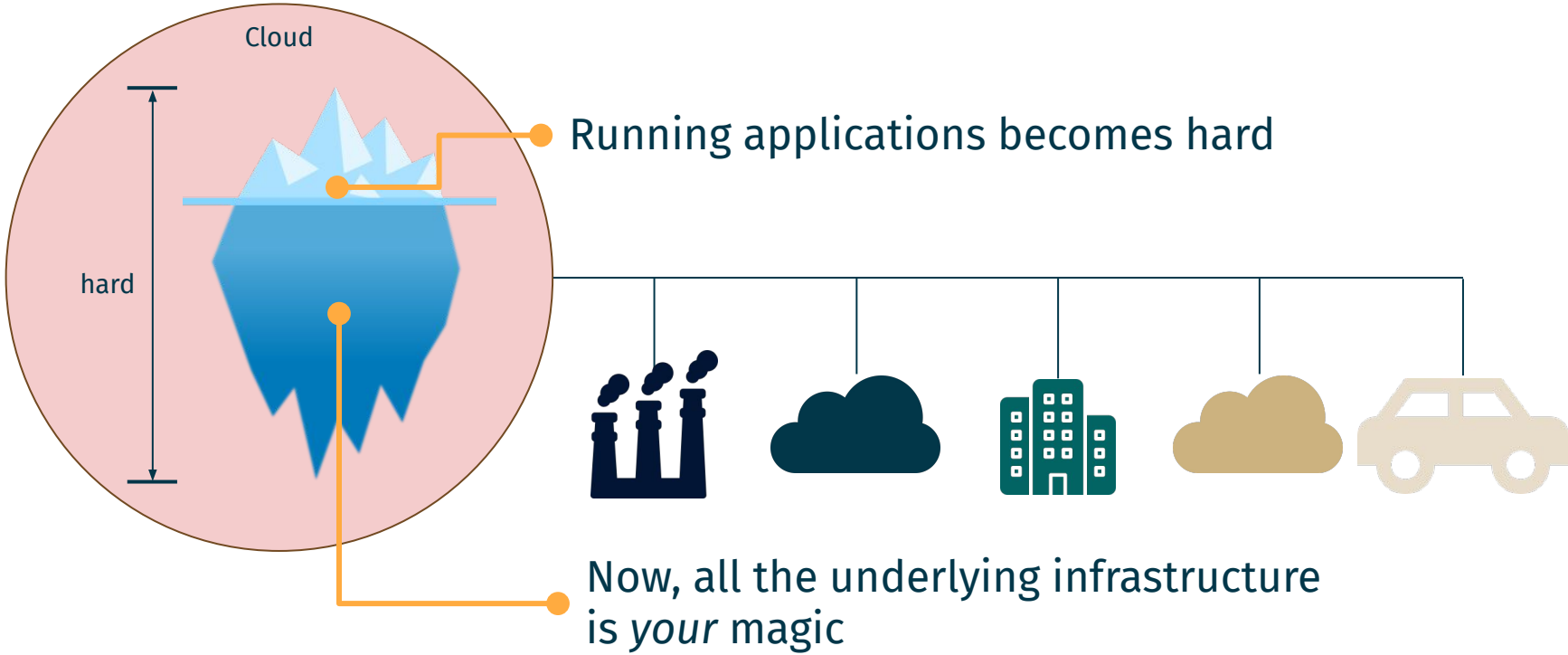
Motivations

- Costs: buying GPUs can be more interesting than renting them
- Latency: Edge Computing to move compute closer to source of data
- Compliance: on-prems processing required

Use cases

- Cloud decentralisation (specialised, distributed clouds)
- Edge computing (smart city, industrial IoT, 5G edge)
- What else? 🤔

# Building a cloud on-premises



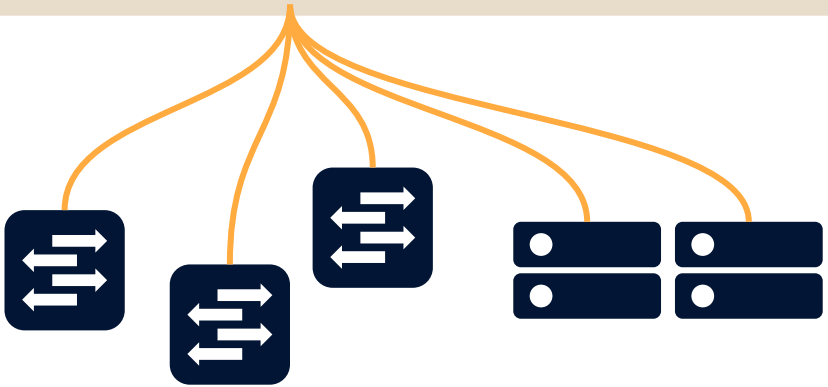
# Open Network Fabric

# Objective: deploy a cloud infra on commodity hardware

User’s hardware: branded or white-box commodity switches, servers

# Objective: deploy a cloud infra on commodity hardware

Network: connectivity, observability, services

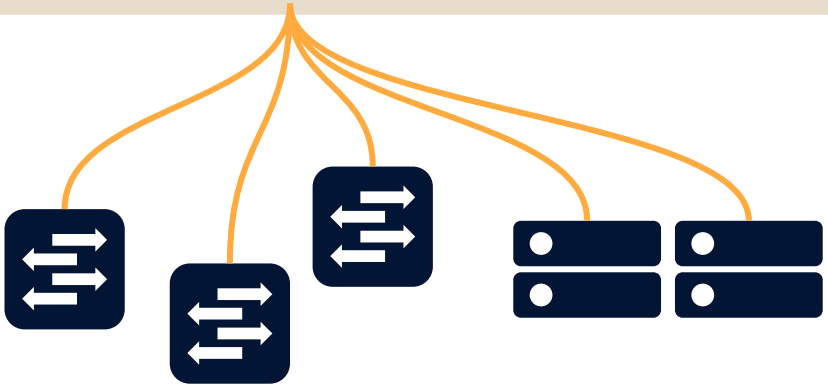


User's hardware: branded or white-box commodity switches, servers

# Objective: deploy a cloud infra on commodity hardware

Virtual Private Clouds (just like with providers)

Network: connectivity, observability, services

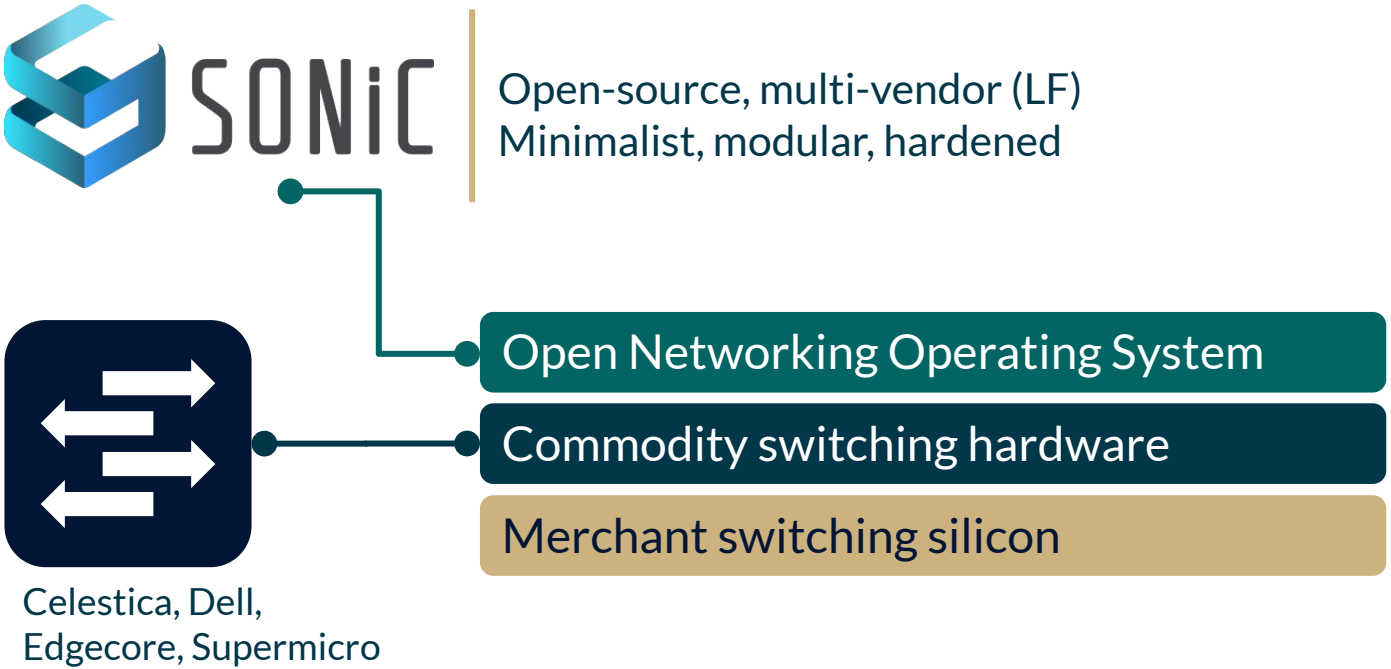


User's hardware: branded or white-box commodity switches, servers

# What components in Open Network Fabric?


Celestica, Dell,
Edgecore, Supermicro

# SONiC operating system for switches



# Kubernetes as Control Plane

Works and feels like a K8s cluster

Integrates into cloud-native operational and observability stacks

GitOps, Infrastructure-as-Code


Open-source, multi-vendor (LF)
Minimalist, modular, hardened

Control Plane

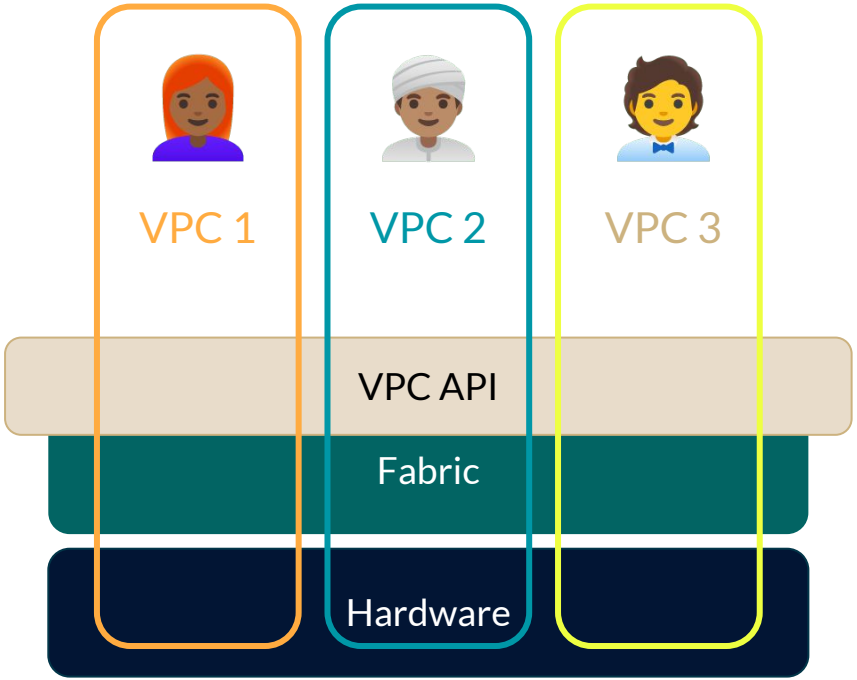
Open Networking Operating System

Commodity switching hardware

Merchant switching silicon

Celestica, Dell,
Edgecore, Supermicro

# Result: VPC as a Service




Open Network Fabric provides a VPC API

Same abstractions and tools as with cloud providers

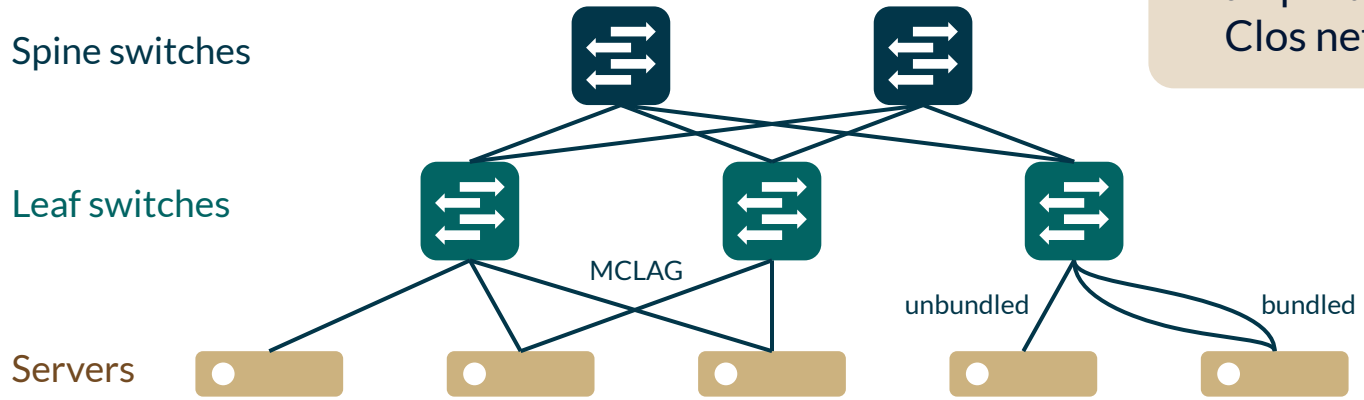
Under the hood: VXLAN-based BGP EVPN

Flexible policies and services

- VPC Policies
- Simple peering API
  (local intra/inter-VPC, external)
- IPAM, DHCP, DHCP-relay

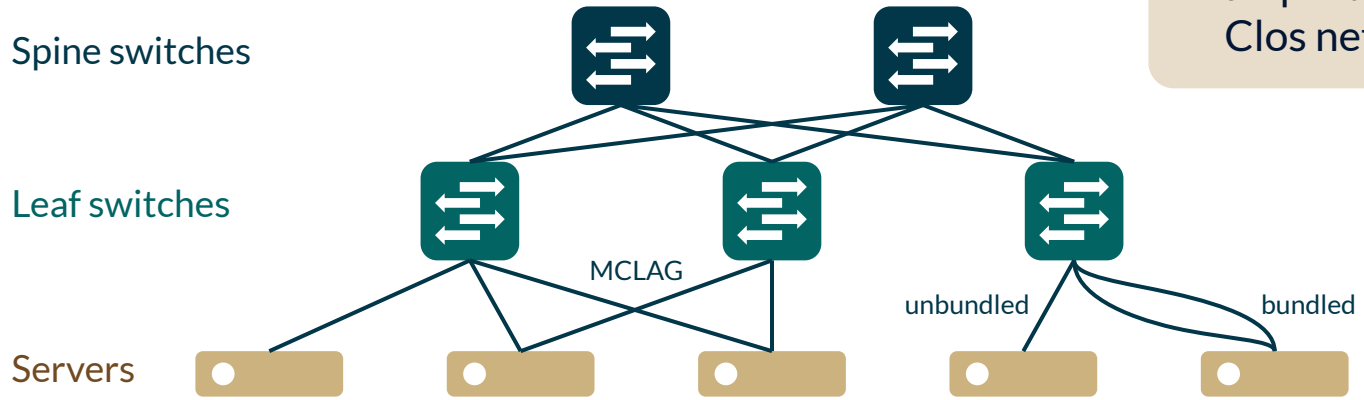
# Workflow

Example topology:
Clos network



# Workflow

Example topology:
Clos network

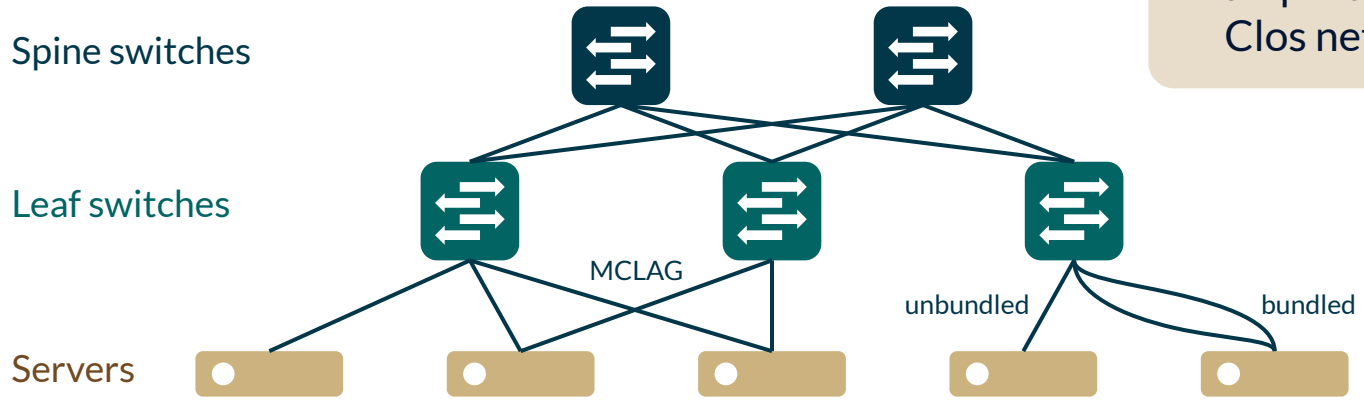



yaml Wiring diagram ✍️

yaml Metadata (Switch profiles)

# Workflow

Example topology:
Clos network




yaml Wiring diagram ✍️

yaml Metadata (Switch profiles)

magic!

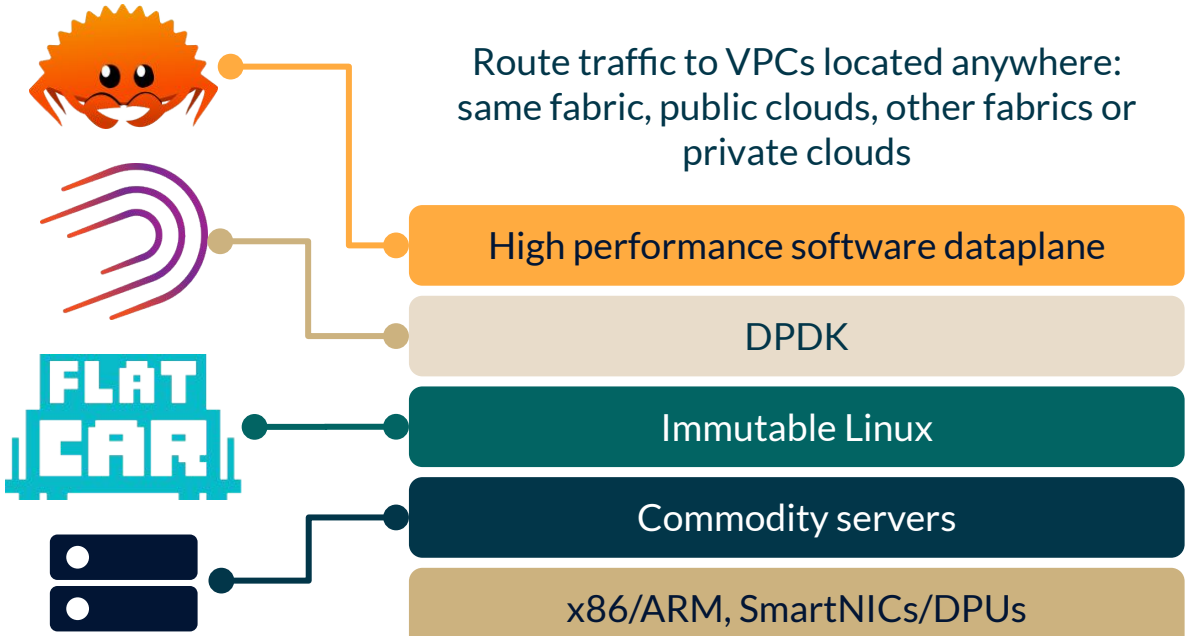
Switches automatically bootstrapped, imaged, configured

“Zero-touch” provisioning, updates, maintenance

Get VPCs; no network knowledge required 👌

# The Gateway

Work in progress




User's hardware: branded or white-box commodity servers and DPUs

# The Gateway

Work in progress

Inter-VPC routing

NAT/PAT

L3/L4 SLB

VPC QoS

Inter* gateway

VXLAN

Firewall

IPsec WireGuard

...

Route traffic to VPCs located anywhere: same fabric, public clouds, other fabrics or private clouds


High performance software dataplane

DPDK

Immutable Linux

Commodity servers

x86/ARM, SmartNICs/DPUs

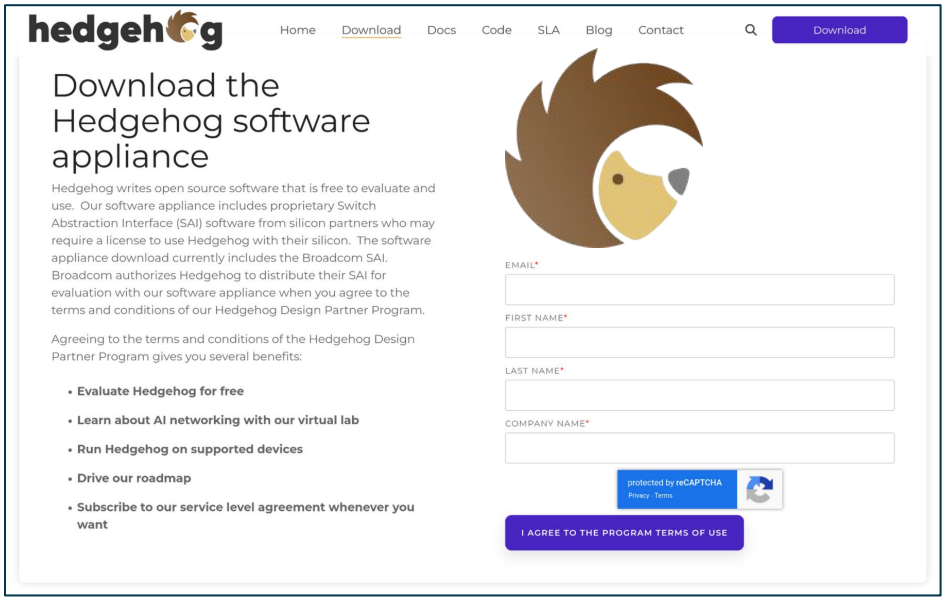
User's hardware: branded or white-box commodity servers and DPUs

# What does it look like?

# The elephant in the room

https://en.wikipedia.org/wiki/Elephant_in_the_room#/media/File:Elephant's_tea_party,_Robur_Tea_Room,_Sydney,_24_March_1939_-_Sam_Hood_(3529604677).jpg

# ⚠️ Open-source with caveats



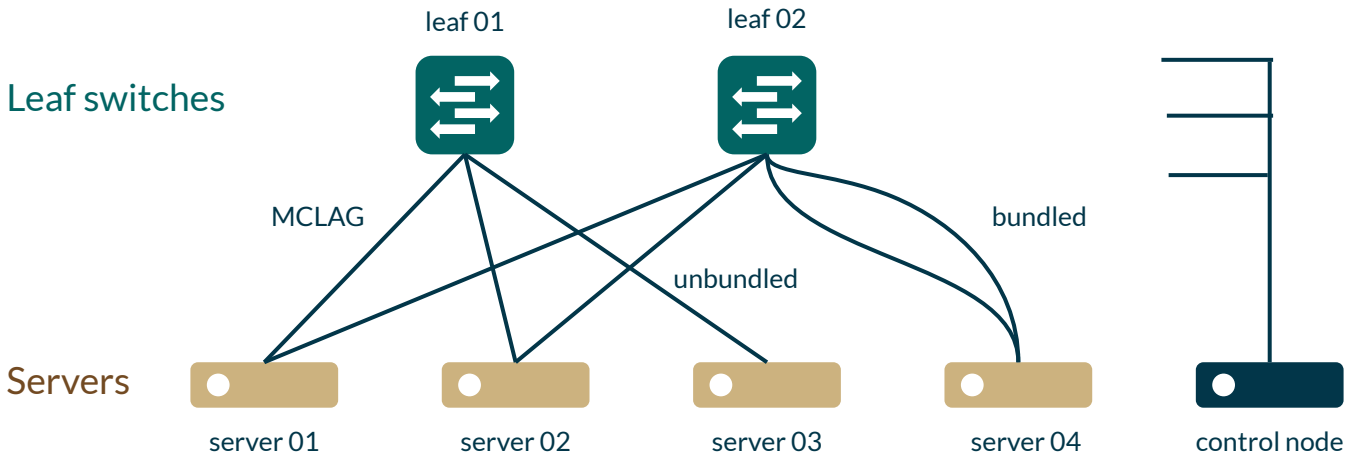

Issues we need to fix:

- Registration required for testing the project 😭
  (Broadcom SONiC image needed, upstream SONiC not yet supported)

- Confusion between project and product, “Hedgehog Fabric” everywhere in docs.
  Hedgehog Fabric == Open Network Fabric

But all code for Open Network Fabric itself *is* open-source

# Sample topology for the demo




Objective: deploy fabric, create two VPCs on servers 1-2, connect them

First step: install tools, then init, download, build, bootstrap the Fabric

```
ubuntu@vlab-quentin:~$ docker login ghcr.io -u <username> -p <password>
ubuntu@vlab-quentin:~$ curl -fsSL https://i.hhdev.io/oras | bash
ubuntu@vlab-quentin:~$ curl -fsSL https://i.hhdev.io/hhfab | bash
ubuntu@vlab-quentin:~$ hhfab --version
hhfab version v0.32.1


ubuntu@vlab-quentin:~$ hhfab init --dev --fabric-mode collapsed-core
15:25:53 INF Hedgehog Fabricator version=v0.32.1
15:25:53 INF Generated initial config
15:25:53 INF Adjust configs (incl. credentials, modes, subnets, etc.)
file=fab.yaml
15:25:53 INF Include wiring files (.yaml) or adjust imported ones
dir=include
ubuntu@vlab-quentin:~$ ls fab.yaml
Fab.yaml


ubuntu@vlab-quentin:~$ hhfab vlab gen
15:26:27 INF Hedgehog Fabricator version=v0.32.1
15:26:27 INF Building VLAB wiring diagram fabricMode=collapsed-core
15:26:27 INF >>> mclagLeafsCount=2 mclagSessionLinks=2 mclagPeerLinks=2
15:26:27 INF >>> orphanLeafsCount=0 vpcLoopbacks=2
15:26:27 INF >>> mclagServers=2 eslagServers=2 unbundledServers=1
bundledServers=1
15:26:27 INF Generated wiring file name=vlab.generated.yaml
```

Register for these

Initialisation; creation of fab.yaml

Generate wiring diagram

```
ubuntu@vlab-quentin:~$ hhfab vlab up
15:27:13 INF Hedgehog Fabricator version=v0.32.1
[ ... ]
15:27:23 INF Downloading name=fabricator/hhfabctl version=v0.32.1 type=oras
[ ... ]
Downloading images                          465.87 KiB / 465.87 KiB    ⁘   done
Downloading boot                              2.06 MiB / 2.06 MiB      ⁘   done
Downloading EFI                               2.02 MiB / 2.02 MiB      ⁘   done
Downloading flatcar_production_image.bin.bz2  493.28 MiB / 493.28 MiB   ⁘   done
[ ... ]
15:30:13 INF Preparing new vm=control-1 type=control
15:30:28 INF Preparing new vm=server-01 type=server
15:30:29 INF Preparing new vm=server-02 type=server
15:30:31 INF Preparing new vm=server-03 type=server
15:30:33 INF Preparing new vm=server-04 type=server
15:30:34 INF Preparing new vm=leaf-01 type=switch
15:30:36 INF Preparing new vm=leaf-02 type=switch
15:30:36 INF Starting VMs count=7 cpu="22 vCPUs" ram="19456 MB" disk="240 GB"
[ ... ]
15:42:04 INF Control node is ready vm=control-1 type=control
15:42:04 INF All VMs are ready
[ keeps running in foreground ]
```

Download components, build, boot, run fabric

```
ubuntu@vlab-quentin:~$ hhfab vlab ssh
[ ... ]
core@control-1 ~ $
```

Connect to control node

```
core@control-1 ~ $ kubectl get switch
NAME      PROFILE   ROLE          DESCR          GROUPS        LOCATIONUUID   AGE
leaf-01   vs        server-leaf   VS-01 MCLAG 1  ["mclag-1"]                  10m
leaf-02   vs        server-leaf   VS-02 MCLAG 1  ["mclag-1"]                  10m

core@control-1 ~ $ kubectl fabric inspect switch -n leaf-01

NAME PROFILE        ROLE          GROUPS  SERIAL  STATE   GEN  APPLIED HEARTBEAT
leaf-01 Virtual Switch  server-leaf   mclag-1       Pending 0/1
```

List info on a switch

```
Ports (in use):

NAME NOS            TYPE          CONNECTION                                ADM/OP (TRANSC) SPEED
TRANSCEIVER
E1/1 Ethernet0      mclag-domain  leaf-01--mclag-domain--leaf-02
E1/2 Ethernet1      mclag-domain  leaf-01--mclag-domain--leaf-02
E1/3 Ethernet2      mclag-domain  leaf-01--mclag-domain--leaf-02
E1/4 Ethernet3      mclag-domain  leaf-01--mclag-domain--leaf-02
E1/5 Ethernet4      mclag         server-01--mclag--leaf-01--leaf-02
E1/6 Ethernet5      mclag         server-02--mclag--leaf-01--leaf-02
E1/7 Ethernet6      unbundled     server-03--unbundled--leaf-01
[ ... ]

Port Counters (↓ In ↑ Out):

NAME SPEED   UTIL %  BITS/SEC IN  BITS/SEC OUT  PKTS/SEC IN   PKTS/SEC OUT  ERRORS  DISCARDS
[ ... ]
```

```
core@control-1 ~ $ kubectl fabric vpc create --name vpc-1 --subnet 10.0.1.0/24 \
                    --vlan 1001 --dhcp --dhcp-start 10.0.1.10
16:52:10 INF VPC created name=vpc-1

core@control-1 ~ $ kubectl fabric vpc create --name vpc-2 --subnet 10.0.2.0/24 \
                    --vlan 1002 --dhcp --dhcp-start 10.0.2.10
16:52:19 INF VPC created name=vpc-2
```

Create new VPCs

```
core@control-1 ~ $ kubectl fabric vpc attach --vpc-subnet vpc-1/default \
                    --connection server-01--mclag--leaf-01--leaf-02
16:52:37 INF VPCAttachment created name=vpc-1--default--server-01--mclag--leaf-01--leaf-02

core@control-1 ~ $ kubectl fabric vpc attach --vpc-subnet vpc-2/default \
                    --connection server-02--mclag--leaf-01--leaf-02
16:52:52 INF VPCAttachment created name=vpc-2--default--server-02--mclag--leaf-01--leaf-02
```

Attach new VPCs to existing connections

```
core@server-01 ~ $ hhnet cleanup
core@server-01 ~ $ hhnet bond 1001 enp2s1 enp2s2
10.0.1.10/24
```

Setup bond interface on servers

```
core@server-02 ~ $ hhnet cleanup
core@server-02 ~ $ hhnet bond 1002 enp2s1 enp2s2
10.0.2.10/24
```

```
core@server-01 ~ $ ping 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Net Unreachable
From 10.0.1.1 icmp_seq=2 Destination Net Unreachable
From 10.0.1.1 icmp_seq=3 Destination Net Unreachable
^C
--- 10.0.2.10 ping statistics —
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2003ms
```

No connectivity

```
core@control-1 ~ $ kubectl fabric vpc peer --vpc vpc-1 --vpc vpc-2
16:58:04 INF VPCPeering created name=vpc-1--vpc-2
```

Setup peering between VPCs

```
core@server-01 ~ $ ping 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=1 ttl=62 time=6.25 ms
64 bytes from 10.0.2.10: icmp_seq=2 ttl=62 time=7.60 ms
64 bytes from 10.0.2.10: icmp_seq=3 ttl=62 time=8.60 ms
^C
--- 10.0.2.10 ping statistics —
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 6.245/7.481/8.601/0.965 ms
```

VPCs connected!

# Thank you!

## Open Network Fabric

`github.com/githedgehog`

Contributions welcome!


`https://hedgehog.cloud`