Homelab

Release 1

Jan Jansen

May 31, 2025

CONTENTS:

CHAPTER

ONE

INTRO

1.1 Homelab

A homelab is an environment where I can try things out, without upsetting a CISO, CDO, CITO . . . . The system is an old HP proliant gen8, with 2TB of space and 256GB of memory, and bi-XEON with 16 cores in total.

1.2 Virtualisation

I choose proxmox as a virtualisation platform, it runs linux containers (LXC), which are kind of cool since they are created quickly, launched quickly. Some problems arise since there are still shared resources with the host. . .

Proxmox supports virtual machines (VM) as well.

VM are used as kubernetes nodes, since using LXC was problematic. This setup allows to run a complete kubernetes multiple node cluster on a single machine.

1.3 Containerisation

Containerization, in simple terms, is like packing a lunchbox for software. It's a way to bundle an application with everything it needs to run—code, libraries, settings, and dependencies—into a single, portable package called a container. This makes it easy to move and run the software consistently on any computer, server, or cloud, without worrying about differences in the environment.

1.4 Kubernetes

- Organizes Containers: It decides which computers (servers) should run each container, spreading them out to avoid overloading any one server. It's like assigning chefs to different kitchen stations to keep things efficient.
- Scales Automatically: If your app gets super popular (like a sudden rush of customers), Kubernetes can add more containers to handle the demand. If things slow down, it removes extras to save resources.
- Keeps Things Running: If a container crashes (like a lunchbox spilling), Kubernetes quickly replaces it with a new one so your app stays available.
- Manages Traffic: It directs users to the right containers, ensuring everyone gets access to the app without delays, like guiding customers to the right food station.

Updates Smoothly

CHAPTER

TWO

KUBERNETES ARCHITECTURE METALLB, TRAEFIK

To make `www.example.com` accessible from my laptop.

2.1 Overview of Setup

running a Kubernetes cluster on Talos Linux nodes, hosted on a Proxmox virtualization platform. The setup includes:

- **MetalLB** for load balancing
- **Traefik** as an ingress controller
- An **Ingress** resource to route traffic to a service for `www.example.com`
- **laptop** resolves `www.example.com` to `10.10.10.10` via `/etc/hosts`
- A **Proxmox LXC container** acts as a router/DHCP server (`192.168.1.200`) on a network bridge (`vmbr1`)
- a router assigns IPs to three Talos nodes and one Talos client

Below, each part is explained in detail.

2.2 1. Proxmox and Networking Setup

Proxmox is the virtualization platform hosting your infrastructure, including the Talos VMs and the router container.

Proxmox Network Bridge (vmbr1)

- Acts as a virtual switch that connects VMs and containers.
- Configured to handle networking for Talos nodes and the client.
- Devices on this bridge typically reside in the `192.168.1.0/24` subnet.

LXC Container (Router/DHCP Server at 192.168.1.200)

- Provides dynamic IP assignment via DHCP.
- Routes traffic between the `vmbr1` subnet and external networks.
- May also offer NAT, DNS, or firewall services.

Network Flow

- Talos nodes and the client receive IPs (e.g., `192.168.1.201-204`) from this container.
- The container enables traffic to flow between your laptop's network and the Kubernetes cluster.

2.3 2. Talos Linux Kubernetes Cluster

Three Talos Cluster Nodes

- VMs on Proxmox configured as Kubernetes nodes.
- a control plane 2 worker nodes
- IPs assigned by DHCP from the LXC router.

One Talos Client

- a LXC device used for managing the cluster using `talosctl`.
- Communicates with the control plane via the Talos API.

Talos Networking

- Nodes are assigned IPs via `vmbr1`.
- Kubernetes networking handles intra-cluster communication.

2.4 3. Kubernetes Networking with MetalLB and Traefik

MetalLB

- Provides load balancing by assigning external IPs (e.g., `10.10.10.10`) to LoadBalancer services.
- Operates in Layer 2 (ARP) or BGP mode.
- A node "owns" the external IP and routes traffic to the appropriate pod.

How MetalLB Works

- Configured with a pool like `10.10.10.0/24`.
- Ensures external traffic is routed to the correct service or pod via ARP.

Traefik as Ingress Controller

- A reverse proxy and ingress controller running in the cluster.
- Exposed via a LoadBalancer service assigned the IP `10.10.10.10`.
- Handles HTTP/S requests and routes them according to Ingress rules.

Ingress Resource Example

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    traefik.ingress.kubernetes.io/router.entrypoints: web, websecure
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
```

(continues on next page)

(continued from previous page)

```
        backend:
          service:
            name: example-service
            port:
              number: 80
```

This configuration tells Traefik to forward traffic for `www.example.com` to the `example-service` on port `80`.

2.5 4. Laptop Configuration

/etc/hosts Entry

Your laptop uses the following line in `/etc/hosts`:

```
10.10.10.10 www.example.com
```

This resolves `www.example.com` locally without DNS.

Routing to the Cluster

- The LXC router at `192.168.1.200` forwards traffic to `10.10.10.10`.
- Routing may involve NAT or static rules to ensure reachability.

2.6 5. How It All Ties Together

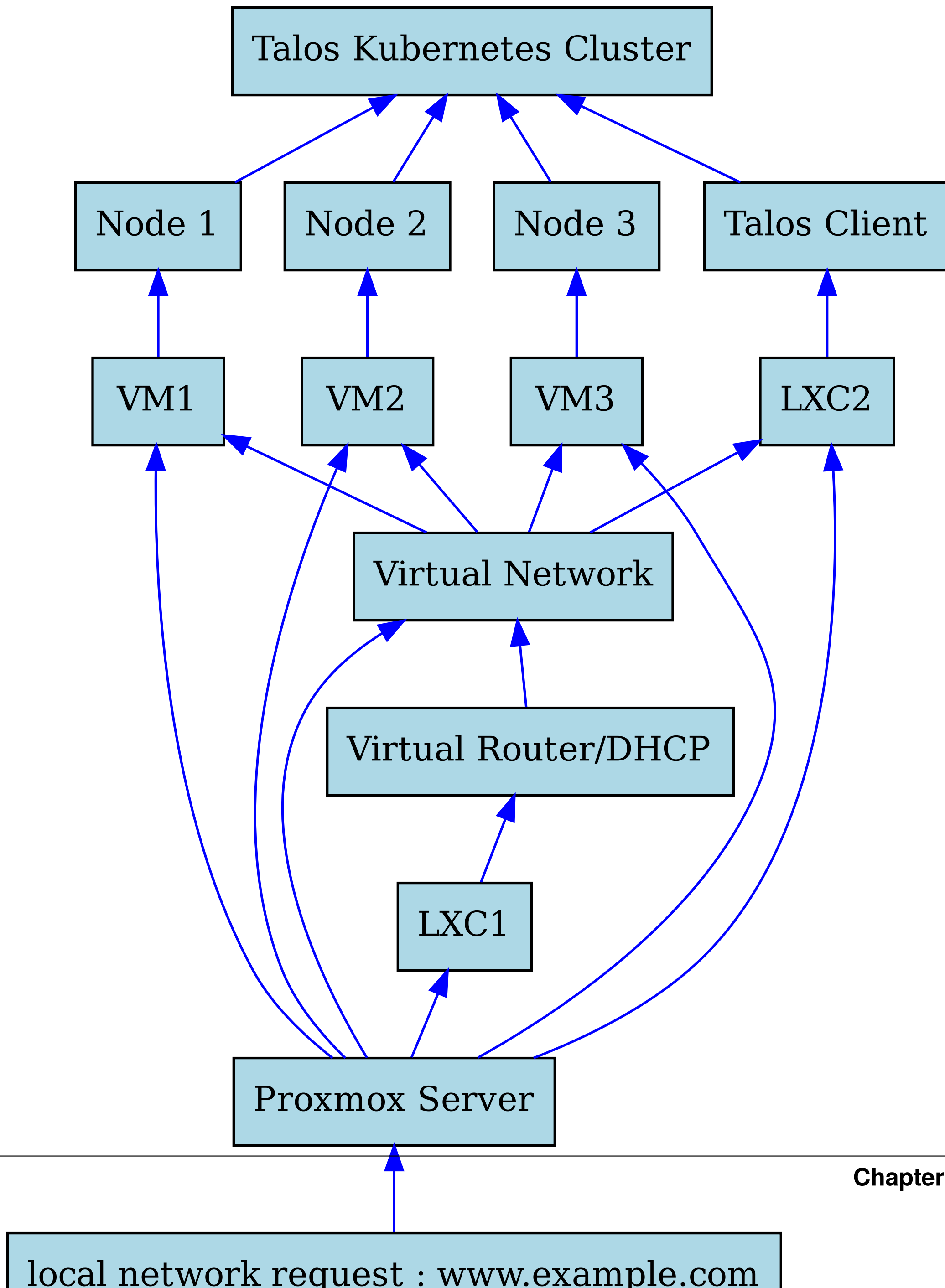
The full path of a request to `www.example.com` is:

1. **Laptop Resolution** - The browser looks up `www.example.com` and resolves it to `10.10.10.10` using `/etc/hosts`.
2. **Network Routing** - Traffic is sent to `10.10.10.10` and routed by the LXC container to the Talos node that owns this IP.
3. **MetalLB** - Owns the `10.10.10.10` IP and ensures the request reaches the correct node.
4. **Traefik** - Receives the HTTP/S request on port 80/443. - Uses the Ingress resource to determine routing.
5. **Kubernetes Service and Pods** - `example-service` forwards the request to one of your pods (e.g., Nginx). - The response is returned back through Traefik and MetalLB.
6. **Response** - Your browser displays the resulting web page from `www.example.com`.

CHAPTER

THREE

TALOS



Talos is a modern OS for Kubernetes. It is designed to be secure, immutable, and minimal. Talos is a self-hosted Kubernetes distribution that runs on bare metal or virtualized infrastructure. Talos is designed to be managed by a central Kubernetes control plane, which can be hosted on the same cluster or on a separate cluster.

talosctl config add my-cluster –endpoints 192.168.0.242

talosctl config info

talosctl config endpoint 192.168.0.242

talosctl gen config my-cluster https://192.168.0.242:6443 –output-dir ./talos-config –force

3.1 new install talos

https://www.talos.dev/v1.9/talos-guides/install/virtualized-platforms/proxmox/

> talosctl gen config my-cluster https://192.168.0.218:6443 talosctl -n 192.168.0.169 get disks –insecure (check disks) talosctl config endpoint 192.168.0.218 talosctl config node 192.168.0.218
>
> talosctl apply-config –insecure –nodes 192.168.0.218 –file controlplane.yaml
>
> talosctl bootstrap talosctl kubeconfig . (retrieve kubeconfig) talosctl –nodes 192.168.0.218 version (verify)
>
> export KUBECONFIG=./talos-config/kubeconfig
>
> > kubectl get nodes kubectl get pods -n kube-system kubectl get pods -n kube-system -o wide

kubectl describe pod my-postgres-postgresql-0 (is very useful in case the pod does get deployed

https://factory.talos.dev/ (create your custom image)

```
talosctl upgrade --nodes 10.10.10.178 --image  factory.talos.dev/installer/
↪c9078f9419961640c712a8bf2bb9174933dfcf1da383fd8ea2b7dc21493f8bac:v1.9.5
```

watching nodes: [10.10.10.178]
talosctl get extensions –nodes 10.10.10.178

NODE NAMESPACE TYPE ID VERSION NAME VERSION 10.10.10.178 runtime ExtensionStatus 0 1 iscsi-tools v0.1.6 10.10.10.178 runtime ExtensionStatus 1 1 schematic c9078f9419961640c712a8bf2bb9174933dfcf1da383fd8ea2b7dc21493f8bac

3.2 adding worker nodes

Since "longhorn" stores data on more than one node, we need to add more nodes to the cluster.

> talosctl apply-config –insecure –nodes 10.10.10.166 –file worker.yaml talosctl apply-config –insecure –nodes 10.10.10.173 –file worker.yaml

kubectl get nodes -o wide NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME talos-2ho-roe Ready <none> 113s v1.32.3 10.10.10.173 <none> Talos (v1.9.5) 6.12.18-talos containerd://2.0.3 talos-swn-isw Ready control-plane 31d v1.32.3 10.10.10.118 <none> Talos (v1.9.5) 6.12.18-talos containerd://2.0.3 talos-v1x-9s4 Ready <none> 2m18s v1.32.3 10.10.10.166 <none> Talos (v1.9.5) 6.12.18-talos containerd://2.0.3 talos-y7t-8ll Ready worker 29d v1.32.3 10.10.10.178 <none> Talos (v1.9.5) 6.12.18-talos containerd://2.0.3

3.3 label nodes

kubectl label nodes talos-v1x-9s4 node-role.kubernetes.io/worker=”” kubectl label nodes talos-2ho-roe node-role.kubernetes.io/worker=””

CHAPTER

FOUR

PROXMOX INSTALLATION NOTES

remove firewall from talos worker node

```
root@pve:~# qm set 109 -net0 virtio,bridge=vmbr0,firewall=0
update VM 109: -net0 virtio,bridge=vmbr0,firewall=0
```

4.1 setting up an extra network bridge vmbr1

on lxc dedicated machine setup dhcp and routing

```
apt install dnsmasq -y
apt install iptables-persistent -y

vi /etc/dnsmasq.conf
interface=eth0
dhcp-range=10.10.10.100,10.10.10.200,12h  # DHCP range for Talos nodes
dhcp-option=3,10.10.10.2                  # Gateway (this machine's eth0 IP)
dhcp-option=6,192.168.0.1                 # DNS (your home router's DNS)

systemctl restart dnsmasq

echo 1 > /proc/sys/net/ipv4/ip_forward

iptables -t nat -L -v
ip route del default via 10.10.10.1 dev eth0
ip route replace default via 192.168.0.1 dev eth1 metric 100

iptables -t nat -A POSTROUTING -s 10.10.10.0/24 -o eth1 -j MASQUERADE
ip route del default via 10.10.10.1 dev eth0
ip route replace default via 192.168.0.1 dev eth1 metric 100



vi /etc/netplan/01-netcfg.yaml
```

```
network:
  version: 2
  ethernets:
    eth0:
```

(continues on next page)

(continued from previous page)

```
      dhcp4: true
      # Prevent DHCP from setting a default gateway if it conflicts
      dhcp4-overrides:
        use-routes: true
        use-dns: true
        route-metric: 2000  # High metric to prioritize eth1's default route
    eth1:
      dhcp4: false
      addresses:
        - 192.168.0.x/24  # Replace with your server's IP on this subnet
      routes:
        - to: 0.0.0.0/0
          via: 192.168.0.1
          metric: 100
        - to: 0.0.0.0/0
          via: 192.168.0.1
          metric: 1024
        - to: 192.168.0.0/24
          via: 0.0.0.0
          metric: 1024
        - to: 192.168.0.1
          via: 0.0.0.0
          metric: 1024
        - to: <gent.dnscache01-ip>
          via: 192.168.0.1
          metric: 1024
        - to: <gent.dnscache02-ip>
          via: 192.168.0.1
          metric: 1024
```

```
# Apply the netplan configuration
sudo netplan generate
sudo netplan apply

# Check the routing table
ip route show

# Check iptables rules
iptables -t nat -L -v

# Check dnsmasq status
systemctl status dnsmasq

# Check if the DHCP server is running and listening on the correct interface
sudo systemctl status dnsmasq

# Restart dnsmasq to apply changes
sudo systemctl restart dnsmasq


netplan apply
```

4.2 Kernel IP routing table

4.3 using the nodeport

192.168.0.251:30743

on my router/dhcp on 10.10.10.2 route port to cluster node IP

iptables -t nat -A PREROUTING -i eth1 -p tcp –dport 30743 -j DNAT –to-destination 10.10.10.118:30743

so running nginx on kubernetes on 10.10.10.255 network is accessible from the outside

4.4 using the IP address

traefik LoadBalancer 10.102.122.212 10.10.10.50 80:32178/TCP,443:32318/TCP 75m app.kubernetes.io/instance=traefik-default,app.kubernetes.io/name=traefik

So now I have to figure out how I can reach 10.10.10.50 from my 192.168.X.X network

on the kubernetes cluster, traefik has been deployed as well as metallb. iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -d 10.10.10.0/24 -j MASQUERADE sh -c "iptables-save > /etc/iptables/rules.v4"

this has been added to th dnsmasq.conf

Listen on the 192.168.0.251 interface interface=eth1 # Replace with your 192.168.0.251 interface (check with *ip a*) listen-address=192.168.0.251

Forward other queries to upstream DNS (e.g., Google DNS) server=8.8.8.8 server=8.8.4.4

Optional: If LXC is your DHCP server, ensure DNS is offered dhcp-option=6,192.168.0.251 # Tells DHCP clients to use this as DNS

4.5 modify dns config on laptop

/etc/resolv.conf

add : nameserver 192.168.0.251

4.6 access http://nginx.example.com/ on talos within 10.10.10.X from 192.168.X.X

(configure metallb, traefik, nginx)

on laptop /etc/hosts : 10.10.10.50 nginx.example.com

on dhcp server (10.10.10.2)

iptables -A FORWARD -s 192.168.0.0/24 -d 10.10.10.0/24 -j ACCEPT iptables -A FORWARD -s 10.10.10.0/24 -d 192.168.0.0/24 -j ACCEPT

```
# Generated by iptables-save v1.8.7 on Thu Apr 10 13:32:59 2025
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A FORWARD -s 192.168.0.0/24 -d 10.10.10.0/24 -j ACCEPT
-A FORWARD -s 10.10.10.0/24 -d 192.168.0.0/24 -j ACCEPT
COMMIT
# Completed on Thu Apr 10 13:32:59 2025
# Generated by iptables-save v1.8.7 on Thu Apr 10 13:32:59 2025
*nat
:PREROUTING ACCEPT [6847:1975161]
:INPUT ACCEPT [158:15156]
:OUTPUT ACCEPT [25:2590]
:POSTROUTING ACCEPT [25:2590]
-A PREROUTING -i eth1 -p tcp -m tcp --dport 30743 -j DNAT --to-destination 10.10.10.
↪118:30743
-A POSTROUTING -s 10.10.10.0/24 -o eth1 -j MASQUERADE
-A POSTROUTING -s 10.10.10.0/24 -o eth1 -j MASQUERADE
-A POSTROUTING -s 10.10.10.0/24 -o eth1 -j MASQUERADE
-A POSTROUTING -s 192.168.0.0/24 -d 10.10.10.0/24 -j MASQUERADE
COMMIT
# Completed on Thu Apr 10 13:32:59 2025
# Check the iptables rules
iptables -t nat -L -v
iptables -L -v

# Check the routing table
ip route show

# Check the network interfaces
ip a
```

CHAPTER

FIVE

KERNEL IP ROUTING TABLE

The following table represents the kernel IP routing table:

Table 1: Kernel IP Routing Table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
default	192.168.0.1	0.0.0.0	UG	100	0	0	eth1
default	192.168.0.1	0.0.0.0	UG	1024	0	0	eth1
10.10.10.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
192.168.0.0	0.0.0.0	255.255.255.0	U	1024	0	0	eth1
192.168.0.1	0.0.0.0	255.255.255.255	UH	1024	0	0	eth1
gent.dnscache01	192.168.0.1	255.255.255.255	UGH	1024	0	0	eth1
gent.dnscache02	192.168.0.1	255.255.255.255	UGH	1024	0	0	eth1

CHAPTER

SIX

LAPTOP (OR PC) MODS

/etc/hosts

10.10.10.50 nginx.example.com 10.10.10.50 registry.example.com

ip route add 10.10.10.0/24 via 192.168.0.251

sudo cp ca.crt /usr/local/share/ca-certificates/ca.crt sudo update-ca-certificates Updating certificates in /etc/ssl/certs…

6.1 make route permanent

```
sudo nano /etc/netplan/01-netcfg.yaml

network:
version: 2
ethernets:
  wlp0s20f3:
    addresses:
      - 192.168.0.103/24
    gateway4: 192.168.0.1  # Replace with your actual default gateway
    routes:
      - to: 10.10.10.0/24
        via: 192.168.0.251

  sudo netplan apply
```

CHAPTER

SEVEN

SDA (SOFTWARE DEFINED ARCHITECTURE)

7.1 Physical Infrastructure

7.2 Proxmox Server Specifications

The foundation of the architecture is a physical server running Proxmox VE hypervisor.

Component	Description
Hypervisor	Proxmox Virtual Environment (VE)
Virtual Machines	Multiple Talos OS nodes forming a Kubernetes cluster
Containers	LXC container serving as a router

7.3 Virtual Machine Configuration

Talos OS Nodes

The cluster consists of multiple Talos OS nodes, with dedicated roles:

- **Control Plane Node(s)**: Manages the Kubernetes control plane
- **Worker Nodes**: Runs application workloads

```
# Example Talos configuration structure (simplified)
machine:
  type: controlplane  # or worker
  network:
    hostname: talos-node-1
  kubernetes:
    version: v1.26.0
```

7.4 Networking Architecture

7.4.1 Network Components

Component	Function
Proxmox Virtual Bridge	Creates isolated network segments for VMs and containers
LXC Router	Routes traffic between internal and external networks
Kubernetes Overlay Network	Enables pod-to-pod communication (Cilium, Flannel, etc.)

7.5 Control & Automation

7.5.1 API Management Layer

This architecture leverages multiple declarative APIs for infrastructure management:

API	Responsibility
Proxmox API	Manages physical resources, VMs, and containers
Talos API	Provides declarative OS configuration and maintenance
Kubernetes API	Orchestrates applications and services

7.6 Benefits of This Architecture

- **Immutable Infrastructure**: Talos OS provides an immutable, declarative operating system
- **High Availability**: Kubernetes manages service availability and distribution
- **Resource Efficiency**: Consolidates multiple services on a single physical server
- **Isolation**: Separate network segments and container boundaries
- **Automation**: API-driven management at all levels

CHAPTER EIGHT

INDICES AND TABLES

- genindex
- modindex
- search