Containers on RouterOS

How to run containers on RouterOS and when does it actually make sense?

Who am I?

Tomas Kirnak

Team Lead @ Unimus

System & Network Architect
Automation & Monitoring

Ex-MikroTik Trainer, Consultant

NETCOREJSA

Unimus

About Unimus

Unimus is a multi-vendor system for:

- Network Disaster recovery
- Change Management
- Network Automation
- Configuration Management
- Network Auditing & Compliance

Come and see us at our booth!

Note for posterity

If you find this presentation online in a .pdf, please watch the video

Proper explanations to every slide and much more information available

https://www.youtube.com/c/TomasKirnak/videos

Container basics

Isn’t this Docker?

- We will be talking about containers in this presentation – Linux Containers are actually not a single thing
- There is Docker, LXC, Podman and more – these are actually not the same things
- So what actually are Linux Containers?

Define “container”…

- A Linux container is a set of 1 or more processes that are isolated from the rest of the system
- Think of it kinda like a VM, except you aren’t running a whole OS, just a single application isolated in a container
- Simply, it’s an entity isolated from the main host, but sharing the main host’s hardware

Can you be more technical?

- Sure. Here is what makes containers possible:

- Namespaces (PID, Network, Mount, IPC, User)
  - Isolate processes within the container from the host and other containers
- Control Groups (cgroups)
  - limit and allocate system resources (CPU, memory, I/O, network) to containers
- Union Filesystem
  - Allows containers to use an isolated overlay filesystem
- Container Runtime
  - responsible for running and managing containers

So what about RouterOS?

- RouterOS provides an OCI container compliant implementation
- This just means you can run the same containers as Docker, LXC, Podman, etc.

- We don’t actually know what RouterOS is running under the hood to provide container support, altho LXC is suspected

Why are containers useful?

So why should I care?

- It’s just another useful tool in the tool belt.
- Very useful in specific cases, not useful in others.

- Allows you to reduce hardware, centralizing services (fewer failure points, less power draw, less to manage)
- You have to deploy a router anyway, so why not use it…

So when is this useful?

- We will go over multiple use-cases and examples in this presentation.
- Hopefully by the end, you will see why and when containers on RouterOS can be an extremely powerful tool.

What hardware is supported?

Container support

- arm64 / Ampere ✔
  - CCR 2216, 2004, RB5009, hAP ax, cAP ax
- CHR (x64) ✔
- arm (careful with resources)
  - RB1100AHx4, L009, RB4011, hAP ac, cAP ac
- x86 (technically yes, realistically no)

Aren’t we forgetting something…

- We also have this monster now… RDS2216

- arm64
- 16 CPU cores
- 32 GB RAM

So what do I actually need?

- RouterOS v7
  - the newer the better
- RAM
  - 50 MB - 200 MB per container, depending on service
- Some CPU power
  - Very much depends on what the container does
- Storage
  - Mandatory, but any USB stick will do

Can you give me an example

- RB5009
- 4 CPU cores, 1 GB RAM

- Will run 2-4 containers
- Will max CPU at ~2 Gbps of traffic using iperf in a container

So how do I deploy a container?

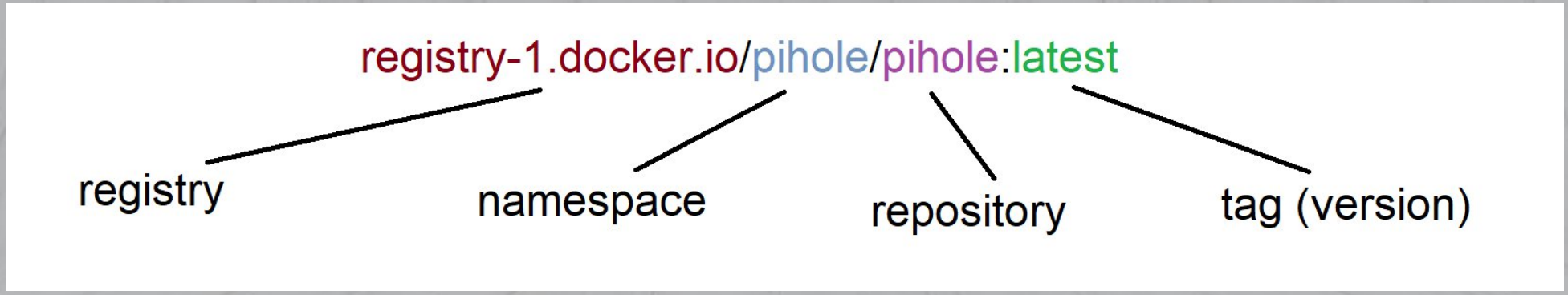
Some container basics…

- Container images need to be pulled from a registry, or an image needs to be provided manually
- Containers are immutable and reverted to the “image” state after each restart
  - Run a pure clean Ubuntu container, “apt install curl”, restart container … and it’s gone.
  - We need to deal with persisting data differently…

Pulling images

- There are many registries (Docker Hub, GitHub, etc.)
- To pull an image, you need an image reference:

registry-1.docker.io/pihole/pihole:latest



What are valid references

- Let’s talk about how each of these behave:

- ubuntu
- ubuntu:22.04
- pihole/pihole
- croc/unimus-core:2.6.2
- ghcr.io/goauthentik/radius:latest

Enough theory, can we do something practical instead?

Hello World!

- We will deploy a network engineer's equivalent of "Hello World!"
- How about a speedtest container!

- Let's start with some preparation…

Prepare our router

- Before deploying container we need to do a one-time setup…

- Deploy container package
- Enable container capability
- Prepare our USB stick storage

Let’s get started…

- Let’s start with a (relatively) empty RB5009

```
/system package update check-for-updates
/system package enable container
/system package apply-changes

/system device-mode update container=yes
```

Let’s prepare storage…

```
/disk
format-drive usb1 file-system=ext4

/file
add name=/usb1/container-pull type=directory
add name=/usb1/containers type=directory
```

Setup container defaults…

- Finally let’s set our default image registry, and let’s setup an image pull directory

```
/container config
set registry-url=https://registry-1.docker.io
tmpdir=/usb1/container-pull
```

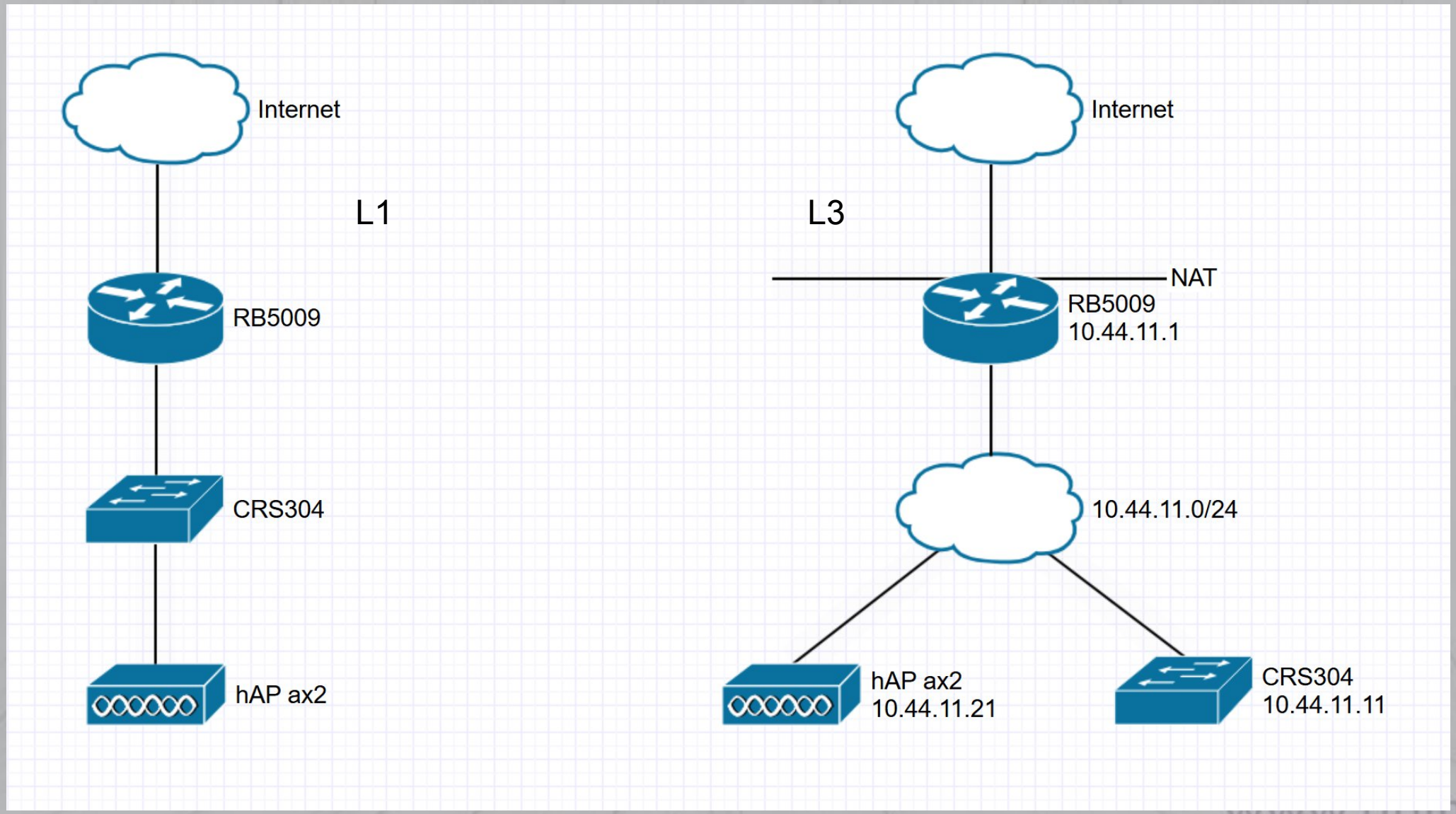
Finally, Hello World!

```
/interface veth
add address=10.0.255.1/31 gateway=10.0.255.0 name=veth1

/ip address
add address=10.0.255.0/31 interface=veth1

/container
add root-dir=/usb1/containers/openspeedtest remote-
image=openspeedtest/latest:latest interface=veth1
```

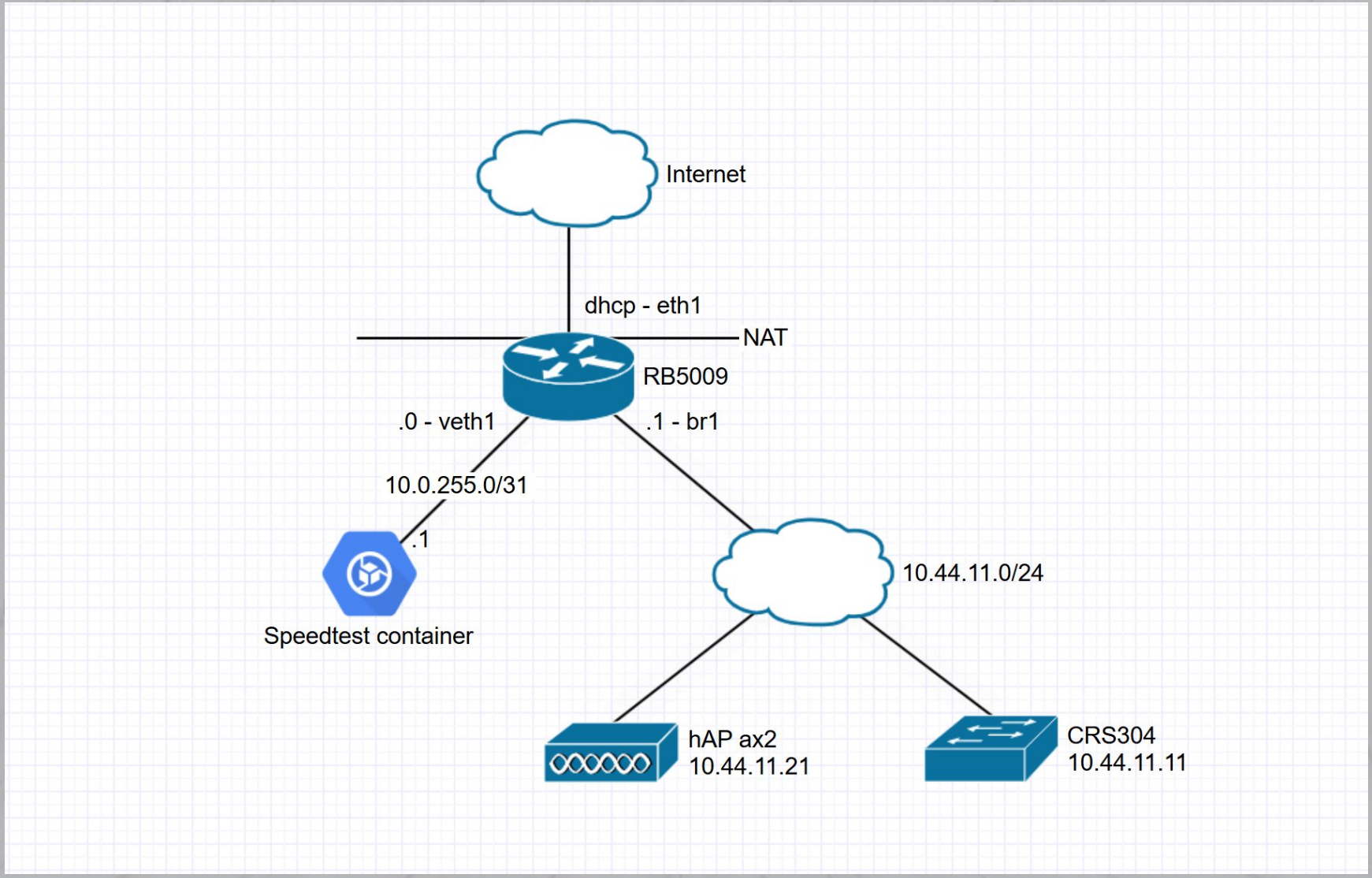
Wait… what was that?

- What was all that networking stuff?
- Well, let’s talk about how container networking on RouterOS works…

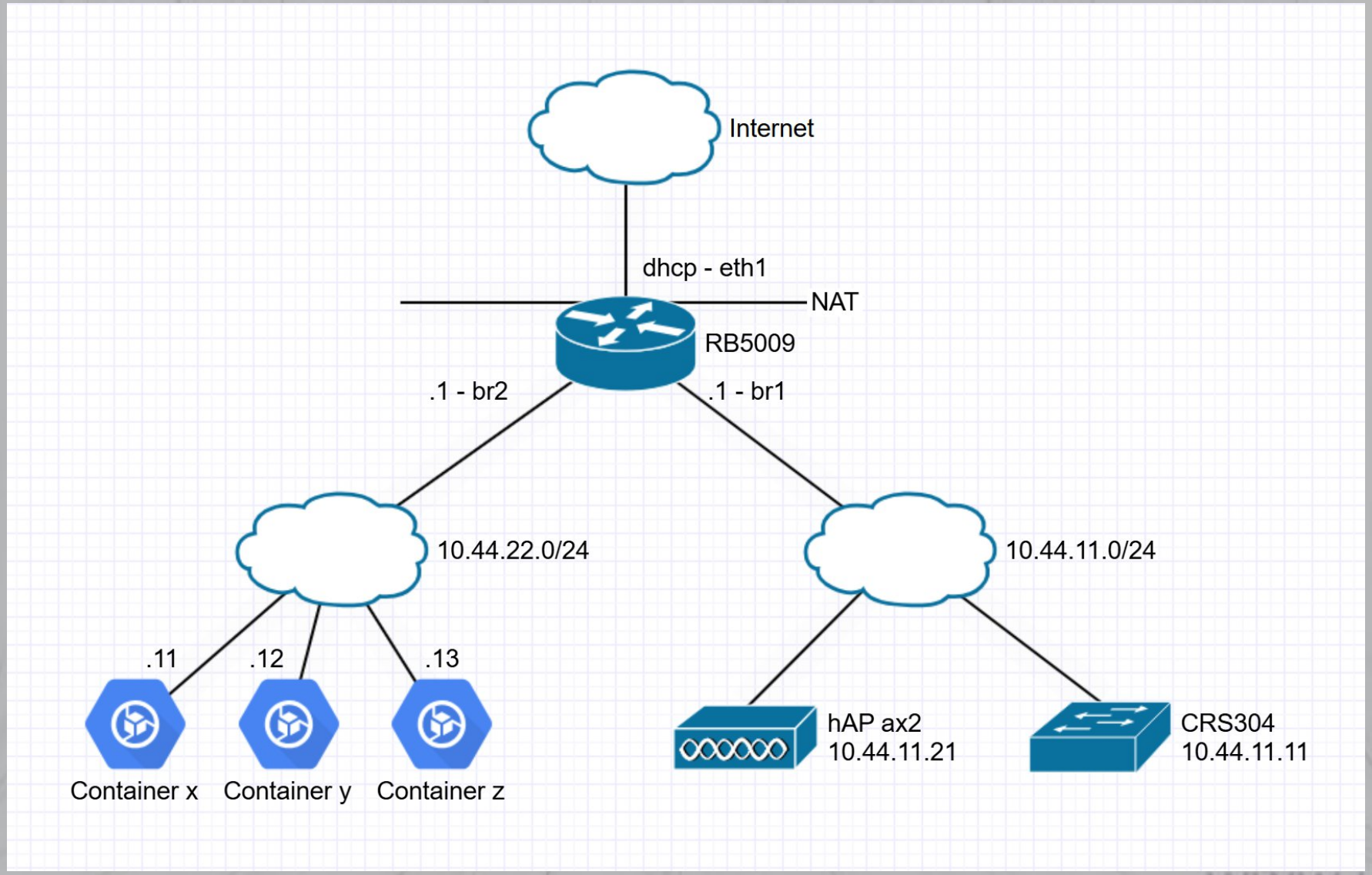
Lab topology...



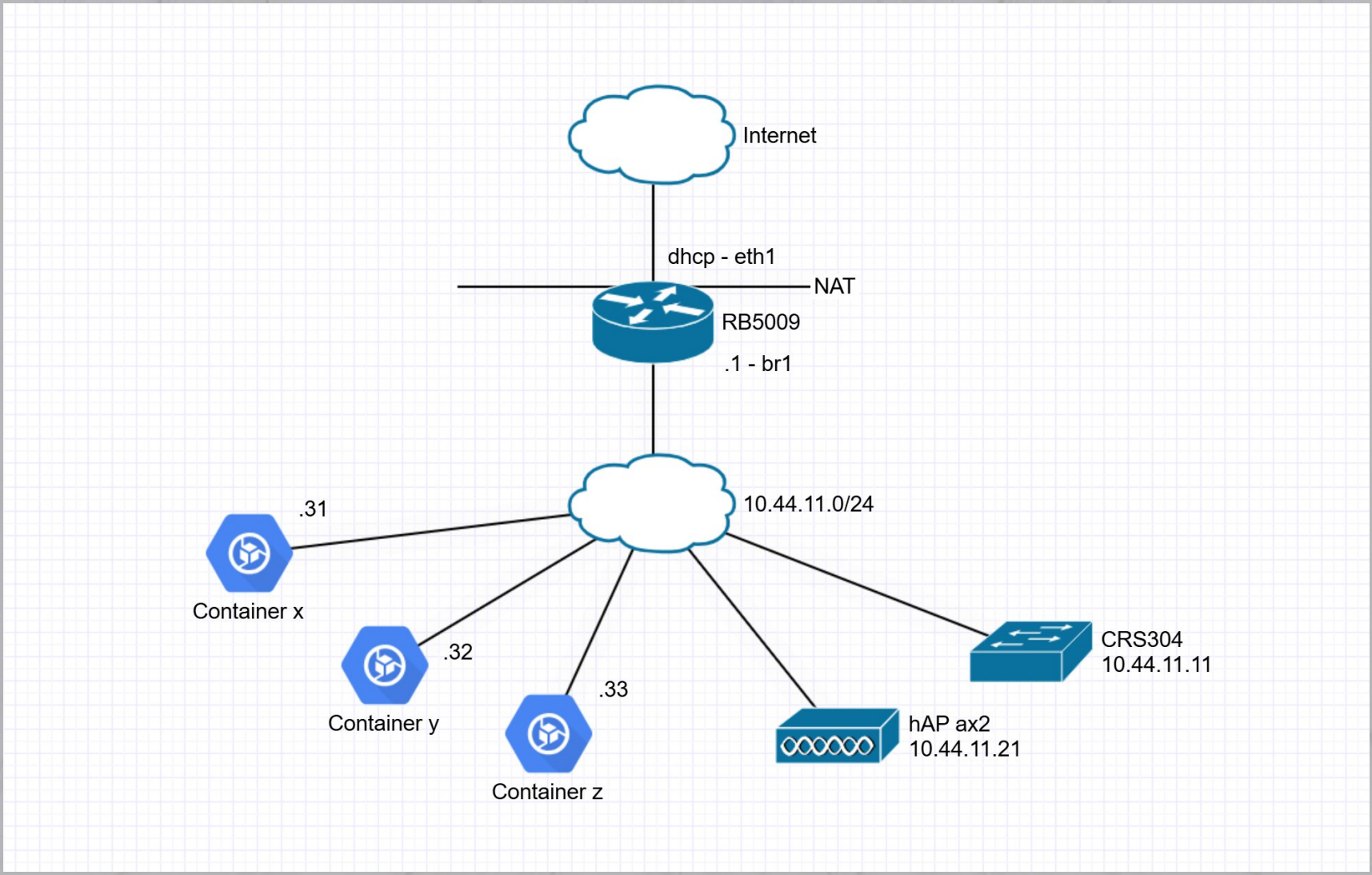
Container connectivity option 1



Container connectivity option 2



Container connectivity option 3



Pros and cons

- Let’s discuss pros and cons of each network setup:

- Each container in their own subnet (or /31)
- One subnet for all containers
- Containers in the main “LAN” subnet

Setup for this presentation

- For the rest of this presentation we will just put all the containers into the main LAN (10.44.11.0/24)
- This is for convenience and time-saving only

- Please consider what is appropriate for production use

So, valid usecase no.1

- Speedtesting and perf. monitoring
- Fast speedtest web interface in 3 commands that works from Windows, Linux, Mac (as opposed to btest)
- Large-scale speedtest deployments...

Scenario – nightly perf. tests

- Previous project – a nation-wide hotel chain wishes to test the internet connection of each hotel nightly
- We deployed a RPi at each hotel (hundreds of RPi that need to be maintained, secured, replaced when broken)
- Why not iperf3 against the router itself…

Let’s get iperf3 on the router

```
/interface veth
add address=10.44.11.31/24 gateway=10.44.11.1
name=veth2

/interface bridge port
add bridge=br1 interface=veth2

/container
add root-dir=/usb1/containers/net-tools remote-
image=iitgdocker/iperf-web:latest interface=veth2
```

Next up – local pollers

- When you manage many disparate networks (in particular MSPs), you need a local poller in each network
- The poller just polls data from local devices (due to NAT, firewalls, etc), server is elsewhere
- Basically, you don’t need to VPN all the networks together just for management purposes

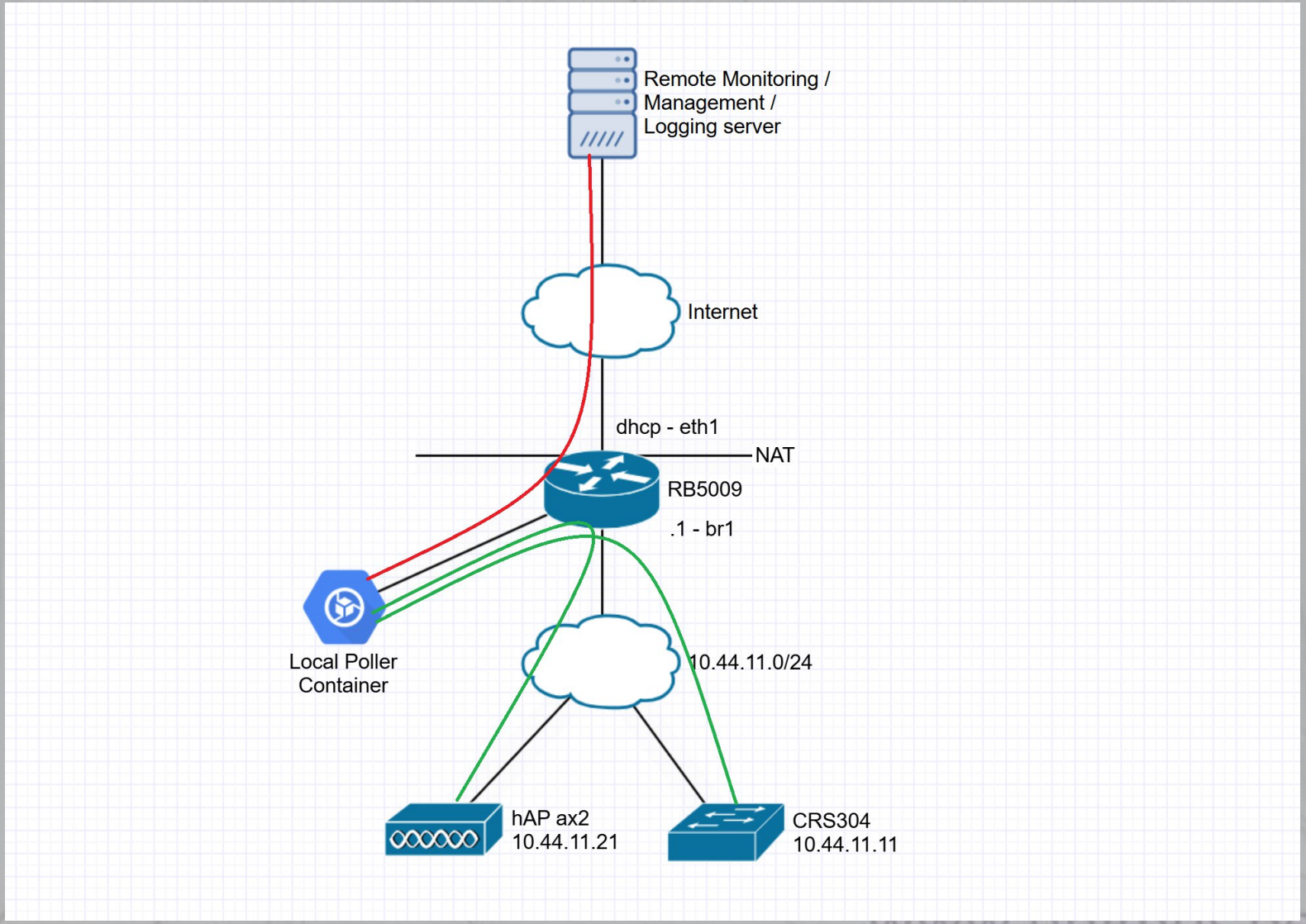
What will we deploy?

- This is useful for:
  - Monitoring / NMS - NetXMS, Zabbix, etc.
  - Config Management / NCM - Unimus
  - Log collection - rsyslog
  - Auth Proxy – Radius, Authentik
  - Etc.

How does this work?

Each system calls this a bit differently:

– Poller
– Proxy
– Remote Agent
– Remote Core
– Outpost



Let’s start with Unimus

- We now need to provide some settings to our container
  - Address of Unimus server
  - Access key
- For this, we can use Environment Variables
  - These will be passed to the container, and the containerized service can use them
  - Each service has their own variables

Let’s start with Unimus – ptl

```
/interface veth
add address=10.44.11.32/24 gateway=10.44.11.1
name=veth3

/interface bridge port
add bridge=br1 interface=veth3
```

Let’s start with Unimus – pt2

```
/container envs

add name=unimus-core key=TZ value=Europe/Bratislava

add name=unimus-core key=UNIMUS_SERVER_ADDRESS
value=11.123.234.55

add name=unimus-core key=UNIMUS_SERVER_PORT value=5509


add name=unimus-core key=UNIMUS_SERVER_ACCESS_KEY
value="M^U<#HnS\$AD\?uLO8\?PLi%H@6qAdXp\$kQ--
GTfmQ/Hdcs4fudMaSA6lmQ==!9arL5YggzxG9u/yOLrnS9+DCkWzV2ndT39/+jMab
G5QfH0PR/9/LBnyQegtuQPZsGa2XU6z2nvaOSfBIDUG2ZczuKv0H4hLtQYyu94JLb
xEbg/EqbbaxeWCvZpQ+abcabcabcabcabcabcabcabcabcabcabcabcfIAKz1QFsB
b8ZwNjztznceC0XzoAEi6PmyCFEwZvNNF+7m8bKLhBGQtly/W1dADFKK1+Tv5Tc7I
NPtpn3JfhM90glabcabcabcabcabcabcabcabcabcabcabcabcabczYFxMbZ5m
SPfxXu3Rdbgz/JYcw/K"
```

Let’s start with Unimus – pt3

```
/container
add root-dir=/usb1/containers/unimus-core remote-
image=croc/unimus-core-arm64:latest interface=veth3
envlist=unimus-core logging=yes
```

We can now manage our network

- We can now manage all devices in our local network from Unimus
- Without VPNs, without any local server, etc.

- Let’s see if everything works…

How about something cool...

- Now that we can manage our network...
- How about we do periodic speedtests (https://speedtest.net) and graph those over time
- We can do that directly from the router with a container!

Lets talk about mounts

- If we want to graph data over time, we need to store it somehow
- However, we mentioned containers don’t keep data, they revert to the image on restart

- So we need to store data somehow, and for this, we use mounts!

How do mounts work

- Basically, we take persistent storage from the host, and map it somewhere inside a container

```
/file
add name=/usb1/containers type=directory
```

- Lets save data from the speedtest container on our USB!

speedtest-tracker

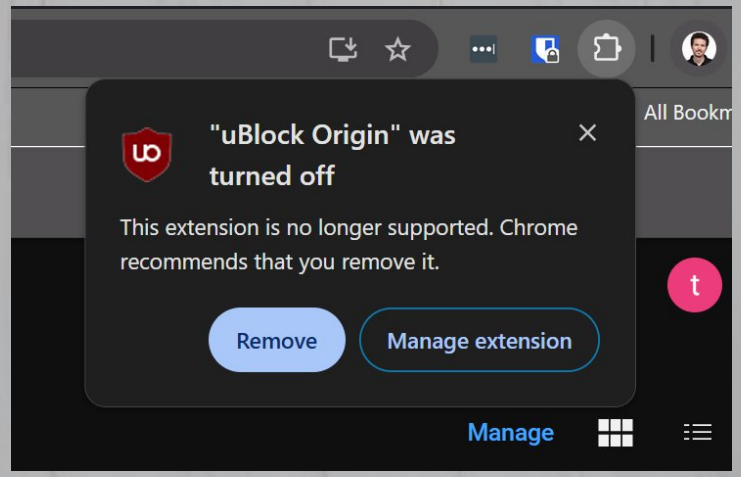
- For convenience (and presentation time saving), we will deploy these through Unimus automation
- Config in the next slides so you can copy it...

Speedtest-tracker config

```
/interface veth
add address=10.44.11.33/24 gateway=10.44.11.1 name=veth4
/interface bridge port
add bridge=br1 interface=veth4
/container envs
add name=speedtest-tracker key=TZ value=Europe/Bratislava
add name=speedtest-tracker key=DB_CONNECTION value=sqlite
add name=speedtest-tracker key=CACHE_STORE value=file
add name=speedtest-tracker key=SPEEDTEST_SCHEDULE value="*/2 * * * *"
add name=speedtest-tracker key=APP_KEY value="base64:QSJSGwavnxF9sV5eaxZzKgjevM0+B6+EtP2NKyaFQQA="
/container mounts
add name=speedtest-tracker-config src=/usb1/containers/speedtest-tracker/config dst=/config
add name=speedtest-tracker-db src=/usb1/containers/speedtest-tracker/database dst=/app/www/database
/container
add root-dir=/usb1/containers/speedtest-tracker/container remote-image=lscr.io/linuxserver/speedtest-tracker:latest
interface=veth4 envlist=speedtest-tracker mounts=speedtest-tracker-config,speedtest-tracker-db logging=yes
```

Continuing with the cool - PiHole

- How about network-wide ad blocking
- PiHole is a DNS server that support ad blocklists
- You can easily save 10% of traffic network-wide transparently for all clients on the network



PiHole config

```
/interface veth
add address=10.44.11.51/24 gateway=10.44.11.1 name=veth5
/interface bridge port
add bridge=br1 interface=veth5
/container envs
add name=pihole key=TZ value=Europe/Bratislava
add name=pihole key=FTLCONF_webserver_api_password value="password"
add name=pihole key=FTLCONF_dns_listeningMode value=all
/container mounts
add name=pihole-config src=/usb1/containers/pihole/config dst=/etc/pihole
/container
add root-dir=/usb1/containers/pihole/container remote-image=pihole/pihole:latest
interface=veth5 envlist=pihole mounts=pihole-config logging=yes
```

Let’s use PiHole

- On router, set PiHole as DNS

```
/ip dns
set servers=10.44.11.51
```

- Since this router is DNS for the rest of the network (through DHCP), we are no blocking ads network-wide
- Test ad blocking - https://canyoublockit.com/

More poller examples

- Let’s look at NetXMS and Zabbix poller
- For convenience (and presentation time saving), we will deploy these through Unimus automation
- Config in the next slides so you can copy it…

NetXMS Remote Agent

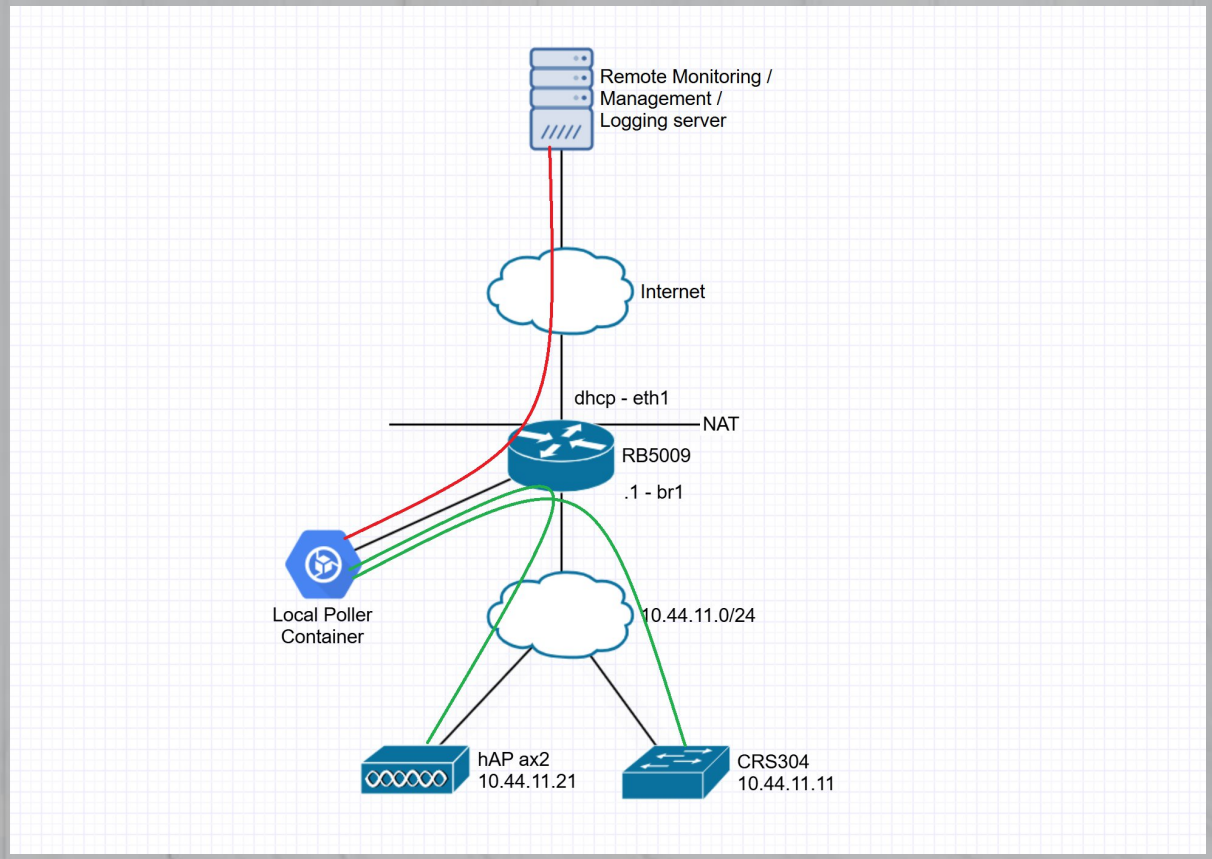
```
/interface veth
add address=10.44.11.34/24 gateway=10.44.11.1 name=veth6
/interface bridge port
add bridge=br1 interface=veth6
/container mounts
add name=netxms-agent-config src=/usb1/containers/netxms-agent/config
dst=/netxms/etc
add name=netxms-agent-data src=/usb1/containers/netxms-agent/data
dst=/netxms/var/lib/netxms
/container
add root-dir=/usb1/containers/netxms-agent/container remote-
image=ghcr.io/alkk/netxms-agent-mikrotik:5.1.4 interface=veth6
mounts=netxms-agent-config,netxms-agent-data logging=yes
```

Zabbix Proxy

```
/interface veth
add address=10.44.11.35/24 gateway=10.44.11.1 name=veth7
/interface bridge port
add bridge=br1 interface=veth7
/container envs
add name=zabbix-proxy key=TZ value=Europe/Bratislava
add name=zabbix-proxy key=ZBX_HOSTNAME value=z-proxy-container.our-net.internal
add name=zabbix-proxy key=ZBX_SERVER_HOST value=zabbix.our-net.internal
/container
add root-dir=/usb1/containers/zabbix-proxy remote-image=zabbix/zabbix-proxy-sqlite3:ubuntu-latest interface=veth7 envlist=zabbix-proxy logging=yes
```

Authentik demo

- Here is a short demo of Authentik…
- Lets make it work with MikroTik through a Radius Outpost
- This adds security due to local proxying (Radius is not encrypted, don’t run it over internet)



Authentik Outpost config

```
/interface veth
add address=10.44.11.61/24 gateway=10.44.11.1 name=veth8
/interface bridge port
add bridge=br1 interface=veth8
/container envs
add name=authentik-outpost key=TZ value=Europe/Bratislava
add name=authentik-outpost key=AUTHENTIK_HOST value=https://auth.our-net.internal
add name=authentik-outpost key=AUTHENTIK_INSECURE value=true
add name=authentik-outpost key=AUTHENTIK_TOKEN value=some-token-value-here
/container
add root-dir=/usb1/containers/authentik-outpost remote-image=ghcr.io/goauthentik/radius:latest interface=veth8 envlist=authentik-outpost logging=yes
```

Let’s use Authentik

- On router, set Radius Client

```
/radius
add address=10.44.11.61 service=login timeout=10ms secret=radius-psk-here
/user aaa
set default-group=write use-radius=yes
/system logging
add topics=radius
```

Final recap

- When should you run something in a container on a router?

- Debugging tools, testing tools – GREAT
- Anything stateless, providing access or convenience – GREAT if not performance limited

- Anything holding data long term – BAD
- Anything critical – EXTRA BAD

That’s it, thank you!

Q&A session