Virtual Network Threat Detection

Network Threat Detection Laboratory

Marc Chamorro

None

Table of contents

1. Intelligent Threat Detection in Virtual Networks

Welcome to the official documentation for the **VNTD project**.

This platform is designed to deploy a **modular**, **scalable**, and fully **virtualized** cybersecurity laboratory. By simulating a segmented enterprise infrastructure, this project facilitates the generation of **real network traffic**, the execution of **simulated attacks**, and the **analysis of security logs** using Artificial Intelligence.

Open Source

This project relies on **open-source** technologies such as Linux, Docker, and Containerlab to ensure accessibility and reproducibility and ease access to research on network simulation and threat detection.

Project Context

This project is developed as a **Final Degree Project (Treball de Final de Grau - TFG)** for the Degree in Computer Engineering in Information Systems and Management at **TecnoCampus (Pompeu Fabra University)**.

It addresses the growing need for accessible research environments in cybersecurity. Traditional physical labs are expensive, difficult to scale and hard to reproduce. This project addresses these limitations by making use of container and network emulation technologies.

Project Information

Author: Marc Chamorro Mollon
Tutor: Pere Barberan Agut
Academic Year: 2025–2026

1.1 Project Resources

View on GitHub

Download Full Documentation (PDF)

Virtualization & Orchestration

Utilizing **Docker** and **Containerlab** to create lightweight and reproducible network nodes that represent realistic infrastructures.

Intelligent Analysis

Integrating **Machine Learning** techniques to detect anomalies and suspicious patterns in network logs that traditional systems might miss.

Real-World Simulation

Implementing **real services** (SSH, FTP, HTTP, MAIL) and security tools (**Suricata**) to generate and monitor real traffic and logs.

1.2 Documentation Roadmap

The documentation is organized to guide you from initial setup to advanced usage:

- **Introduction**: Detailed overview of goals and scope.
- **Getting Started**: Prerequisites and requirements.
- **Installation**: Step-by-step setup guide.
- **Usage**: How to operate and manage the environment.
- **Architecture**: Network design and traffic flows.

March 16, 2026

2. Introduction

The **Intelligent Threat Detection in Virtual Networks** project consists of the design and deployment of a **modular cybersecurity laboratory**. It simulates a realistic, segmented enterprise infrastructure using virtualization technologies to support **training, research, and experimentation**, enabling users to safely study attack techniques, defensive mechanisms, and traffic analysis.

2.1 Purpose and Objectives

The primary objective of this project is to **generate legitimate network security data** by executing simulated attacks in a controlled environment. This data is then centralized, processed, and analyzed in real time to detect anomalies and unwanted patterns.

Specific objectives:

- [x] **Infrastructure Simulation:** Deploying a virtual network that mimics a corporate environment, including DMZ, internal networks and administration zones.
- [x] **Threat Detection:** Implementing IDS (Intrusion Detection Systems) like **Suricata** to monitor traffic and generate logs.
- [x] **Data Analysis:** Centralizing logs (Filebeat) and applying AI algorithms (Isolation Forest) to identify security incidents.

2.2 Architecture Overview

The simulation is designed to be **modular** and **scalable**. The infrastructure mimics a real enterprise network, composed of:

Zone	Purpose
Internet / External	Simulates the public internet and external actors.
Attacker Network	A dedicated segment for generating malicious traffic and executing attacks.
Enterprise Network	The core infrastructure, protected by firewalls and segmented into DMZ, Internal, Monitoring, Users, Administration and Management zones.

> ⚠ **Safe Environment**
>
> This environment is **not designed for production use** and should never be exposed to real external networks. All attacks must be executed within this isolated environment.

2.3 Key Features

- **Container-Based:** Built on **Docker** and **Containerlab**, ensuring the environment is lightweight and portable compared to traditional VM-based labs.
- **Integrated Services:** Nodes run **real services** (Nginx, Postfix, vsftpd) to ensure realistic traffic behavior.
- **AI-Powered:** Utilization of the **Isolation Forest** algorithm detect unusual patterns in complex log data, providing an additional layer of detection beyond fixed rules.

2.4 Target Audience

This platform is intended for:

- **Students:** For practical training in a safe, isolated environment.
- **Researchers:** To experiment with new detection methodologies and dataset generation.
- **Network Administrators:** To test security configurations and topologies with limited resources.

February 21, 2026

3. Getting Started

Ensure your system meets these criteria before proceeding to the installation.

3.1 System Requirements

The environment is designed to be lightweight, but simulating a full enterprise topology with monitoring services requires moderate resources.

- **Architecture:** x86_64 / amd64.

Minimum Requirements

Resource	Requirement
CPU	12 cores
RAM	20 GB
Storage	20 GB free disk space

These requirements allow the complete environment to run including all the monitoring stack.

Reduced Requirements Mode

If the **Elastic stack is disabled**, the environment can run with reduced memory:

Resource	Requirement
CPU	8–12 cores recommended
RAM	16 GB
Storage	20 GB

In this mode the monitoring stack should be disabled.

Hardware Requirements

The monitoring stack (Suricata + Elastic components) is resource intensive. Long execution periods with limited resources may cause instability if system resources are insufficient.

Monitoring Stack

The user can run the environment nonetheless with reduced resources. Although given that not enough resources are available, it is suggested that secondary containers are removed from the topology (e.g. benign / entire floor 2).

Operating System

The project is built and tested on **Linux**.

Ubuntu / Debian Windows macOS

Recommended OS: **Ubuntu 22.04+** or **25.10**. This is the native environment for the automation scripts.

Native Windows is **not** supported directly. You **must** use a dedicated Virtual Machine (VMware, VirtualBox) running Linux.

Not officially tested. A Linux-based Virtual Machine is highly recommended for compatibility with Containerlab.

> **Virtualization Isolation**
>
> It is highly recommended to install this environment inside a **Virtual Machine** or a dedicated system, completely isolated from your personal host settings and system configurations to prevent attacks from leaking into the outside world.

3.2 Software Dependencies

The following tools are required for the installation process and environment management:

1. **Curl:** Required for downloading installation scripts.
2. **Git:** Necessary for cloning the repository and version control.
3. **SSH Client:** Used to connect to the virtual network devices, managed by Containerlab.
4. **Docker:** The core container engine.
5. **Containerlab:** The orchestration tool for the network topology.

3.3 Next Steps

If you meet the requirements, move to the Installation Guide { .md-button }

March 11, 2026

4. Installation Guide

This guide details the **step-by-step process** to set up the **Laboratory** environment.

These instructions assume you are running a fresh installation of **Ubuntu 25.10** (or similar Debian-based OS) within a controlled environment.

The official Containerlab installation process can be found at: Containerlab Install

4.1 1. Prepare the System

Before installing the core tools, ensure your system is up to date and essential utilities are installed.

Update System

```
sudo apt update && sudo apt upgrade -y
```

Install Essential Utilities

Install curl for downloading scripts and ssh for managing connectivity to the virtual nodes.

```
sudo apt install -y curl git
sudo apt install -y ssh
```

> **SSH Installation**
>
> Installing SSH independently is recommended as it is later utilized by Containerlab to manage virtual devices.

4.2 2. Install Containerlab & Docker

Docker is the engine that manages the virtual nodes, and Containerlab is the tool that allows the deployment and connection of containers. While these can be installed separately, the official Containerlab page provides a script to automatically install the latest versions of both services.

Installation

```
curl -sL https://containerlab.dev/setup | sudo -E bash -s "all"
```

To be more specific, this command: - Installs the `git` and `make` packages - Installs Docker - Installs Containerlab - Configures permissions and SSH access

> **Alternative Docker Installation**
>
> Docker may not install properly (a common issue). Alternative commands to install Docker are:
>
> ```
> curl -sL https://containerlab.dev/setup | sudo -E bash -s "install-docker"
> ```
>
> ```
> curl -sSL https://get.docker.com/ | sudo sh
> ```

Configure Permissions

By default, Docker requires root privileges. To run Docker commands as a standard user, you must add your user to the docker group.

```
sudo usermod -aG clab_admins $USER
```

> **Apply Changes**
>
> You must log out and log back in (or restart the VM) for the group membership to take effect.

4.3 3. Install Containerlab

Containerlab orchestrates the Docker containers to form the network topology. The installation is handled by an automated script provided by the Containerlab developers.

```
# Download and install Containerlab
bash -c "$(curl -sL [https://containerlab.dev/setup](https://containerlab.dev/setup))"
```

Containerlab Permissions

To allow Containerlab to manage network interfaces without constant sudo prompts, add your user to the clab_admins group (created during installation).

```
sudo usermod -aG clab_admins "$USER"
```

4.4 4. Verify Installation

Once all components are installed and you have re-logged into your session, verify that the environment is operational.

Verify Docker

Run the "hello-world" container to ensure the Docker daemon is active and accessible.

```
docker run hello-world
```

Verify Containerlab

Check the installed version to ensure the binary is in your PATH.

```
clab version
```

4.5 5. Clone the Repository

Finally, clone the project repository to your local machine to access the topology definitions, scripts, and Dockerfiles.

```
git clone https://github.com/Marc-Chamorro/virtual-network-threat-detection
cd virtual-network-threat-detection/
```

You are now ready to build the images and deploy the labs.

February 21, 2026

5. Usage Guide

The project is controlled primarily through a **centralized automation script**: `run.sh`. This script manages both Docker and Containerlab, ensuring project naming conventions (`_vntd` suffix) are maintained.

5.1 Core Logic: `run.sh`

To start the control menu, navigate to the project root and execute:

```
./run.sh
```

> **Execution**
>
> Change the scripts permissions with: `chmod +x run.sh`.

> **Automation first**
>
> Always prefer using the `run.sh` commands over manual Docker or Containerlab commands to ensure the environment remains consistent with the documentation and security policies.

5.2 Recommended Workflow

For standard usage of the environment, follow these steps:

1. **Preparation:** Ensure your vendor images (like cEOS) are in docker/import/ and run `Image Control` -> `Create` and `Import` to build/import required images.
2. **Deployment:** Go to `Topology Control` -> `Deploy` and select your desired scenario (default scenario: `topology.clab.yml`).
3. **Verification:** Once the deployment finishes, use status of the environment will appear on the screen along the state of all nodes.
4. **Experimentation:** Access the nodes via SSH or `docker exec` to perform traffic generation or security analysis.
5. **Cleanup:** Always run `Topology Control` -> `Destroy` before finishing your session to ensure system resources are released.

5.3 Management Modules

Image Management Topology Management

Access this menu to handle the lifecycle of the Docker containers. All images managed through this menu are automatically appended with the `_vntd` suffix to distinguish them from other images on your system.

- **Create Images:** Scans the `docker/build/` directory and builds every valid image found. It automates the tagging process so the images are ready for deployment.
- **Import Images (.tar.xz):** Scans the `docker/import/` directory for vendor-provided images (e.g., Arista cEOS). It automatically imports and tags them with the previous format.
- **Delete Images:** A cleanup utility that removes all local images containing the `_vntd` tag. This is useful for clearing disk space or forcing a fresh rebuild.
- **Display Images:** Lists all currently available images in your local Docker registry that belong to this project.

> **Pro-Tip: Ignore Images**
>
> Directories in `docker/build/` starting with an underscore (e.g., `_mls`) are **ignored** by the automatic build process.

Orchestrate the network simulation using Containerlab.

- **Deploy Topology:** Displays available topologies within the `labs/` directory and allows you to select one to launch. This command handles the creation of the virtual environment.
- **Destroy Topology:** Stops all running containers from a specific lab and removes the network interfaces and bridges created by Containerlab. This should always be done before closing the machine to avoid future networking issues.
- **Display Available Topologies:** Lists the lab scenarios currently defined in the `labs/` folder and indicates on screen the active/running ones.

> **Cleanup**
>
> Always destroy active topologies before shutting down the system to avoid networking or performance issues.

5.4 Connectivity

Connectivity to a node can be achieved by executing:

```
docker exec -it <container_name> bash
```

The `bash` element opens an interactive shell inside the container; it can be replaced with any other CLI command.

5.5 Extending the Lab

New topologies can be added into the project and may reuse components, coexist with other scenarios while keeping changes at the core of the architecture minimal.

More topologies

Additional topologies can be added to the `labs/` directory. Configuration elements and machines can be reused across multiple nodes.

February 21, 2026

6. Architecture

6.1 Architecture Overview

This project implements a fully **virtualized enterprise network laboratory** designed to simulate realistic network environments and security scenarios. It uses **containers** to emulate routers, switches, firewalls, services, and users while maintaining a low resource usage.

The architecture emphasizes **realism**, **modularity** and **isolation**, making it suitable for learning, experimentation, and security research.

It closely mimics a real-world enterprise network while remaining lightweight and reproducible.

> **Production**
>
> This environment is **not designed for production use** and should never be exposed to real external networks.

Architectural Goals

The architecture is designed to:

- Simulate a **realistic enterprise network environment**
- Apply **network segmentation** using VLANs and **security zones**
- Provide **real network services** (DNS, DHCP, Web, SSH)
- Enable **traffic inspection** and **threat detection**
- Be **reproducible**, **extensible**, and easy to **modify**

> **Ease of Use > Optimization**
>
> The design prioritizes *clarity* and *traceability* of traffic over extreme optimization, ensuring network behavior is clear, specific and debuggable.

Scalability and Extensibility

Because the lab is built on Containerlab and containerized components:

- New services, devices, or entire network segments can be added by modifying the topology YAML.
- Existing images can be reused with different configurations.
- Multiple topologies can be created without changing the architecture

Design Philosophy

The architecture is built upon four main pillars to ensure the environment is suitable for cybersecurity research:

Isolation

The entire laboratory runs within an isolated Docker network, ensuring that simulated attacks do not affect the host machine or real external networks.

Modularity

Network devices are decoupled from their configurations. This allows the same container image to behave differently depending on the services enabled and attached configuration files.

Observability

All parts of the network are designed to be monitored, with dedicated zones for IDS and centralized logging.

Explicit Configuration

All IP addresses, routing capabilities, VLANs, and service behavior are explicitly defined in YAML and scripts to avoid hidden defaults.

Security Zones

The infrastructure is logically divided into four functional zones to prevent unrestricted movement:

Zone	Purpose
Internet Core	Acts as the central exchange point connecting all external and internal elements.
Attacker Network	Represents external threats and hosts the Kali Linux node.
Benign Network	Represents legitimate external users interacting with enterprise services
Enterprise Infrastructure	The core of the project, featuring a segmented architecture with a firewall, DMZ, internal services, and user floors.

Enterprise Isolation

The enterprise zone is isolated by routing and firewall rules to prevent unrestricted lateral movement. Movement is controlled.

Scope

- **Network Design**: Topology structure and component roles.
- **Addressing & VLANs**: IP planning and segmentation strategy.
- **Traffic Flows**: Expected communication paths and control points.

February 21, 2026

6.2 Network Design

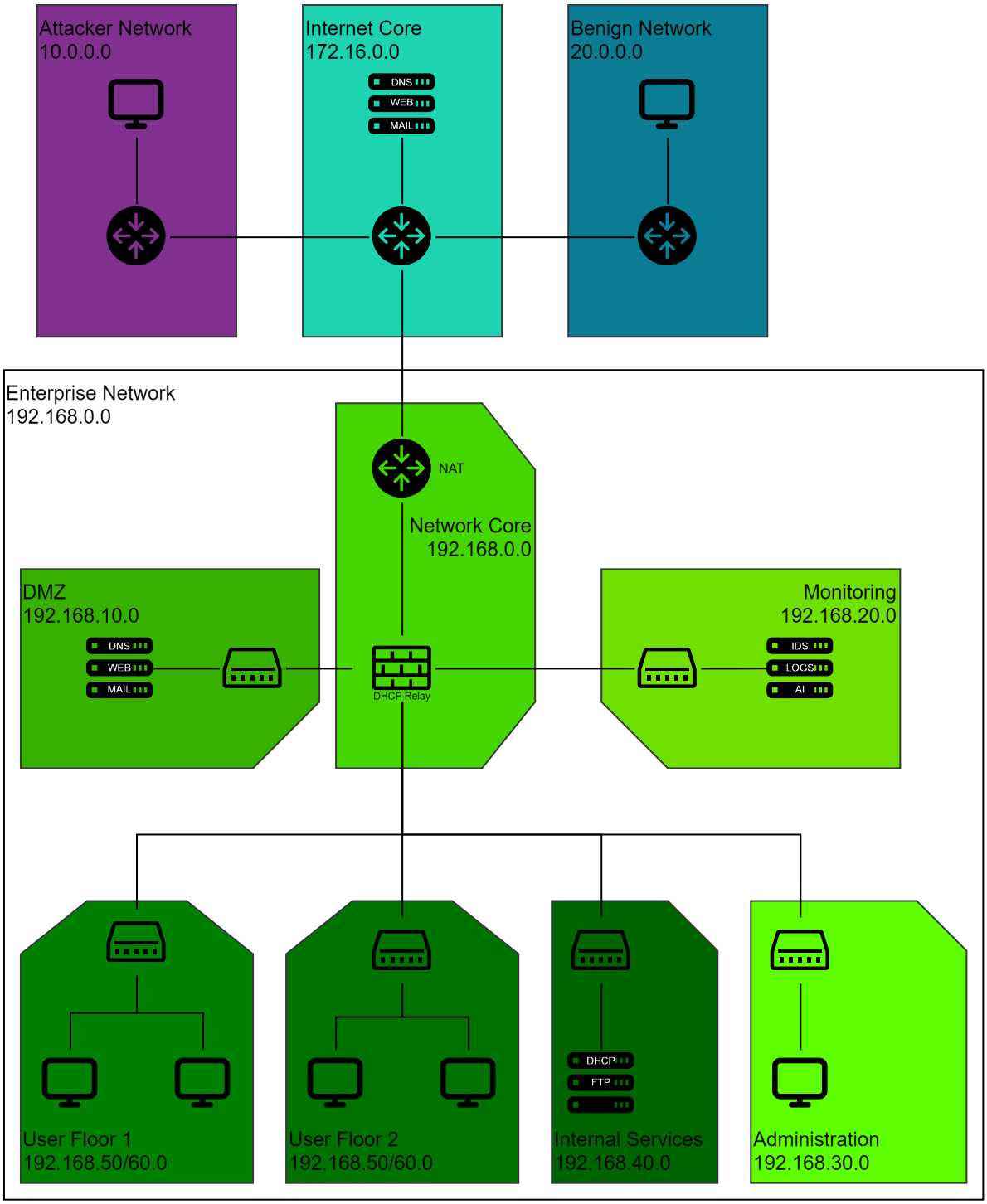
This page describes the **logical and physical design** of the VNTD network topology, focusing on how components are interconnected and how responsibilities are distributed across the infrastructure.

Topology Overview

The topology simulates a **multi-zone enterprise network** connected to external devices. It is built around a central Internet core and a segmented enterprise network protected by a router and a firewall.

The topology consists of:

- An Internet core router.
- Separate external networks (attacker and benign).
- An enterprise router.
- A central firewall.
- Multiple VLAN-based internal segments.



Provided topology

```
graph TD
    subgraph External_Networks
        Attacker[Attacker Network - Kali]
        Benign[Benign Network - Alpine]
    end

    subgraph Internet_Infrastructure
        Router_Internet[Router Internet]
        Server_Internet[Internet Server]
    end

    subgraph Enterprise_Core
        Router_Edge[Enterprise Edge Router]
        Firewall[Central Firewall]
```

```
    end

    subgraph Internal_Segments
        DMZ[VLAN 10 - DMZ Services]
        Logwatch[VLAN 20 - Monitoring]
        Admin[VLAN 30 - Management]
        Services[VLAN 40 - Internal Services]
        Users[VLAN 50/60 - User Floors]
    end

    Attacker --- Router_Internet
    Benign --- Router_Internet
    Router_Internet --- Server_Internet
    Router_Internet --- Router_Edge
    Router_Edge --- Firewall
    Firewall --- DMZ
    Firewall --- Logwatch
    Firewall --- Admin
    Firewall --- Services
    Firewall --- Users
```

External Networks

Internet Core

The router (`router_internet`) located at the center of the topology represents the public Internet core. It serves as the interconnection point for:

- External benign traffic.
- External attacker traffic.
- Enterprise outbound and inbound traffic.

This router uses FRRouting (FRR) to provide realistic routing behavior.

Within the network, a server (`internet_server`) can be appreciated, simulating common internet services.

Attacker Network

The attacker network simulates a hostile external actor:

- A dedicated router (`router_attacker`).
- A Kali Linux-based attacker node.

This network is intentionally separated to allow controlled attack generation.

> **Warning**
>
> Attack simulations should only be executed inside this isolated environment.

Benign Network

The benign network simulates legitimate external users:

- A dedicated router (`router_benign`).
- A lightweight client node.

This allows differentiation between malicious and legitimate traffic.

Configuration Persistence

All configurations for routers and switches are decoupled from the images and mounted via bind in the topology definition file. Changes applied directly on the machines do not persist.

Enterprise Core

Enterprise Edge Router

The `router_enterprise` node connects the enterprise network to the Internet. Its responsibilities include:

- Routing between enterprise and external networks (`NAT`).
- Forwarding traffic toward the firewall.
- Acting as a clear limit between external and internal domains.

Firewall

The firewall is the **central enforcement point** of the enterprise:

- Enforces inter-VLAN policies.
- Controls inbound and outbound traffic.
- Mirrors traffic to the Logwatch.
- Hosts DHCP relay functionality.
- Acts as the default gateway for all VLANs.

VLAN Communication

All enterprise VLANs are isolated by default. Inter-VLAN communication is only possible through explicit firewall rules.

Layer 2 Segmentation

VLANs are implemented using the custom firewall images, but L2 package traffic is managed with **Arista cEOS switches**, providing realistic L2 behavior:

- Access ports for end devices.
- Trunk ports for multi-VLAN floors.
- Clear separation between zones.

Each VLAN maps to a dedicated VLAN gateway. This VLAN gateway represents the firewall. The firewall works both as a L3 switch and as a security device.

Monitoring Placement

Traffic monitoring is performed by a dedicated node located in the **Monitoring VLAN (VLAN 20)**, reading all traffic outgoing and incoming the enterprise network, ensuring:

- Visibility into enterprise traffic.
- Isolation from services and users.

Traffic is mirrored by the firewall using the **iptables TEE mechanism**, combined with the node operating in **promiscuous mode**, allows the node to receive packets without interfering with the original flow.

Tip

This placement ensures maximum visibility with minimal configuration.

Monitoring node

It acts purely as an **observer**, ensuring the integrity of the simulated network traffic.

This node, called **logwatch**, acts as a centralized analysis system responsible for capturing, processing and visualizing network logs. Such is composed of four main components:

- **Suricata** — Intrusion Detection System (IDS), inspects mirrored packets and generates registers.
- **Filebeat** — Log shipper, collects Suricata logs and forwards them to Elasticsearch for processing.
- **Elasticsearch** — Log storage and indexing, stores and indexes log data for efficient processing and analysis.
- **Kibana** — Visualization interface, provides dashboards and visual analysis tools.

Unlike traditional IDS where components are separated, this project integrates the entire monitoring components within a single container. This design simplifies deployment and reduces the number of containers used.

Communication

This node **cannot sends traffic back into the network**, ensuring it does not interfere with normal network operation.

March 11, 2026

6.3 Addressing & VLANs

The VNTD laboratory uses a predictable **IPv4 addressing scheme** designed to simplify debugging and align with common enterprise practices.

VLAN and Subnet Mapping

Every enterprise function is mapped to a dedicated VLAN and subnet, with the Firewall serving as the gateway. External networks use distinct address spaces to avoid overlap.

VLAN	Name / Purpose	Subnet	Gateway
-	Router - Router	172.16.x.0/30	—
-	Internet Core	172.16.100.0/24	172.16.100.1
-	Attacker Network	10.0.0.0/24	10.0.0.1
-	Benign Network	20.0.0.0/24	20.0.0.1
10	DMZ	192.168.10.0/24	192.168.10.1
20	Monitoring & Logwatch	192.168.20.0/24	192.168.20.1
30	Administration	192.168.30.0/24	192.168.30.1
40	Internal Services	192.168.40.0/24	192.168.40.1
50	User Floor 1 & 2	192.168.50.0/24	192.168.50.1
60	User Floor 1 & 2	192.168.60.0/24	192.168.60.1

IPv6

IPv6 is intentionally out of scope for this project.

Gateway

For all enterprise VLANs:

- The firewall interface is the default gateway.
- No direct routing exists between VLANs.
- NAT and forwarding decisions are centralized.

Note

This design simplifies troubleshooting and ensures all inter-zone traffic is visible from a single point.

IP Assignment Strategy

The project employs a hybrid model for IP assignment to reflect realistic corporate environments.

Static Assignment **Dynamic Assignment (DHCP)**

Used for core infrastructure nodes to ensure reliability:

- **Routers & Firewalls:** Manually configured in startup scripts.
- **Servers:** DMZ and Internal servers use fixed IPs (e.g., `192.168.10.10`).

Used for end-user workstations in VLANs 50 and 60:

- **Server:** Centrally managed by `internal_server` (VLAN 40).
- **Relay:** The Firewall hosts the `isc-dhcp-relay` service to bridge requests across VLANs.

DNS Addressing Considerations

DNS servers are intentionally placed in different zones, which allows testing of internal vs external name resolution services.

- Internal DMZ DNS: `dmz_server`
- External DNS: `internet_server`

DNS Infrastructure

DNS servers are intentionally placed in different zones, which allows testing of internal vs external name resolution services. - **Internal DNS (`dmz_server`):** Resolves local hostnames and forwards unknown requests to the internet server. - **External DNS (`internet_server`):** Simulates public DNS service.

March 11, 2026

6.4 Traffic Flows

This page documents the **expected traffic flows** within the architecture and the **security logic** enforced by the network devices. Understanding these flows is critical for debugging and validating the network.

Routing & Connectivity Logic

The environment uses a hybrid routing model to ensure internal isolation while maintaining controlled external accessibility.

External Routing (OSPF)

The **Enterprise Edge Router** (**router_enterprise**) uses OSPF to communicate with the Internet Core: - **Advertisement:** It announces the public address (`172.16.30.2/30`). - **Static Internal Routing:** To ensure traffic reaches internal VLANs, the router has static routes pointing all `192.168.0.0/16` traffic to the **Firewall** (`192.168.0.2`).

Internal Routing

- **Default Gateway:** The firewall acts as the default gateway for all enterprise VLANs (10, 20, 30, 40, 50 & 60).
- **Default Route:** The firewall sends all unknown traffic to the Enterprise Router interface (`192.168.0.1`).

Security Policies (iptables)

The Firewall implements a **Default DROP** policy for all INPUT and FORWARD packets. Only explicit flows are permitted.

Core Firewall Rules

- **Stateful Inspection:** All established and related traffic is allowed via `conntrack` , ensuring response packets from valid connections are not blocked.
- **Management:** ICMP (Ping) is allowed from the enterprise subnets to the firewall itself for diagnostic and testing purposes.
- **Loopback:** Full access is allowed on the local loopback (`lo`) interface.

> **Policies format**
>
> Policies are designed to be explicit rather than permissive.

Inter-Zone Communication

Source	Destination	Permitted Traffic / Protocols
Admin (VLAN 30)	Any	Unrestricted full access to all subnets and Internet
Users (50/60)	Internet, DMZ, Services	Web browsing, internal services access, and DNS queries
Services (40)	Admin, Users	General connectivity and responses to internal requests
Monitoring (20)	-	None, restricted to passive observation
DMZ (10)	Internet	DNS forwarding and Outbound Mail Relay (SMTP)

> **Intra-VLAN Traffic**
>
> Traffic between devices from the same VLAN is permitted and handled by the Arista L2 switches without firewall intervention (Given that both devices are found within the same switch).

Specific Protocol Flows

Inbound (DNAT)

Accessing enterprise services from the Internet (DMZ Server) consists of a two-step process:

1. **Router Enterprise:** Internet -> Firewall Link

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j DNAT --to-destination 192.168.0.2
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 25 -j DNAT --to-destination 192.168.0.2
```

2. **Central Firewall:** Router Link -> DMZ Server

```
iptables -t nat -A PREROUTING -i eth1 -p tcp -m multiport --dports 80,22,53,25 -j DNAT --to-destination 192.168.10.10
iptables -t nat -A PREROUTING -i eth1 -p udp --dport 53 -j DNAT --to-destination 192.168.10.10
```

Outbound (SNAT)

Accessing the Internet from within the network consists of:

1. **Firewall:** Ensures communication policies are met, and if so, redirects traffic to the Router.
2. **Router Enterprise:** Allows communication with the outside world through Masquerade.

Infrastructure Service Flows

DHCP Relay Mechanism DNS Resolution Mail Service (SMTP/IMAP)

Since the **DHCP Server** is found in VLAN 40, but clients are in VLAN 50/60, the firewall bridges the gap performing a relay:

1. **Client**: Broadcasts a `DHCPDISCOVER` on its local bridge.
2. **Firewall**: The `isc-dhcp-relay` service picks up the broadcast and forwards it as **unicast** to `192.168.40.10`.
3. **Server**: Sends a `DHCPOFFER` back to the firewall relay.
4. **Firewall**: Forwards the offer back to the originating VLAN floor.

The DMZ server provides DNS service to individuals within the enterprise network. Devices using DHCP will receive the DNS address automatically upon IP assignment. This service responds to the name addresses referencing the internal server.

Given the situation that an unknown address is received, the DNS forwards the request to the DNS located on the Internet Server:

- **Internal Query**: Clients query the `dmz_server` (`192.168.10.10`).
- **External Forwarding**: If the `dmz_server` cannot resolve the name, it forwards the request to the `internet_server` (`172.16.100.100`) at the Internet core.

The mail architecture follows a hub-and-spoke model:

- **Outbound**: Internal clients send mail to the DMZ Server, which relays it to the Internet Mail Server (if the destination is not the server itself).
- **Inbound**: The Internet server forwards mail to the Enterprise IP, where it is sent with DNAT through the router and firewall to port 25 of the DMZ node.
- **Retreival**: Users access their mail via IMAP on port 143 (unencrypted for inspection purposes).

Traffic Mirroring (IDS)

To enable threat detection without modifying the traffic flow or introducing latency, the firewall implements traffic mirroring using the `TEE` function. And with the addition of the **Logwatch** node, the architecture supports a complete security flow:

```
sequenceDiagram
    participant Ext as Internet Core
    participant FW as Firewall
    participant DMZ as DMZ Server
    participant IDS as Logwatch

    Ext->>FW: Malicious Packet (eth1)
    Note over FW: iptables TEE Rule
    FW-->>IDS: Cloned Packet (VLAN 20)
    FW->>DMZ: Original Packet
    IDS->>IDS: Storage and processing
```

- **Mechanism:** Every packet going through the Internet-facing interface (`eth1`) of the firewall is cloned and sent to the **IDS node** (`192.168.20.10`).
- **Security Constraint:** The IDS node is strictly prohibited from sending any traffic back into the network, making sure it remains a passive observer.

The **logwatch node** receives mirrored traffic and performs the following processing pipeline:

```
graph LR
    FW[Central Firewall] -->|TEE / Traffic Mirroring| IDS[Suricata IDS]
    IDS -->|JSON Event Logs| FB[Filebeat]
    FB -->|Port 9200| ES[Elasticsearch]
    ES -->|Visualization| KB[Kibana]

    subgraph Monitoring_Stack [Logwatch Node]
        ES
        KB
```

```
        FB
    end
```

1. **Packet Inspection** Suricata analyzes packets in real time and generates structured JSON registers.
2. **Log Shipping** Filebeat monitors Suricata output files and forwards the events to Elasticsearch.
3. **Indexing** Elasticsearch parses and stores the logs.
4. **Visualization** Kibana allows the exploration of events, build dashboards, and analyse network patterns.

March 11, 2026

7. Docker & Containerlab

7.1 Docker & Containerlab Overview

The project relies on the combination of **Docker** and **Containerlab** to provide a lightweight, isolated, and fully reproducible laboratory environment. While Docker manages individual software images and containers, Containerlab serves as the manager that defines how these nodes are interconnected to form a network.

> **Evolving project**
>
> This documentation evolves alongside the code. If something outdated is spotted or can be improved, feel free to propose changes directly on GitHub.

Design Philosophy

The environment is built around three core principles to ensure its utility for cybersecurity research and training:

Reproducibility

Every user runs the exact same software versions and configurations, eliminating inconsistencies between different host environments.

Modularity

Images are single-purpose (e.g., router, web server, firewall) to mimic physical network devices.

Persistence

Containers function as always-on hardware devices. They utilize specific entrypoints to remain active and keep their state intact throughout the lab session.

System Integration

The collaboration between these technologies is structured as follows:

- **Docker's Role:** Provides the operating system, including the operating system layer, networking capabilities, and essential utilities (e.g., `iproute2`, `frr`, `nginx`) for each node.
- **Containerlab's Role:** Instantiates containers from these images, creates virtual network links, aand connects the topology according to the `.clab.yml` definition file.

Additional Testing Topologies

In addition to the primary enterprise topology, the project includes **simplified testing environments**.

These topologies are located in the `labs/` directory:

- `V1_Testing.clab.yml`
- `V2_Testing.clab.yml`

These environments are intentionally smaller and are used for:

- Rapid testing of configuration changes.
- Debugging services without deploying the full infrastructure.
- Experimenting with simplified network scenarios

> **Smaller environments**
>
> These testing topologies reuse the same images and configuration patterns used in the main environment but reduce the number of nodes to improve startup times.

Navigation

- **Image Catalog**: Detailed breakdown of custom images available in the project.
- **Dockerfile Standards**: Guidelines for building consistent and efficient images.
- **Entrypoints & Behavior**: Initialization and dynamic service management.
- **External Images**: How to use vendor-proprietary images like Arista cEOS.

March 11, 2026

7.2 Image Catalog

All custom images developed for this project are built with the `_vntd` suffix to ensure they remain unique within the local Docker registry and prevent naming conflicts.

Operating System Base Images

The project utilizes two primary base environments configured for either robust infrastructure services or lightweight user endpoints.

Debian Slim Alpine Linux

- **Image Identifier:** `debian:12-slim`
- **Usage:** Serves as the foundation for most infrastructure nodes, including routers, firewalls, and servers.
- **Characteristics:** Selected for its stability, modern Linux environment with a small footprint, and minimal storage footprint, making it ideal for simulating general-purpose Linux routers and servers.

- **Image Identifier:** `alpine_vntd` (based on `wbitt/network-multitool:alpine-extra`)
- **Usage:** Simulates user workstations and lightweight end-nodes.
- **Technical Detail:**
- `dhcpcd`**:** Allows users to receive IP address automatically. Relies on the startup script, so the machine waits for the DHCP server to start and offer an address. Otherwise, it believes no DHCP server is available and stops asking.
- `lftp`**:** Client FTP service.
- `mutt`**:** Mail User Agent. Binds a configuration file for a specific user so that each machine has an already included profile and no manual action is needed. Relies on the startup script, to create a couple of mandatory directories.
- **Environment Variables:** You can control this container using the following variables in your topology file *(these attributes are not applied on the* `entrypoint`*, but rather on the* `startup` *file)*:
- `DHCP_CLIENT=1`: Starts the DHCP client service daemon, which makes it so that it doesn't stop asking for an address until one is assigned.
- `IFACE="eth1"`: Define the interface used on the device. If none is assigned, it'll use `eth1` by default.
- `MUTT_CLIENT=1`: Starts the `mutt` client configuration process. Requires a user configuration file to be assigned.

Network & Infrastructure Nodes

Router (`router_vntd`)

A general-purpose Linux router. Unlike the official FRR image, this image is built on Debian and installs the FRR service via package managers, allowing for more system-level manipulation.

- **Base:** `debian:12-slim`
- **Key Packages:**
- `frr`, `frr-pythontools`: For dynamic routing (OSPF, BGP).
- `iptables`, `iproute2`: For packet manipulation and static routing.
- `tcpdump`: For traffic capture.
- **Configuration:**
- **IP Forwarding:** Enabled at build time by setting `net.ipv4.ip_forward=1` in `/etc/sysctl.conf`.
- **OSPF:** Enabled at build time by setting `ospfd=yes` in `/etc/frr/daemons`.

FRR (`frr_vntd`)

A direct import of the official FRRouting image.

- **Source:** `quay.io/frrouting/frr:10.5.0`
- **Use Case:** Pure routing between networks where realism and network tools are less critical.

> **Important**
>
> This image does not support NAT; hence the creation of a dedicated router image.

Firewall (`firewall_vntd`)

A dedicated node for simulating network security boundaries. Using a custom image helps it start faster, use fewer resources, and simplifies network rules management.

- **Base:** `debian:12-slim`
- **Key Packages:**
- `iptables`: The core packet filtering tool.
- `conntrack`: Enables stateful inspection capabilities.
- `bridge-utils`: For transparent bridging scenarios.
- `isc-dhcp-relay`: Relaying DHCP traffic between clients and the DHCP server.
- **Startup Behavior:**
- Flushes all existing `iptables` rules (NAT, Mangle, Filter) on boot.
- Enables IP Forwarding.
- Sets default policies (INPUT/FORWARD DROP, OUTPUT ACCEPT)
- **Environment Variables:** You can control this container using the following variables in your topology file:
- `DHCP_RELAY=1`: Starts the DHCP relay service (configuration files are required to make this service work).

MLS (`mls_vntd`)

> **Ignore images**
>
> This directory starts with `_` and is currently ignored by build scripts.

Intended to simulate a Multi-Layer Switch.

- **Key Packages:** `bridge-utils` (brctl).
- **Behavior:** Cleans network tables and enables forwarding.

Service Nodes

Server (`server_vntd`)

An image designed to simulate an endpoint server providing various network services. Features can be toggled via environment variables.

- **Base:** `debian:12-slim`
- **Services Included:**
- **SSH:** `openssh-server` (Configured to allow password authentication).
- **Web:** `nginx`.
- **DHCP:** `isc-dhcp-server`.
- **DNS:** `dnsmasq`.
- **FPT:** `vsftpd`.
- **MAIL:** `postfix`, `dovecot-core`, `dovecot-imapd`.
- **Environment Variables:** You can control this container using the following variables in your topology file:
- `SSH_SERVER=1`: Starts the SSH daemon. Creates user `vntd` with password `pswd`.
- `WEB_SERVER=1`: Starts Nginx and serves a default HTML page.
- `DHCP_SERVER=1`: Intended to start the DHCP service (configuration files are required to make this service work (2)). Requires the following variables to be configured for the service to properly work (used for the system to wait for the parameters before starting the server; if not, the service would crash).
- `IFACE: "eth1"`: Define the interface used on the device.
- `IP_ADDR: "192.168.40.10"`: Define the IP address to be assigned to the interface.
- `DNS_SERVER=1`: Start the DNS service (configuration files are required to make this service work).
- `FTP_SERVER=1`: Intended to start the FTP service (configuration files are required to make this service work (2)).
- `MAIL_SERVER=1 || MAIN_MAIL_SERVER=1`: Starts the mail server as a secondary or primary mail provider (requires multiple configuration files (3)).

Security & Monitoring

Kali (`kali_vntd`)

A simulation of an attacker machine.

- **Base:** `kalilinux/kali-rolling`
- **Key Packages:**
- `nmap` : Port scanning.
- `openssh-client` : Remote connectivity.
- Standard network tools: `iproute2` , `net-tools` , `curl` .

Logwatch (`logwatch_vntd`)

The `logwatch` node is responsible for centralized monitoring and threat detection within the VNTD environment, this consists of a **single container running multiple services**.

This approach simplifies deployment and reduces inter-container dependencies.

- **Base:** `debian:12-slim`
- **Services Included:**
- `suricata` : Network intrusion detection and traffic inspection.
- `filebeat` : Log collection and forwarding.
- `elasticsearch` : Log storage and indexing.
- `kibana` : Log visualization and dashboards.
- **Environment Variables:** You can control this container using the following variables in your topology file:
- `IP_ADDR: "192.168.20.10"` : Define the IP address to be assigned to the interface.
- `IP_GTWY: "192.168.20.1"` : Define the IP gateway address to be assigned to the interface.
- `IFACE: "eth1"` : Define the interface used on the device.
- `SURICATA_SERVICE=1` : Starts the suricata service on the defined interface.
- `ELASTIC_STACK=1` : Starts all elastic services (configuration files are required to make these services work). These are:
- `filebeat`
- `elasticsearch`
- `kibana`

> **Resource consumption**
>
> Monitoring services can take a significant amount of time to fully initialize, especially the elastic components. It is normal for the stack to require **a couple of minutes** before all services become operational.

March 11, 2026

7.3 Dockerfile Standards

To maintain consistency across the laboratory environment, all custom images adhere to a strict set of guidelines. These standards ensure that images are lightweight, non-interactive during build, and persistent during runtime.

Design Considerations

1. Non-Interactive Environment

To prevent build failures caused by package managers requesting user input, images are configured for non-interactive environments.

```
ENV DEBIAN_FRONTEND=noninteractive
```

2. Image Optimization (Cleanup)

To keep image sizes minimal, cached package lists are removed within the same RUN instruction once the software is installed. This prevents the cache from being stored unnecessarily.

Pattern:

```
RUN apt-get update && apt-get install -y \
    <package_name> \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*
```

This prevents the cache from being stored unnecessarily.

3. Service Persistence

In a standard Docker environment, a container exits as soon as its main process finishes. However, a network node (like a router) must stay alive even if it's doing nothing but listening.

If no specific service (like a web server) keeps the container occupied, use:

```
CMD ["sleep", "infinity"]
```

Standard Package Sets

Most container images (routers, switches, endpoints) include a set of network diagnostic tools to facilitate debugging during labs.

Network Diagnostic Tools

- `iproute2` (`ip`): Modern interface configuration.
- `net-tools` (`ifconfig`, `netstat`): Controlling network subsystem.
- `iputils-ping` (`ping`): Connectivity testing.
- `traceroute`: Path analysis.
- `tcpdump`: Packet capture.
- `curl`: HTTP connectivity testing.

Network Control

- `procps`: Provides sysctl for enabling IP forwarding.

- `iptables`: Packet filtering.

February 21, 2026

7.4 Entrypoints & Runtime Behavior

The `entrypoint.sh` script defines the **logic** to be executed at startup time. Unlike the `Dockerfile` (which defines the static content), the entrypoint defines the **runtime configuration** once the container initializes.

The Role of Entrypoints

In this virtual lab environment, entrypoints serve three critical functions:

1. **Network Initialization:** Enabling IP forwarding or bridging, allowing containers to function as routers.
2. **Sanitization:** Clearing pre-existing firewall rules to ensure no leftover configurations interfere with the lab topology.
3. **Service Orchestration:** Starting background daemons based on variables before holding the container open.

Common Routines

Enabling IP Forwarding

For a Linux container to act as a router (passing packets between interfaces), IP forwarding must be enabled in the kernel. This is typically done in the entrypoint:

```
sysctl -w net.ipv4.ip_forward=1
```

Firewall Sanitization

Images like `firewall_vntd` and `router_vntd` often include commands to flush iptables rules. This ensures that the device starts with a known state, rather than inheriting random rules or Docker's default NAT rules that might interfere with the lab topology.

```
# Example from firewall entrypoint
iptables -F         # Flush filter table
iptables -t nat -F  # Flush NAT table
iptables -X         # Delete user-defined chains
```

The "Keep-Alive" Loop

Because containers are *ephemeral* (lasting for a short time), the script must not end. If the script finishes, the container dies. The standard way to keep the node active in Containerlab is:

```
sleep infinity
```

Special Cases

The server entrypoint utilizes environment variables passed by Containerlab to decide which services to launch at boot.

> **Dynamic Entrypoint**
>
> This feature is not limited to the `server_vntd` image, other images also offer this feature. Although not as many services are provided.

Logic Flow Example:

```mermaid graph TD Start[Container Start] --> CheckSSH{SSH_SERVER=1?} CheckSSH -- Yes --> StartSSH[Create user vntd & Start SSHD] CheckSSH -- No --> CheckWeb{WEB_SERVER=1?} CheckWeb -- Yes --> StartWeb[Generate HTML & Start Nginx] CheckWeb -- No --> Persistence[Execute sleep infinity] StartSSH --> CheckWeb StartWeb --> Persistence
```

Example

1. Check if `SSH_SERVER=1` -> Configure keys, create user, start sshd.
2. Check if `WEB_SERVER=1` -> Generate index.html, start nginx.
3. Execute `sleep infinity`.

February 21, 2026

7.5 Supported External Images

While most images are built from source Dockerfiles, the environment supports the integration of vendor-provided images.

Import Directory

The automation scripts automatically search for compressed image archives in the following directory:

`docker/import/`

> **Directory Requirement**
>
> If the `import` directory does not exist, it must be created manually before running the build scripts.

Arista cEOS

The primary use case for the import functionality is integration of **Arista cEOS**. This allows users to work with an industry-standard CLI (Command Line Interface) indistinguishable from physical Arista switches.

> **Important**
>
> Arista cEOS does not support assigning access lists to VLANs or ports. This feature is locked.

Requirements

- **Format:** `.tar.xz`.
- **Naming:** The script names the image by trimming the filename at the first `-` and lower casing its characters.
- **Tagging:** The imported image will be tagged based on the previous recovered name and appended with `_vntd`.
- *Example:* `cEOS-lab-4.32.0F.tar.xz` -> `ceos_vntd`.

Licensing, Download & Import

> **Proprietary Software**
>
> Arista cEOS is **not open source** open source. Users must register for a valid account and accept the vendor's EULA to download the image from their support portal.

1. Register at Arista Software Downloads.
2. Navigate to **cEOS-lab** releases.
3. Download the `.tar.xz` archive.
4. Place the file in `docker/import/`.
5. Run the project's build script.

February 21, 2026

8. Labs

8.1 Labs Overview

The **Labs** module represents the core of the project's network simulation. It defines the virtual network topologies that deployed to simulate enterprise environments.

This section covers the logic behind topology definition, the orchestration process, and how individual nodes are transformed into functional network devices.

Key Concepts

The laboratory environment is built upon three fundamental layers:

Topology Definition

The blueprint of the network, defined in YAML format. It specifies which nodes to create, their resource limits, and how to wire them together.

Node Configuration

The internal logic of each device. This includes routing protocols (OSPF), security policies (iptables), and switch configurations (VLANs).

Runtime Environment

How Containerlab instantiates the topology using Docker.

Architectural Relationship

The labs are designed to be completely modular. By changing a single configuration file or a link in the topology definition, the environment can turn from a simple connectivity test to a full enterprise infrastructure with a DMZ and multi-floor workstation segments.

Persistence Strategy

To maintain reproducibility, follow a strict **configuration independence** strategy. Configurations are never stored inside the container images; they are injected at runtime using bind mounts.

Navigation

- **Topology File**: A deep dive into the `*.clab.yml` structure and syntax.
- **Lab Configuration**: Detailed explanation of how routers, firewalls, and switches are configured.
- **Available Architectures**: Detailed breakdown of the provided environment.

February 21, 2026

8.2 Topology Definitions

The project uses **Containerlab**'s YAML-based topology definition format (`.clab.yml`) to define the virtual network. This file acts as a *manager*, telling Docker which containers to start and how to link their network interfaces.

File Structure

A standard topology file in is organized into two primary sections: `nodes` and `links`.

```
name: virtual-env

topology:
  nodes:
    # Node definitions ...
  links:
    # Link definitions ...
```

Nodes

The `nodes` section defines every device in the network. Each entry specifies:

> **Mandatory & Optional**
>
> Mandatory → **M** Optional → **O**

- `kind` [M]: The type of node (e.g., `linux` for standard containers, `arista_ceos` for switches). The kind must be supported by Containerlab.
- `image` [M]: The Docker image to use (e.g., `frr_vntd:latest`, `kali_vntd:latest`).
- `startup-config` [O]: Provide startup configuration that the node applies on boot.
- `binds` [O]: Files from the host's device that are mounted into the container to inject configurations.
- `exec` [O]: Commands to run immediately after the container starts (useful for IP assignment).
- `env` [O]: Environment variables (e.g., enabling `SSH_SERVER`).
- `dns` [O]: Used to assign a DNS server address easily, accompanied usually of `servers` with a list of all addresses.

> **Exec & Binds interaction**
>
> `exec` and `binds` are commonly used together to inject configuration files into the container and then execute such scripts upon startup.

Additional sections definitions can be added to change the behavior and capabilities of devices to easily test the behavior under high-stress environments:

- `cpu` [O]: Define a maximum number of CPU cores to be assigned to a specific node. (e.g., `2`)
- `memory` [O]: Maximum amount of RAM available to a single node. (e.g., `4GB`)

Example: Internet Router

```
router_internet:
  kind: linux
  image: frr_vntd:latest
  binds:
```

```
    - ./config/router/daemons:/etc/frr/daemons
    - ./config/router/internet/frr.conf:/etc/frr/frr.conf
```

Example: Internet Server

```
internet_server:
  kind: linux
  image: server_vntd:latest
  env:
    WEB_SERVER: 1
    SSH_SERVER: 1
    DNS_SERVER: 1
  binds:
    - ./config/server/dns/internet/dnsmasq.conf:/etc/dnsmasq.conf
  exec:
    - ip addr add 172.16.100.100/24 dev eth1
    - ip link set eth1 up
    - ip route del default
    - ip route add default via 172.16.100.1
```

Example: Attacker

```
attacker:
  kind: linux
  image: kali_vntd:latest
  exec:
    - ip addr add 10.0.0.2/24 dev eth1
    - ip link set eth1 up
    - ip route del default
    - ip route add default via 10.0.0.1
  dns:
    servers:
      - 172.16.100.100
```

Links

The `links` section defines the virtual cables connecting the nodes. Each link is defined by a pair of endpoints in the format `"node_name:interface_name"`.

Example: Connecting Internet Router to Enterprise Router

```
links:
  - endpoints: ["router_internet:eth3", "router_enterprise:eth1"]
```

Runtime Artifacts

When a topology is deployed, Containerlab creates a directory named `clab-<topology_name>` (e.g., `clab-virtual-env`).

This directory contains:

- Generated certificates
- Node-specific runtime data
- Node configuration files

> **Artifacts persistence**
>
> The `clab-virtual-env` directory is **not permanent** and managed by Containerlab. Do **not** store permanent configurations here, as they may be wiped on redeployment. Always use the `config/` directory for persistent data or any other directory of your choice.

Startup Dependencies

Complex environments may require a **controlled startup order** to ensure that services start only after the required infrastructure is available.

Containerlab allows defining these dependencies using **groups**, **stages**, and **healthchecks**.

This project uses this mechanism to guarantee that critical infrastructure components start in a specific order.

Health Checks

A **healthcheck** verifies that a container is operational before allowing other components to start.

Example used in the monitoring node:

```
groups:
  watcher:
    healthcheck:
      test:
        - CMD-SHELL
        - ip addr show dev eth1 | grep "$IP_ADDR"
      start-period: 15
      interval: 5
      timeout: 3
      retries: 20
```

This check ensures that the container: - Has started successfully. - Has assigned the expected IP address.

Only after this validation is successful the dependent nodes are allowed to continue with the startup process

---

Startup Stages

Startup stages define deployment dependencies between nodes.

Example:

```
groups:
  switches:
    stages:
      create:
        wait-for:
          - node: logwatch
            stage: healthy
```

This specific configuration ensures:

1. The `logwatch` node must start first.
2. Containerlab waits until the healthcheck succeeds.
3. Only then are switches and other infrastructure nodes created.

This prevents scenarios where the network nodes are up before the monitoring node is ready.

Startup Order in the VNTD Topology

The topology follows this startup order:

Stage	Components
1	Monitoring node (logwatch)
2	Core infrastructure (firewall and switches)
3	Routers
4	End hosts

This ordering ensures that monitoring and infrastructure services are fully operational before user traffic is generated.

YAML Anchors

To reduce duplication in the topology definition file, the project uses YAML anchors. These allow defining reusable configuration templates that can later be referenced multiple times.

Example:

```
pc_template: &user_pc
  kind: linux
  group: hosts
  image: alpine_vntd:latest
  env:
    DHCP_CLIENT: 1
```

Nodes can later inherit the configuration using:

```
pc_vlan50_1:
  <<: *user_pc
  binds:
    - ./config/pc/mutt/enterprise/alice:/root/.mutt/muttrc
```

> **Anchor overwrite value**

When a node uses an anchor, its values / settings can be overwritten by simply setting a new value in the already defined parameter.

This technique provides the benefits of: - Reduced duplicated configuration. - Simplified maintenance. - Consistency across multiple nodes.

Why Anchors Instead of Groups or Kinds

Although Containerlab provides **groups** and **kinds** support, they serve different purposes and cannot be fully customized.

Groups

Allow assigning common runtime behaviour such as startup stages and health checks.

However, a node can belong to **only one group**.

Therefore, groups are mainly used for **deployment management**. These are not intended for creating templates.

Devices are assigned to a specific group:

Group	Description	Nodes
watcher	Threat detection and Logs	`logwatch`
switches	Enterprise core devices	`switch_*`, `firewall`
routers	All routers available	`router_*`
hosts	Servers and users connected	`*_server`, `attacker`, `benign`, `pc_admin`, `pc_*`

Kinds

Kinds represent predefined node types supported by Containerlab, such as: - `linux` - `arista_ceos`

These are **hard-coded platform defined** and cannot easily be extended with common user configurations.

Therefore, they are not suitable for reusable topology templates.

YAML Anchors

YAML anchors provide a **configuration reuse mechanism** independent from Containerlab.

They allow: - Defining templates for common nodes - Keeping topology file compact - Avoiding inconsistencies between nodes

Therefore, the project uses anchors for reusable cofigurations, groups for startup orchestration and stages for startup order definition.

Topology Startup Flow

In complex environments, the order in which nodes are initialized can significantly affect the environment. Some nodes depend on components that must already be operational.

In the provided enterprise topology, the monitoring system must be ready before traffic starts. Therefore, it must be the first node to be available, and the rest should start progressively.

To handle these dependencies, the topology defines a **structured startup flow** using Containerlab orchestration configurations.

Deployment

The startup sequence implemented in the topology is:

```
flowchart TD

A[Logwatch Node<br>Monitoring Stack] --> B[Firewall]
B --> C[Layer 2 Switches]
C --> D[Enterprise Routers]
D --> E[Servers & Client Devices]
```

The monitoring device is started first so it can begin analyzing traffic immediately. Next, the connectivity nodes are launched to bring up the core infrastructure and allow the monitoring node to start receiving cloned messages. Finally, the end nodes are started once the core nodes start to become operational.

This sequence ensures that the monitoring node can capture and process as many logs as possible.

Additional Resources

Here are some additional resources that may help create and manage custom topology definition files:

- Containerlab Supported Kinds: Containerlab
- Containerlab Topology Definition: Containerlab
- Containerlab Nodes: Containerlab
- Containerlab Lab Directory and Configuration Artifacts: Containerlab

March 11, 2026

8.3 Lab Configuration Guidelines

While the topology file defines the *structure* of the network, the behavior of the individual nodes is determined by their configuration files. In this project, we decouple configuration from the container images to allow for quick response and flexibility.

This document outlines how to configure the different types of nodes available in the environment, using the provided Docker images as a baseline.

1. Linux Routers (FRRouting / Custom Image)

Nodes using the `frr_vntd` or `router_vntd` images are powered by either **FRRouting (FRR)** or a custom image with routing services installed. Configuration is managed via file mounts rather than interacting with the shell manually.

Required Files

- **`/etc/frr/daemons`**: Controls which routing protocols are enabled. You must explicitly set these up like `ospfd=yes` or `bgpd=yes`.
- **`/etc/frr/frr.conf`**: The configuration file containing interface IP addresses, routing policies, and protocol definitions.

Best Practices

- Mount a centralized `daemons` file if all routers use the same protocols.
- Keep separate `frr.conf` files for each router in your `config/` directory (e.g., `config/router/internet/frr.conf`).

---

2. Linux Firewalls & Servers (Custom Images)

Generic Linux nodes (Debian, Alpine) often require more advanced setup sequences to give a dedicated image the designated capabilities (e.g., setting up `iptables`, creating bridges, or starting services).

The Startup Script Pattern

Instead of hardcoding commands in the `exec` section of the topology file, we use a `startup.sh` scripts mounted to the container.

Topology Definition

```
firewall:
  kind: linux
  image: firewall_vntd:latest
  binds:
    - ./config/firewall/my-lab/startup.sh:/startup.sh
  exec:
    - sh /startup.sh
```

Script Content `startup.sh`

- **IP Addressing:** `ip addr add ...`
- **Interface Management:** Creating bridges (`ip link add type bridge`) or VLAN sub-interfaces (`ip link add link eth0 name eth0.10 type vlan id 10`).
- **System Toggles:** Enabling IP forwarding (`sysctl -w net.ipv4.ip_forward=1`).
- **Security Rules:** Defining `iptables` or `nftables` policies.

3. Arista cEOS Switches

Arista cEOS (Containerized EOS) nodes emulate enterprise-grade switching. They consume standard EOS CLI configuration files.

> **Arista Limits**
>
> Arista cEOS switches do not support assigning access lists to either VLANs orports. This means such switches cannot be used as a Multi Layer Switch (L3). Hence, the provided `_mls` images.

Configuration Injection

Containerlab supports a native `startup-config` parameter for Arista nodes. This applies the configuration immediately upon boot.

Topology Definition:

```
switch_access:
  kind: arista_ceos
  image: ceos_vntd:latest
  startup-config: ./config/switch/access_layer.cli
```

Configuration File (`.cli`)

The file should contain standard Arista CLI commands.

Example Switch Config (.cli)

```
hostname switch-access
!
vlan 10
   name Chosen_Name
!
interface Ethernet1
   switchport mode access
   switchport access vlan 10
!
```

4. Configuration Persistence Strategy

To ensure your lab remains reproducible and easy to manage, follow these guidelines:

1. **Do not modify containers manually:** Avoid running `docker exec` to change configurations interactively, as these changes are lost when the lab is destroyed.
2. **Use the `config/` directory:** Store all `frr.conf`, `startup.sh`, and `.cli` files in a structured directory tree within your project (e.g., `config/<node_type>/<location_node>/`).
3. **Bind Mounts:** Always map the local files to the expected paths inside the container using the `binds` section of the topology file.

By following this pattern, you can switch between completely different network behaviors (e.g., OSPF vs. BGP) simply by changing the mount paths in your topology file, without rebuilding the Docker image.

February 21, 2026

9. Scripts & Automation

9.1 Scripts & Automation Overview

To minimize human errors and automate the workflow, the project includes a comprehensive set of automation scripts. These tools manage the entire lifecycle of the laboratory, from building custom Docker images to orchestrating complex network topologies with Containerlab

Core Philosophy

The automation framework is built upon three fundamental principles:

Centralized Control

A single entry point (`run.sh`) provides access to every management function, eliminating the need to memorize complex CLI flags.

Naming Conventions

The system automatically enforces project-specific tags (using the `_vntd` suffix) to prevent conflicts with other local Docker resources.

Interactive Guidance

User-friendly interfaces that guide the operator through operations by removing the needing of memorize complex flags or instructions.

Main Entry Point: `run.sh`

The `run.sh` script, located at the project root, serves as the main entry point for interacting with the environment. It acts as a wrapper that delegates tasks to specialized scripts in the `scripts/` directory.

Execution

To launch the project management menu, execute:

```
./run.sh
```

Permissions

Ensure the script is executable before the first run: chmod +x run.sh

Navigation

The automation logic is divided into two modules:

- **Lab Management**: Orchestrates Containerlab deployments and monitors active topologies.
- **Image Management**: Automates Docker build, import, and cleanup operations.

February 21, 2026

9.2 Image Management Scripts

Located in `scripts/images/`, these scripts automate the **Docker** lifecycle build process of the custom Docker images required for the laboratory infrastructure. These are necessary to ensure all nodes in the topology run the correct customized software versions defined in the project.

Interactive Controller: `menu.sh`

Role: The controller for all Docker operations. **Input:** Project root directory. **Behavior:** Displays options to create, import, delete, or display images and calls the corresponding script based on user input. **Command:**

```
./menu.sh /path/to/project/dir/virtual-network-threat-detection
```

Operational Workflow

Building Custom Images (`create.sh`)

Role: Builds custom Docker images from source. **Input:** Project directory. **Workflow:** 1. **Scanning:** Iterates through subdirectories in `docker/build/`. 2. **Filter:** Skips any directory starting with an underscore (`_`). This allows for "draft" folders to exist without breaking or building unnecessary images. 3. **Tagging:** Automatically tags the built image as `<directory_name>_vntd`. !!! example A folder named `kali` results in an image named `kali_vntd`. 4. **Build:** Executes `docker build -t <image_name> <directory>`. **Command:**

```
./create.sh /path/to/project/dir/virtual-network-threat-detection
```

Importing External Images (`import.sh`)

Role: Imports vendor-supplied images (e.g., Arista cEOS). **Input:** Project directory. **Workflow:** 1. **Scanning:** Scans `docker/import/` for .tar.xz archives. 2. **Tagging:** Extracts the base name before the first hyphen (`-`) and converts it to lowercase, and appends `_vntd` at the end. !!! example `cEOS-lab-4.32.0F.tar.xz` → imported as `ceos_vntd`. 3. **Import:** Executes `docker import <file> <image_name>`. **Command:**

```
./import.sh /path/to/project/dir/virtual-network-threat-detection
```

Delete Images (`images/delete.sh`)

Role: Cleans up the created Docker images. **Input:** None required. **Workflow:** 1. **Filter:** Filters for all images with a name ending in `_vntd`. 2. **Delete:** Executes `docker rmi -f <image_name>` to force removal. 3. **Safety:** Only affects images with the project suffix, leaving other system images intact. **Command:**

```
./delete.sh
```

Display Available Images (`images/display.sh`)

Role: Lists project-specific images. **Input:** None required. **Behavior:** Filters docker images using `*_vntd` as reference and displays image details. **Command:**

```
./display.sh
```

Technical Summary

Function	Command Path	Target Directory
Create	`scripts/images/create.sh`	`docker/build/`
Import	`scripts/images/import.sh`	`docker/import/`
Delete	`scripts/images/delete.sh`	Local Registry
Display	`scripts/images/display.sh`	Local Registry

Build Optimization

The build script uses the local Docker cache. If you modify a specific configuration file inside the build directory, only the affected elements will be rebuilt during the next `Create` process.

February 21, 2026

9.3 Lab Management Scripts

Located in `scripts/clab/`, these **Containerlab** scripts provide a text-based interface to ensure safer and more convenient user experience. They handle automatic topology discovery and status checking.

Interactive Controller: `menu.sh`

Role: The controller for all topology operations. **Input:** Project root directory. **Behavior:** Presents options to deploy, destroy, or display topologies and calls the corresponding script based on user input. **Command:**

```
./menu.sh /path/to/project/dir/virtual-network-threat-detection
```

Core Operations

Topology Deployment (`deploy.sh`)

Role: Deploys a network topology. **Input:** Project `lab/` directory. **Workflow:** 1. **Scanning:** Looks for `*.clab.yml` files in the `labs/` directory. 2. **Status Check:** Warns if other topologies are currently running to prevent excessive resource consumption. 3. **Selection:** Prompts the user to select a lab from a numbered list. 4. **Execution:** Invokes `clab deploy --topo <selected_file>`. **Command:**

```
./deploy.sh /path/to/project/dir/virtual-network-threat-detection/labs
```

Environment Destruction (`destroy.sh`)

Role: Destroys a running topology. **Input:** Project `lab/` directory. **Workflow:** 1. **Status Check:** Uses `clab inspect` to verify if any labs are active and, if not, stops the process. 2. **Scanning:** Looks for `*.clab.yml` files in the `labs/` directory. 3. **Selection:** Prompts the user to select a lab from a numbered list. 3. **Execution:** Invokes `clab destroy -t <selected_file>`. This stops containers and removes virtual bridges. **Command:**

```
./destroy.sh /path/to/project/dir/virtual-network-threat-detection/labs
```

Visibility and Inspection (`display.sh`)

Role: Provides environment state visibility. **Input:** Project `lab/` directory. **Behavior:** Lists all `*.clab.yml` files found in the `labs/` folder and displays all active topologies with their respective nodes. **Command:**

```
./display.sh /path/to/project/dir/virtual-network-threat-detection/labs
```

> **Persistence Practise**
>
> Avoid storing permanent configuration files inside the `clab-<name>` directories generated during deployment. Use the project's `config/` directory and binds instead.

Technical Summary

Function	Command Path	Target Directory
Deploy	`scripts/clab/deploy.sh`	`lab/`
Destroy	`scripts/clab/destroy.sh`	`lab/`
Display	`scripts/clab/display.sh`	`lab/`

Free Resources

After using the environment, stop it to ensure no unnecessary resources are consumed.

February 21, 2026

10. Network Services

10.1 Network Services Overview

The **Network Services** section documents the core infrastructure services that support basic network operations within the VNTD laboratory.

These services provide fundamental functionality such as address assignment and name resolution, and are required for user endpoints and internal systems to communicate correctly across the architecture and interact with the environment.

Scope

This section focuses on **infrastructure-level services** rather than application-level systems, specifically on:

- How services are architected
- How they interact with VLANs, routing, and firewall policies
- How service traffic flows across zones

Service behavior is described from a **network perspective**, not from an application or performance view.

Available Services

DHCP (Dynamic Host Configuration Protocol)

Provides dynamic IP addressing, default gateways, and DNS settings to users across multiple VLANs. Explore DHCP Design

DNS (Domain Name System)

Enables hostname resolution for both internal enterprise resources and external internet simulations. Explore DNS Design

Application services

Application-level services (HTTP, SSH, etc.) are documented separately.

Relationship with Architecture

These services are tightly coupled with the overall architecture. While the **DHCP Server** resides in the Internal Services zone (VLAN 40) and the **DNS Server** in the DMZ (VLAN 10), they must be acccessible by the users through the central firewall.

For context on VLAN placement, routing logic, and security enforcement, refer to:

- Architecture Overview
- Traffic Flows

Application Services

Application-level services like HTTP, SSH, FTP, and MAIL are documented separately in the Services section.

February 21, 2026

10.2 DHCP

DHCP Service Design

DHCP is a **core infrastructure service** that enables scalable client connectivity while maintaining strict VLAN isolation and centralized control.

Purpose of DHCP in the Lab

The DHCP service is responsible for:

- Dynamically assigning IP configurations to user endpoints.
- Distributing network parameters consistently.
- Reducing manual configuration on client nodes.
- Simulating real enterprise endpoint behavior.

DHCP is intentionally limited to **user-facing VLANs** and is not used for infrastructure or service nodes.

Centralized DHCP Architecture

The lab implements a **centralized DHCP server model**, where:

- A single DHCP server runs in VLAN 40.
- Multiple user VLANs are served by that server.
- All DHCP traffic traverses the firewall.

This approach reflects common enterprise deployments and simplifies management, logging, and policy enforcement.

TRAFFIC FLOW LOGIC

```
flowchart LR
    Client -->|DHCPDISCOVER| Relay
    Relay -->|Unicast| DHCP_Server
    DHCP_Server -->|DHCPOFFER| Relay
    Relay -->|Broadcast| Client

    Client[Client VLAN 50/60] -->|Broadcast| Relay[Firewall Relay]
    Relay -->|Unicast| Server[DHCP Server VLAN 40]
    Server -->|Unicast| Relay
    Relay -->|Broadcast| Client
```

Architectural Constraints

The DHCP design is constrained by the following architectural choices:

- **Isolation:** User devices (VLAN 50/60) cannot communicate directly with the server (VLAN 40) but through the firewall.
- **Enforcement:** All DHCP traffic must traverse the firewall, where security policies can be applied.
- **Scope:** DHCP is reserved for user workstations; infrastructure nodes (routers, servers) use static assignments to ensure stability.

Because DHCP relies on broadcast-based discovery, a direct client–server model is not viable.

DESIGN BENEFITS

This design provides several advantages:

- **Scalability:** New user VLANs can be added without deploying new DHCP servers.
- **Security:** DHCP traffic is explicitly limited to a specific set of VLANs.
- **Maintainability:** All address pools are defined in a single location.
- **Realism:** Mirrors enterprise DHCP deployments.

> **Scalability**
>
> The design makes it easy to add new devices to the domains without modifying the DHCP server logic.

ROLE OF THE FIREWALL

The firewall plays a critical role in the DHCP architecture:

- Acts as the default gateway for all VLANs.
- Hosts the DHCP **relay agent**.
- Enforces security policies on DHCP traffic.

The firewall forwards all DHCP traffic between the allowed VLANs requesting addresses and the DHCP server.

February 21, 2026

DHCP Server Configuration

This document details the **configuration and behavior** of the provided DHCP server used in the VNTD lab.

The DHCP server is hosted on the `internal_server` node within the **Internal Services (VLAN 40)** zone.

Service Activation

The service is powered by the **ISC DHCP Server** and runs on a dedicated internal service node. This placement ensures controlled access and avoids exposure to other zones or undesired external networks.

To make use of the service included in the `server_vntd` container, proceed with the following steps:

1. TOPOLOGY

Set as environment variables (`env`) the following elements in the desired container using the `server_vntd` image:

- DHCP_SERVER - Enable the service (any value other rather than 1 prevents the service from starting).
- IFACE - Selected interface.
- IP_ADDR - Address assigned to the interface.

> **Consideration**
>
> These values do **not configure the device** or the DHCP service; **manual file configuration still needs to be done** if the user wants to change the service configuration. Instead, they are used for the `entrypoint.sh` script so it waits until the interface and address are assigned to initialize the service. If the service starts before the interface or address is assigned, the service won't start.

```
env:
    DHCP_SERVER: 1
    IFACE: "eth1"
    IP_ADDR: "192.168.40.10"
```

2. CONFIGURATION FILES

Bind both configuration files required for the service to work:

- `dhcpd.conf` - Address pools and service configuration.
- `isc-dhcp-server` - Selected interface to operate on.

```
binds:
    - ./config/server/dhcp/dhcpd.conf:/etc/dhcp/dhcpd.conf
    - ./config/server/dhcp/isc-dhcp-server:/etc/default/isc-dhcp-server
```

DHCP Software

The service is implemented using **`isc-dhcp-server`**. The server is to be explicitly bound to a single interface: `eth1`. This prevents the DHCP daemon from listening on unintended interfaces (even though in this scenario, all `internal_server` traffic is routed through the same interface).

Configuration is defined in:

- `/etc/default/isc-dhcp-server`
- `/etc/dhcp/dhcpd.conf`

All DHCP configuration files are stored under the project `config/server/dhcp` directory and mounted into the container at runtime.

Persistence

Changes made directly inside the running container are **not persistent** and will be lost on redeployment.

Address Definitions

The server defines dedicated pools for each user VLAN in the `dhcpd.conf` file:

VLAN	Subnet	IP Range	Gateway
40	192.168.40.0/24	- (Declared only)	192.168.40.1
50	192.168.50.0/24	192.168.50.10 - .254	192.168.50.1
60	192.168.60.0/24	192.168.60.10 - .254	192.168.60.1

Topology Declaration

It is necessary to declare the address of the network the DHCP device belongs to, even if it does not offer any service; otherwise, the service will not work.rt.

Clients receive as a part of their configurations the assignment of a default gateway and a DNS server address. This avoids manual client configuration and centralized configuration for ease of management.

Configuration example

```
subnet 192.168.50.0 netmask 255.255.255.0 {
    range 192.168.50.10 192.168.50.254;
    option routers 192.168.50.1;
    option subnet-mask 255.255.255.0;
    option broadcast-address 192.168.50.255;
}
```

Configuration example - Topology Declaration

```
subnet 192.168.40.0 netmask 255.255.255.0 {
}
```

The DHCP can also be configured to provide a DNS address to the clients by making use of the following:

Configuration example - Provide DNS

```
option domain-name "enterprise.local";
option domain-name-servers 192.168.10.10, 192.168.40.10;
```

Global Configuration

DNS settings can be applied globally, regardless of the VLAN receiving the same DNS parameters.

February 21, 2026

DHCP Relay Configuration

The **DHCP Relay** mechanism is essential for providing dynamic addressing in a segmented network where clients and the server reside in different Layer 3 domains. This mechanism allows forwarding all DHCP communication between zones. Without a relay, DHCP would fail by design.

The DHCP relay service is hosted on the `firewall` node.

Service Activation

The service is powered by the **ISC DHCP Relay** and runs on the dedicated firewall node. As it runs on the **Central Firewall**, it serves as the default gateway for all VLANs and has direct interfaces in every segment, handling responses and forwarding DHCP traffic between clients and the server.

Considering it already enforces inter-VLAN policies, makes the firewall the optimal location for DHCP relaying.

To utilize the DHCP Relay service included in the `firewall_vntd` container, proceed with the following steps:

1. TOPOLOGY

Set as environment variables (`env`) the following elements in the desired container using the `firewall_vntd` image:

- DHCP_RELAY - Enable the service (any value other rather than 1 prevents the service from starting).

> **Consideration**
>
> This setting also establishes the necessary firewall rules to allow DHCP traffic between VLANs.

```
env:
    DHCP_RELAY: 1
```

2. CONFIGURATION FILES

Bind both configuration files required for the service to work:

- `startup.sh` - Required to start and set all firewall rules.
- `isc-dhcp-relay` - Interfaces to offer service and the interface connected with the DHCP server.

```
binds:
    - ./config/firewall/enterprise/startup.sh:/startup.sh
    - ./config/firewall/enterprise/isc-dhcp-relay.conf:/etc/default/isc-dhcp-relay
exec:
    - sh /startup.sh
```

> **Startup Script**
>
> The startup script **always** needs to be executed on the Firewall device to ensure traffic rules, VLANs and policies are enforced.

DHCP Relay Software

The service is implemented using **`isc-dhcp-relay`**. The service has definesthe definition of where the requests should be sent and which interfaces to listen on. Such are defined like:

```
SERVERS="192.168.40.10"              # 1
INTERFACES="br-vlan50 br-vlan60 eth5" # 2
```

1. The IP address of the central DHCP server in VLAN 40.
2. The internal bridges for user floors and the interface connected to the services VLAN.

Security Policies

The firewall must permit UDP traffic on ports 67 and 68 for the relay process to work. The `startup.sh` script includes rules to allow the firewall to process incoming petitions from user subnets and communicate with the server.

The relay does not ignore firewall filtering. All DHCP traffic is subject to all policies enforced.

Inspection

To debug DHCP transactions, use the following command on the firewall:

```
tcpdump -i <interface> -f 'port 67 or port 68'
```

February 21, 2026

DHCP Client Behavior

Clients in the VNTD environment are designed to simulate realistic enterprise endpoints using lightweight **Alpine Linux** images. These are designed to be minimal, automated, and representative of real enterprise endpoints.

Client Scope

DHCP clients include:

- User workstations in VLAN 50.
- User workstations in VLAN 60.

Service Activation

The service is powered by **DHCP Client** and runs on all `alpine_vntd` machines.

To utilize the DHCP client included in the end users devices, proceed with the following steps:

1. TOPOLOGY

Set as environment variables (`env`) the following elements in the desired container using the `alpine_vntd` image:

- DHCP_CLIENT - Starts the interface and waits for an address to be assigned on that specified interface.

> **Interface**
>
> The interface chosen must be the same as the one used on the topology definition file to connect with other devices.

```
env:
    DHCP_CLIENT: 1
    IFACE: "eth1"
```

2. CONFIGURATION FILES

The only necessary bind required for the device to make use of the DHCP service is:

- `startup.sh` - Wakes up the interface, waits for an address to be assigned.

```
binds:
    - ./config/pc/startup.sh:/startup.sh
exec:
    - sh /startup.sh
```

Automated Startup

Each client workstation (e.g., `pc_vlan50_1`) executes the `startup.sh` script upon launch to automate the network configuration process.

The script performs:

- Interface activation.
- DHCP request for IPv4 configuration.
- Automatic application of routing and DNS settings.

This ensures that clients are fully operational without manual intervention.

Waiting for the DHCP process

If the DHCP provider doesn't start and offer service on time, the devices may stop asking. To solve this, simply make the device wait until the device obtains an address.

```
# from labs/config/pc/startup.sh

while true; do
    if dhcpcd -4 -w "$IFACE"; then
        break
    fi
    sleep 5
done
```

Applied Parameters

Once the ip assignment process is complete, the client automatically receives and applies:

- **IP Address and Mask**.
- **Default Gateway:** Points to the Firewall interface (e.g., `192.168.50.1`).
- **DNS Resolver:** In this case it points to the DMZ Server (`192.168.10.10`).

Network Dependency

Client connectivity depends on: - Correct firewall relay configuration. - Active DHCP server. - Allowed DHCP traffic in firewall policies. If any of these components fail, clients will not obtain network access.

Troubleshooting

If a client fails to obtain an IP, verify:

1. Connectivity between the clients and the server.
2. The firewall node has `DHCP_RELAY` enabled.
3. The `internal_server` is running and the DHCP service is active.
4. The L2 bridges on the firewall (`br-vlan50`, `br-vlan60`) are up and STP is disabled.

Check IP

Clients can inspect their state using: `ifconfig` This tool is useful for understanding applied configuration.

February 21, 2026

10.3 DNS

DNS Service Design

TNS is a critical infrastructure service that provides hostname resolution for internal enterprise services and simulated external internet resources while respecting network segmentation and security boundaries.

Purpose of DNS in the Lab

The DNS service is responsible for:

- Resolving hostnames for internal enterprise services.
- Supporting user access to DMZ-hosted resources.
- Forwarding unresolved queries to external DNS servers.
- Simulating realistic enterprise name resolution behavior.

DNS is required for both usability and realism, as most applications rely on name resolution rather than raw IP addresses.

Hybrid DNS Model

The DNS design needs to consider these constraints:

- VLANs are isolated at Layer 3.
- All DNS traffic must traverse the firewall.
- All DNS requests are managed by the `Internal Server`.
- External resolution is managed by the `Interent Server`.

These ensure visibility and control over all name resolution activity.

Therefore, the lab employs a **hybrid resolution model** to ensure both local efficiency and realistic internet behavior:

- **Authoritative resolution:** The internal server resolves local domains like `enterprise.local`.
- **Recursive forwarding:** Requests for unknown domains are forwarded to the simulated internet resolver.

This allows internal services to be resolved, managed and optimized locally while still enabling Internet access.

Placement Strategy

DNS services are deployed in two distinct locations:

Resolver	Node	Zone	Role
Internal DNS	`dmz_server`	DMZ (VLAN 10)	Primary resolver for enterprise users.
External DNS	`internet_server`	ISP Core	Simulated public internet DNS.

This placement reflects a common enterprise pattern where:

- DNS is reachable by internal users.
- DNS can communicate with other external resolvers.
- Exposure is controlled through firewall policies.

The DNS service within the enterprise is not directly accessible from the Internet. However, the external one can be accessed by the internal one.

Integration with DHCP

DNS and DHCP are intentionally coupled:

- DHCP provides the DNS server address to clients (DMZ Server).
- Clients do not configure resolvers manually.
- DNS behavior is consistent across all user VLANs.

> **DHCP Coupling**
>
> This dependency is intentional and mirrors common enterprise environments.

February 21, 2026

DNS Server Configuration

Both the DMZ and Internet servers use **dnsmasq** to provide lightweight and efficient name resolution. The documentation focuses on clarity, explicit behavior, and ease of debugging.

Usage

To make use of the DNS service included in the `server_vntd` container, proceed with the following steps:

1. Set as environment variables (`env`) the following elements in the desired container using the `server_vntd` image:

- DNS_SERVER - Enable the DNS service (any value other rather than 1 prevents the service from starting).

```
env:
    DNS_SERVER: 1
```

1. Bind the configuration file required for the service to work:

- `dnsmasq.conf` - DNS configuration, name resolution, address assigned.

> **Note**
>
> The DNS server from the enterprise uses a different `.conf` file adapted to its needs. The one for the internet server provides a '*simpler*' format.

```
binds:
    - ./config/server/dns/enterprise/dnsmasq.conf:/etc/dnsmasq.conf
```

Service Placement

The DNS server runs on two specific devices:

- **Internal Server**
- **Node:** `dmz_server`
- **VLAN:** 10
- **IP Address:** `192.168.10.10`
- **Internet Server**
- **Node:** `internet_server`
- **VLAN:** Does not belong to any VLAN
- **IP Address:** `172.16.100.100`

Such nodes may also host other services, but the DNS is isolated from others.

DNS Software

The service is implemented using:

- **`dnsmasq`**

The server is to be bound to a specific port: `53` . Most configuration documentation can be found within the same `.conf` file through comments.

All DNS configuration files are stored under the project `config/` directory and mounted into the container at runtime.

> **Warning**
>
> Changes made inside the running container are **not persistent** and will be lost on redeployment.

DNS Configuration (`internet_server`)

Names Defined: Resolves names to addresses.

Name	Address
`internet.com`	172.16.100.100
`www.internet.com`	172.16.100.100
`enterprise.com`	172.16.30.2
`www.enterprise.com`	172.16.30.2

Reverse DNS (PTR Defined): Resolves addresses to names.

Address	Name
172.16.100.100	`internet.com`
172.16.30.2	`enterprise.com`

This configuration allows clients to communicate with the Internet Server and the DMZ Enterprise Server.

> **Note**
>
> Although the 172.16.30.2 address belongs to the router, all traffic is redirected to the DMZ server.

Forwarding: The server does not forward DNS requests to any outside network. If the name or address is unknown, the service won't be able to respond.

DNS Configuration (`dmz_server`)

Names Defined: Resolves names to addresses.

Name	Address
`enterprise.local`	192.168.10.10
`www.enterprise.local`	192.168.10.10
`enterprise.com`	192.168.10.10
`www.enterprise.com`	192.168.10.10
`internal.enterprise.local`	192.168.40.10

The DMZ Server also contains name resolution for the `.com` pointing to itself. This allows users to connect to the DMZ just as external users/clients would but removes the need to leave the network.

Reverse DNS (PTR Defined): Resolves addresses to names.

Address	Name
192.168.10.10	`enterprise.local`
192.168.10.10	`enterprise.com`
192.168.40.10	`internal.enterprise.local`

This configuration allows clients to communicate with the Internet Server and the DMZ Enterprise Server (and the latter without leaving the network).

Forwarding: Given the situation where the server does *not* know the name or address, a request to the DNS located on the internet will be made in order to attempt to recover the requested address.

server=172.16.100.100

From the internal network, through the **firewall's** point of view, only the server can communicate with any external DNS service.

February 21, 2026

DNS Resolution Flow

This page describes how **DNS queries** traverse the VNTD network based on the source and the requested domain.

Internal Client Flow (VLAN 50/60)

When an internal workstation attempts to resolve a name, the following process occurs:

1. **Client:** Consults `/etc/resolv.conf` (updated automatically by DHCP) and sends a query to `192.168.10.10`.
2. **Firewall:** Forwards the DNS packet (UDP/53) from the user VLAN to the DMZ.
3. **DMZ Server:**
   - If the name is known by the server, it responds directly.
   - If the name is **NOT** known by the server,, it forwards the query to `172.16.100.100`.
4. **Firewall & Router:** Perform NAT/Routing to reach the Internet Core.
5. **ISP Server:** Responds to the query.

Client Scope

DNS clients can be any device with access to a DNS server. Devices in the network need to either manually add the DNS address or receive it through DHCP.

DNS Manual Assignment - Attacker - Benign - Administrator

DHCP Assignment - PC VLAN 50 1 & 2 - PC VLAN 50 1 & 2

DNS address assignment does not rely on a working DNS server. Assignment can be made nonetheless, but the service won't work.

Manual Assignment

The easiest way to manage and make a network node use a DNS server is to directly assign it to them through the `topology` file.

DNS manual assignment

```
dns:
    servers:
        - 172.16.100.100
```

This way, addresses can be easily managed and modified without resorting to scripting or startup files.

DHCP Assignment

For devices receiving service from a DHCP server, the way to go is to wait for the DHCP server to offer the already configured in within DNS addresses into the devices connectivity settings.

DHCP provides DNS

Given that the DHCP service is not working, not only won't these devices receive an IP address, but also a DNS address.

Check IP

Clients can inspect their state using: `ifconfig` This tool is useful for understanding applied configuration.

February 21, 2026

11. Services

11.1 Application Services

This section documents **application-level services** deployed within the laboratory.

Unlike core network services (DHCP, DNS), these services: - Operate at higher layers - Are workload-oriented - Primarily used to generate realistic traffic - Are intentionally simple and minimal.

Scope of Application Services

Application services in this project are used to:

- Simulate real client–server interactions
- Generate observable traffic patterns

> **Note**
>
> Services are deployed for realism and traffic generation, not functionality richness.

Purpose and Design

The services integrated into the environment are designed to be minimal and predictable, maximizing focus on security monitoring and traffic analysis.

WEB (HTTP)

Standard Nginx server providing basic web traffic on Port 80. View Details

SSH (Secure Shell)

Remote access simulation for administrative tasks and credential abuse scenarios. View Details

FTP (File Transfer)

Unencrypted file management via Port 21, featuring user isolation and chroot jails. View Details

MAIL (SMTP/IMAP)

Realistic hub-and-spoke infrastructure for enterprise email exchange. View Details

Design Principles

Application services follow these principles:

All application services follow these core technical requirements:

- **Minimal Configuration:** Optimized to use few resources while maintaining fidelity.
- **Predictability:** Clearly defined traffic patterns for easier rule creation.
- **Transparency:** Security features like TLS are intentionally disabled to facilitate clear traffic inspection and learning.
- **Decoupled Logic:** Most services are hosted on the `server_vntd` image and enabled via environment variables.

February 21, 2026

11.2 Web Service

The web service implements **basic HTTP traffic** simulation with minimal configuration. It is primarily used to test web-based detection, logging, and traffic classification.

Technical Implementation

The service utilizes **Nginx**, which is pre-installed in the server image.

Activation

The service is enabled only by setting the following environment variable in the topology file:

```
env:
    WEB_SERVER: 1
```

If this variable is absent or has any other value different than 1, the Nginx process will not initialize.

> **Activation Through Variables**
>
> This allows a single container image to serve multiple roles depending on runtime configuration.

Deployment Status

In the default provided topology, the web service is active *(on the containers using the `server_vntd` image)*:

- `dmz_server` : Internal organization website.
- `internet_server` : External public website simulation.

Served Content

When enabled, the service creates a single static page:

```
Hello from Nginx on the web server
```

This content is written directly to:

```
/var/www/html/index.nginx-debian.html
```

There are: - No dynamic pages - No authentication - No application logic

Web Behavior

The web server is started using the system service manager:

```
service nginx start
```

Nginx runs in the background using default settings.

> **Traffic within the enterprise**
>
> Web traffic uses standard TCP port 80 and must traverse firewall and routing policies to reach the destination floor.

Info

Due to its simplicity, this service is ideal for testing HTTP-based detection, logging, and traffic classification.

How to use

To generate HTTP traffic against the web service, use a simple HTTP client such as `curl` frp, any client workstation or external node:

```
curl http://<hostname>
curl <hostname>
```

I

IP addresses can also be used if the domain names are unknown or a DNS service is not available. By default, all servers have a DNS-resolvable hostname.

If the service is running correctly, the service will respond with a message.

The available hostnames are defined in the DNS configuration: - DNS Names Assignment

February 21, 2026

11.3 SSH Service

The SSH service provides a mechanism for executing remote commands and simulate administration connections. It is intentionally configured with weak security settings to support credential abuse and brute-force experimentation.

Implementation Details

The service is based on **OpenSSH Server** and is managed via environment variables in the container entrypoint file.

Activation

The SSH service is enabled only if the following environment variable is set:

```
env:
    SSH_SERVER: 1
```

If this variable is absent or has any other value different than 1, the Nginx process will not initialize.

> **Note**
>
> This allows the same container image to be reused with or without SSH enabled.

Deployment Status

In the default provided topology, the SSH service is active *(on the containers using the `server_vntd` image)*:

- `dmz_server` : Internal organization website.
- `internet_server` : External public website simulation.
- `internal_server` : Interenal organization service provider.

Default Credentials

When the SSH service is enabled, a dedicated user account is created automatically.

Parameter	Value
Username	`vntd`
Password	`pswd`

These credentials are defined directly in the startup script.

> **Credentials**
>
> These credentials are intentionally weak. They exist solely for lab and testing purposes.

User and Authentication Configuration

At startup, the following actions are performed:

1. A new user (`vntd`) is created.
2. A password is assigned using chpasswd.
3. SSH is explicitly configured to:

- Allow password authentication.
- Allow root login.

Additionally, a user-specific (`vntd`) SSH configuration block is appended to ensure password access for the created user.

SSH Behavior

Once configured, the SSH service is started using the system service manager:

```
service ssh start
```

SSH traffic: - Uses TCP port 22. - Traverses firewall and routing policies.

> **Security**
>
> Repeated failed SSH attempts are useful for simulating brute-force or credential abuse scenarios.

How to use

The Alpine client images include a SSH Client, which is used to interact with the SSH service. To connect to the SSH service, use the following command:

```
ssh vntd@<name or address>
```

> **IP**
>
> IP addresses can be used if the domain names are unknown or a DNS service is not available. By default, all servers have a DNS-resolvable hostname.

When prompted, enter the default password:

```
pswd
```

The available hostnames are defined in the DNS configuration: - DNS Names Assignment

February 21, 2026

11.4 FTP Service

This document describes the **FTP service implementation** used in the laboratory.

The FTP service is provided by **vsftpd** and is enabled conditionally at container startup.

Service Activation

The FTP service is enabled only if the following environment variable is set:

```
env:
    FTP_SERVER: 1
```

If the variable is not set (or set to any other value), the SSH service is not started.

Additionally, the containerlab element requires the following attribute:

```
runtime: docker
```

If this attribute is not provided, there is a risk that the FTP service may break not only the container using it, but also other containers.

It is therefore strongly recommended to include it.

> **Reusability**
>
> This allows the same server image to be reused with or without FTP enabled.

User Creation

FTP users are created automatically at startup based on a configuration file.

Source file

Users are defined in:

```
/labs/config/server/ftp/vsftpd.chroot_list
```

Example content:

```
user5_1
user5_2
user6_1
user6_2
userAdmin
```

Rules: - Empty lines are ignored - Lines starting with `#` are ignored - Each valid line represents one FTP user

---

Default Credentials

All FTP users share the same default password.

Parameter	Value
Password	`pswd`

Security concerns

These credentials are intentionally weak. They exist solely for lab and testing purposes.

Home Directories and Isolation

For each FTP user: - A dedicated home directory is created:

```
/ftp/<username>
```

- The user is chrooted to this directory - Users cannot access other users' files - Shell access is disabled (`/sbin/nologin`)

Permissions: - User has full control over their own directory - No access outside their chroot

Info

This setup mirrors a simple multi-user internal FTP service.

FTP Behaviour

Once configured, the FTP service is started using the system service manager:

```
service vsftpd start
```

FTP traffic: - Uses TCP port 21 - Traverses firewall and routing policies

Observation

FTP traffic on TCP port 21 is unencrypted. This allows students to capture and analyze valid FTP commands and credentials in transit.

How to use

The Alpine client images include lftp, which is used to interact with the FTP service.

Connecting:

```
lftp user5_1@internal.enterprise.local # The IP address can also be used
```

When prompted, enter the default password:

```
pswd
```

Once connected, the user is placed directly in their home directory.

The available hostnames are defined in the DNS configuration: - DNS Names Assignment

Info

From the provided topology, the FTP service is available only on the `internal_server` using the `server_vntd` image. By default, all servers have a DNS-resolvable hostname.

Uploading Files:

Create a file locally:

```
echo "hello ftp" > test.txt
```

Upload it to the server:

```
put test.txt
```

Downloading Files:

List files on the server:

```
ls
```

Download a file:

```
get test.txt
```

Creating Directories:

```
mkdir docs
```

Deleting Files or Directories:

Delete a file:

```
rm test.txt
```

Delete a directory:

```
rmdir docs
```

Modifying Files:

Files can be overwritten by re-uploading them:

```
put test.txt
```

Exiting the Session:

```
exit
```

February 21, 2026

11.5 Mail Service

This document describes the **Mail service implementation** used in the laboratory.

The mail infrastructure is composed of: - **Postfix:** SMTP server (mail transfer). - **Dovecot:** IMAP server (mail retrieval). - **Mutt:** Mail client (end-user interaction).

The same `server_vntd` image is reused to implement: - The **Enterprise Mail Server (DMZ)**. - The **Main Internet Mail Server**.

Architecture

The mail architecture simulates a realistic **hub-and-spoke deployment**: - The **Enterprise Mail Server (DMZ)** handles internal users. - The **Internet Mail Server** acts as the public authority SMTP. - Enterprise SMTP forwards outbound mail to the Internet server. - The Internet server discards unknown destinations.

```
sequenceDiagram
    participant PC as User Workstation (VLAN 50)
    participant DMZ as Enterprise Mail (DMZ)
    participant ISP as Internet Mail (Public)

    Note over PC, ISP: Outbound Email
    PC->>DMZ: SMTP Outbound (Port 25)
    DMZ->>ISP: Forwarding to Central Hub

    Note over ISP, PC: Inbound Email
    ISP->>DMZ: Forwarding via Port Redirection
    DMZ->>DMZ: Store in User Maildir
    PC->>DMZ: IMAP Retrieval (Port 143)
```

Traffic flow:

```
Internal Client -> DMZ Mail Server -> Internet Mail Server -> Destination
```

Inbound traffic:

```
Internet Mail Server → Router (port 25 forward) → DMZ Mail Server
```

Traffic whose destination is the user's same mail service provider does not get redirected.

> **Security rules**
>
> From the Internet towards the Enterprise network, the router forwards **TCP port 25** traffic to the `dmz_server`.

Service Activation

The mail services are enabled only **one** of the following environment variables is set:

```
env:
    VAR_NAME: 1
```

VAR_NAME	Purpose
MAIL_SERVER	Enables Enterprise mail server (spoke)
MAIN_MAIL_SERVER	Enables Internet main mail server (central hub)

If neither variable is set: - Postfix does not start. - Dovecot does not start.

File Structure and Configuration

Mail configuration is injected using **bind mounts** defined in the topology.

SMTP (Postfix)

Enterprise SMTP File:

```
./config/server/smtp/enterprise/main.cf
```

Purpose: - Defines SMTP behavior for Enterprise domain. - Forwards outbound mail to the Internet server. - Defines relay host and domain policies.

Internet SMTP (Main Server) Files:

```
./config/server/smtp/internet/main.cf
./config/server/smtp/internet/transport
./config/server/smtp/internet/relay_recipients
```

File	Purpose
`main.cf`	Core configuration
`transport`	Defines domain routing rules
`relay_recipients`	Specifies valid recipient addresses

When `MAIN_MAIL_SERVER` is set, the entrypoint: - Adjusts file permissions and ownership. - Runs `postmap` on transport and relay files.

> **Unknown Destination**
>
> Unknown mail addresses are discarded.

Postfix is initialized automatically based on the chosen configuration.

IMAP (Dovecot)

Files:

```
10-auth.conf
10-mail.conf
auth-system.conf.ext
dovecot.conf
```

Purpose: - Authentication configuration. - Maildir storage definition. - System user authentication. - Global Dovecot behavior.

Dovecot provides: - IMAP on port 143. - Maildir mailbox access.

> **IMAP vs POP3**
>
> POP3 is intentionally not installed.

Mail Users Directory

All mail users are defined through bind:

```
./config/pc/mutt -> /mail-users:ro # Read Only
```

Structure:

```
/<directory>/
├── enterprise/
│   ├── alice
│   ├── clark
│   └── ...
└── internet/
    └── olivia
```

Each file represents a user account.

At startup: - The entrypoint scans the directory. - Creates system users. - Sets passwords equal to usernames (weak, but easy to manage). - Creates Maildir. - Sets ownership. - Disables shell access.

> **Bind user files**
>
> User files are bound to avoid creating a new file containing only the users to be created by each server.

User Creation

Users are automatically created using:

```
useradd -m -s /sbin/nologin <user>
```

For each user: - Home directory created. - Maildir initialized. - Password = username. - IMAP access allowed. - Shell access disabled.

This ensures isolation between users and realistic mail behavior in a simplified lab management.

Network Behavior

Service	Port	Protocol
SMTP	25	TCP
IMAP	143	TCP

TLS is **not** enabled. This allows clear: - Traffic inspection. - IDS monitoring. - Clear-text credential observation.

From the router, inbound port 25 traffic from Internet is forwarded to `dmz_server`.

Mail Client (Mutt)

End-user systems use **Mutt** as the mail client. Each PC binds a specific configuration file:

Example:

```
./config/pc/mutt/enterprise/clark -> /root/.mutt/muttrc
```

Example configuration:

```
set from = "clark@enterprise.com"
set realname = "Clark Enterprise"
set use_from = yes

# IMAP
set folder = "imap://clark@192.168.10.10:143/"
set imap_user = "clark"
set imap_pass = "clark"
set spoolfile = "+INBOX"

# SMTP
set smtp_url = "smtp://clark@192.168.10.10/"
set smtp_pass = "clark"

# Editor
#set editor = "vim"
set charset = UTF-8

# SSL
set ssl_starttls = no
set ssl_force_tls = no
```

Each PC can simulate a different user depending on which bind is used.

This design allows: - Simulating multiple employees. - Credential exposure. - Clear and simple mail flow testing.

Using the Mail Service

From any client, write on the terminal:

```
mutt
```

> **Mandatory Connectivity**
>
> The device **must** have connection to the mail provider. Otherwise, the automatic sign up process will fail. For DHCP users, wait for a moment or check if an address has been assigned using `ifconfig`.

> **Refresh Mutt**
>
> The `mutt` interface is simple and does **not** detect new emails. To receive new emails, the user needs to exit the program and access it again.

Security considerations

This setup is intentionally weak and uses on purpose: - Weak passwords. - No TLS. - Plain-text authentication.

This is required for: - Clear and easy traffic inspection. - IDS pattern detection.

> **Lab Isolation**
>
> This configuration must **never** be used in production environments. This is designed for training and must remain isolated to prevent the laboratory from acting as a real relay for external spam or malicious mail.

February 21, 2026

12. Monitoring

12.1 Monitoring & Threat Detection

The VNTD environment integrates a centralized monitoring system responsible for processing and visualizing network events across the simulated enterprise network.

The monitoring architecture relies on a **single node called** `logwatch`, which integrates several industry-standard open-source tools.

Monitoring Pipeline

```
flowchart LR

Traffic[Mirrored Traffic] --> Suricata

Suricata --> Logs[EVE JSON Logs]

Logs --> Filebeat

Filebeat --> Elasticsearch

Elasticsearch --> Kibana
```

The monitoring pipeline follows an architecture designed around logs, where traffic is inspected, transformed into structured events, indexed, and finally visualized.

```
graph LR
    A[Traffic Source] -->|iptables TEE| B[Suricata IDS]
    B -->|JSON Logs| C[Filebeat]
    C -->|Ingest| D[Elasticsearch]
    D -->|Query| E[Kibana Dashboard]

    subgraph "Detection Service"
    B
        subgraph "Log Processing"
        C
        D
        E
        end
    end
```

Ass seen, the node is divided into two primary functional areas: - IDS that inspects mirrored traffic from the firewall and stores it into registers using the **Suricata** service. - Centralized logging platform consisting of **Filebeat**, **Elasticsearch** and **Kibana**.

Data Flow

The monitoring process follows several stages:

1. **Traffic Analysis:** Network traffic is mirrored towards the `logwatch`. **Suricata** analyzes packets in real time.
2. **Event Generation:** Suricata generates structured **EVE JSON logs** describing network events.
3. **Log Shipping: Filebeat** monitors these log files and forwards them to the storage backend.
4. **Indexing: Elasticsearch** stores and indexes events, making them searchable.
5. **Visualization: Kibana** provides dashboards and search interfaces for analysts.

Web Interfaces

Some services expose web interfaces accessible from the host system.

Service	Access
Kibana	`http://<ip_address>:5601`
Elasticsearch	`http://<ip_address>:9200`

It's recommended to use a web explorer to access both sites, using the IP address generated by *Containerlab* on setup.

Architecture Overview

The monitoring stack deployed in the `logwatch` node consists of four major components:

- **Suricata**: Network packet inspection.
- **Elasticsearch**: Log storage and indexing.
- **Kibana**: Visualization and data exploration.
- **Filebeat**: Log forwarding and parsing.

March 11, 2026

12.2 Suricata

Suricata is the **Network Intrusion Detection System** used within the VNTD laboratory.

The primary purpose of this engine is to record **all** incoming and outgoing network traffic for later processing.

Within the project architecture, Suricata acts as the **first stage of the monitoring stack**, generating structured network events that are later processed by the rest of the monitoring stack. Therefore, it operates as a **passive sensor**.

It receives mirrored traffic from the firewall and analyzes it without interfering with other network operations.

Architecture

The monitoring architecture relies on **traffic mirroring** from the firewall towards the monitoring node.

The firewall duplicates traffic using `iptables TEE` and sends the duplicated packages to the monitoring interface of the `logwatch` container.

```
flowchart LR

    CLIENT[Client Devices]
    FW[Firewall]
    IDS[Logwatch Node<br>Suricata IDS]

    CLIENT --> FW
    FW -->|Normal Forwarding| INTERNET[Internet]
    FW -->|Traffic Mirror| IDS
```

Passive Monitoring

The IDS operates strictly in passive mode, this means: - No packet modification. - No packets forwarded by the IDS.

> **Promiscuous Mode**
>
> The monitoring interface needs to be set to promiscuous mode to capture all mirrored traffic received from the firewall.

> **Monitoring Only**
>
> The IDS is not configured to block traffic. It operates only as a detection and logging system.

Service Activation

Suricata is started only when the following environment variables are defined:

```
env:
    SURICATA_SERVICE: 1
```

If the variable is not set Suricata does not start and therefore no traffic inspection occurs.

This mechanism allows the monitoring stack to be enabled or disabled easily depending on the lab scenario.

Additional Environment Variables

Additional environment variables are required to start the machine correctly.

```
    IFACE: "eth1"
    IP_ADDR: "192.168.20.10/24"
    IP_GTWY: "192.168.20.1"
```

Although the interface variable is not required to be set (by default, it is set to `eth1`), other variables are necessary to establish a connection with the network topology.

Startup Behavior

When the container starts, the entrypoint script performs the following steps:

1. Waits until the monitoring interface exists.
2. Assigns the configured IP address.
3. Enables promiscuous mode.
4. Launches Suricata in background mode.

Example command executed:

```
suricata -c /etc/suricata/suricata.yaml -i eth1
```

Parameter	Purpose
-c	Specifies configuration file
-i	Interface used for packet capture

File Structure and Configuration

Suricata configuration is provided through binds.

Primary configuration file:

```
/etc/suricata/suricata.yaml
```

This file defines: - Capture interface configuration and capture settings. - Detection settings. - Logging outputs. - App layer protocol configuration. - Other advanced settings...

Bind example:

```
binds:
  - ./config/logwatch/suricata/suricata.yaml:/etc/suricata/suricata.yaml
```

This allows modifying the Suricata service behavior without rebuilding the container image.

Rules

The rules define the logic used to identify / store network traffic to generate logs accordingly.

> **Rule Updates**
>
> In this laboratory, environment rules are included with the configuration file. Any changes applied to the file will require a service (or environment) restart.

One of the main advantages Suricata offers is its ability to configure the service to detect or ignore traffic from specific protocols, allowing users to only process concrete traffic.

> **Monitoring Health**
>
> To verify if Suricata is correctly receiving packets, you can check the statistics log: `tail -f /var/log/suricata/stats.log`.

Log Generation

Suricata generates several log files located in:

```
/var/log/suricata/
```

The most relevant feature of Suricata in this lab is the file output. All data is recovered and logged into a single estructured JSON file:

`eve.json`

Considering the security events are stored in a structured JSON format, it offers seamless integration with the Elastic stack.

Other generated files:

File	Purpose
`fast.log`	Human readable alert logs
`suricata.log`	Engine diagnostic logs, useful to check if the service has started correctly and any possible errors
`stats.log`	Performance metrics, resource utilitzation, packet statistics, engine performance

Using Suricata

Suricata runs automatically once the monitoring container starts.

To inspect generated events manually:

```
tail -f /var/log/suricata/eve.json
```

> **JSON**
>
> It's recommended to use formatting tools such as `jq` to ease the process of reading JSON files.

Lab Security Warning

The IDS configuration is designed for **educational and testing environments**. It prioritizes: - Simplicity. - Transparency. - Easy inspection of network traffic.

> **Not for Production**
>
> The configuration used in this laboratory environment should not be deployed in production networks.

Additional Resources

For additional documentation, check the official documentation:

- Docs - Suricata

March 11, 2026

12.3 Elasticsearch

Elasticsearch acts as the **central log storage and indexing engine** for the monitoring infrastructure.

It receives structured events from **Filebeat**, stores them internally as documents and allows efficient querying and visualization through **Kibana**.

Therefore, *Elasticsearch* is responsible for: - Central service for storing data. - Monitoring stack core service for communication between all services. - Indexing and analyzing data.

Architecture

Elasticsearch is responsible for **data storage and indexing** within the monitoring pipeline.

```
flowchart LR

    FILEBEAT[Filebeat]
    ELASTIC[Elasticsearch]
    KIBANA[Kibana]

    FILEBEAT --> ELASTIC
    ELASTIC --> KIBANA
```

Elasticsearch acts as the core service for all Elastic components. It receives data and provides it for further analysis and processing.

Service Activation

Elasticsearch is started only when the following variable is set:

```
env:
    ELASTIC_STACK: 1
```

If the variable is not set, none of the Elastic components (Elasticsearch, Filebeat, Kibana) start.

This allows disabling the full monitoring stack when running the laboratory on lower resource systems.

Startup Behaviour

The monitoring container `entrypoint` performs the followign operations:

1. Adjust kernel parameters recommended to run Elasticsearch.

```
sysctl -w vm.max_map_count=262144
```

!!! note "Memory Requirement" Although not necessary to set such parameter, Elastic official documentation suggests may help the service run smoother.

2. Start the Elasticsearch service.
3. Wait for the API service to be up and running.
4. Configures security users and roles.

Example startup command:

```
    su -s /bin/bash elasticsearch -c "/usr/share/elasticsearch/bin/elasticsearch" &
```

The service must be executed using the **elasticsearch** user. This profile is automatically created by the elastic installer and needs no additional manual configuration to use. Hence, the use of `su` (switch user) and the `-s` (use bash as the shell) and `-c` (which operation to execute) parameters.

File Structure and Configuration

Elasticsearch configuration is provided through binds.

Elasticsearch primary configuration file:

```
/etc/elasticsearch/elasticsearch.yml
```

This file defines: - Service behaviour. - Security settings. - Node roles. - Network parameters.

Bind example:

```
binds:
  - ./config/logwatch/elasticsearch/elasticsearch.yml:/etc/elasticsearch/elasticsearch.yml
  - ./config/logwatch/elasticsearch/heap.options:/etc/elasticsearch/jvm.options.d/heap.options
```

See heap.options file in the next point

This allows modifying the Elasticsearch service behavior without rebuilding the container image.

Heap Options

Elasticsearch consumes a significant amount of RAM. By default, it allocates aroung **50%** of the total memory to Elasticsearch.

Although it's often recommended to allocate **16GB** to the service, many users cannot afford to dedicate so many resources.

Therefore, the amount of RAM consumed by the service can be limited in the `heap.options` file.

```
/etc/elasticsearch/jvm.options.d/heap.options
```

This file defines the amount of RAM allocated to the Elasticsearch JVM. In this laboratory environment, the RAM consumption has been limited to **2GB**, although it can be further reduced to **1GB** given the scale of the environment.

```
-Xms2g
-Xmx2g
```

- `Xms` - initial heap size.
- `Xmx` - maximum heap size.

For best performance, both values should be set to the **same value**.

Data Storage

Elasticsearch stores its data inside the directory:

```
/var/lib/elasticsearch
```

> **No persistence**
>
> Given it's a Containerlab environment, data persistence is not possible since the environment is designed to be cleaned at every execution.

Security Configuration

During container initialization the entrypoint automatically configures several elements.

Elastic Superuser

Default system user:

elastic

The password is automatically generated during installation and displayed on the terminal. Since obtaining this value may be difficult in an automated environment or may need to be updated, the following tool is used:

```
RESET_OUTPUT=$(/usr/share/elasticsearch/bin/elasticsearch-reset-password -u elastic -b)
```

Then, from the given response, it's extracted and stored during the entrypoint execution for configuring the environment and creating additional users.

The password is random and cannot be specifically set.

> **Filebeat**
>
> The password is later used by Filebeat to set up its service.

Kibana System User

User:

kibana_system : pswd_vntd

This user allows Kibana to communicate with Elasticsearch. This user is already provided by the Elasticsearch service and only needs to have a new password.

Filebeat Internal User

A dedicated user is created for Filebeat:

filebeat_internal : pswd_vntd

With the aassociated role created as well:

filebeat_writer

> **New User**
>
> The justification for this decision can be found at Filebeat.

Permissions included:

Permissions	Type	Purpose
monitor	cluster	Read cluster status
read_ilm	cluster	Read lifecycle policies
read_pipeline	cluster	Read ingest pipelines
manage_ilm	cluster	Manage lifecycle policies
manage	cluster	Cluster admin operations
all	cluster	Full cluster admin
auto_configure	index	Auto-create indices
create_doc	index	Insert new documents
write	index	Full document writes
create	index	Index new documents
create_index	index	Create indices
manage	index	Full index admin

This ensures Filebeat can send logs and properly communicate with Elasticsearch.

Administrator User

A new administrative user is created for the Kibana web login.

admin : 12345aA

This user has the **superuser role**.

> ⚠ **Weak credentials**
>
> These credentials are intentionally weak and should only be used in isolated and controlled environments.

API Access & Verification

The Elasticsearch service publishes a REST API.

Default port:

9200

Default endpoint:

http://localhost:9200

Example terminal request:

```
curl http://localhost:9200
```

This previous request can be also be used to check whether the environment is up and working or not.

The site can also be accessed from a web explorer.

Localhost -> IP Address

To connect to the web site for testing the environment and it's accessibility, use the Containerlab generated IP management address.

You can verify the health of the service directly from the terminal from the same machine:

```
curl -X GET "http://localhost:9200/_cluster/health?pretty"
```

Additional Resources

For additional documentation, check the official documentation:

- Docs - Elasticsearch

March 11, 2026

12.4 Filebeat

Filebeat acts as the **bridge between Suricata and Elasticsearch**.

Its function is to recover all generated logs by **Suricata** and send them directly to **Elassticsearch** for later examination and processing.

Therefore, *Filebeat* is responsible for: - Collecting Suricata logs. - Parsing structured events. - Sending data to Elasticsearch. - Loading dashboards and pipelines.

Architecture

Filebeat operates between the IDS and the data storage service.

```
flowchart LR

    SURICATA[Suricata IDS]
    FILEBEAT[Filebeat]
    ELASTIC[Elasticsearch]
    KIBANA[Kibana]

    SURICATA --> FILEBEAT
    FILEBEAT --> ELASTIC
    FILEBEAT --> KIBANA
```

Filebeat need to be able to connect with Elasticsearch to generate the data pipelines and send the data. But also with Filebeat to load pre-built dashboards, visualizations and templates.

Service Activation

Filebeat runs only when **both monitoring components are enabled**.

```
env:
    SURICATA_SERVICE: 1
    ELASTIC_STACK: 1
```

If either variable is missing or has a value different than 1, the service won't start.

This ensures Filebeat only runs when both Suricata and analysis tools (Elastic stack) are available.

File Structure and Configuration

Filebeat is configured via two primary files:

filebeat.yml **suricata.yml**

Main configuration file defines the output destination. Filebeat and Elasticsearch reside on the same `logwatch` node for simplified management.

Located in `modules.d/`, this file define which services the filebeat is supposed to recover the logs from.

Filebeat configuration file:

```
/etc/filebeat/filebeat.yml
```

This file defines: - Log input paths. - Elasticsearch output. - Modules enabled. (In this scenario activated through commands) - Kibana output.

Bind example:

```
binds:
  - ./config/logwatch/filebeat/filebeat.yml:/etc/filebeat/filebeat.yml
  - ./config/logwatch/filebeat/suricata.yml:/etc/filebeat/modules.d/suricata.yml
```

See modules file in the next point

This allows modifying the Filebeat service behavior without rebuilding the container image.

Configuration Permissions

Configuration files required to run the Filebeat environment requires their permissions to be adjusted.

Required permissions:

filebeat.yml - **Owner:** root - **Permissions:** 600

suricata.yml - **Owner:** root - **Permissions:** 644

Specially, the `filebeat.yml` file. The service refuses to start if the configuration file is writable by other users.

> **Permissions**
>
> Once the permissions are changed, the user will need to change these manually if he desires to apply changes to the file. Permissions 644 are more than enough to apply changes to the specified file.

Modules

Filebeat uses the **Suricata module**. Modules simplify data parsing and include:

- Predefined pipelines.
- Dashboards.
- Index templates.

The included module can be found at:

```
/etc/filebeat/modules.d/suricata.yml
```

This file defines the path from where the filebeat recovers the logs to send to Elasticsearch.

To see all available modules, use the command:

```
filebeat modules list
```

> **Modules support**
>
> All supported modules already include a template file within the `modules.d` directory.

The entrypoint automatically enables the module with:

```
filebeat modules enable suricata
```

This way, the configuration file needs no additional parameters.

Log Suricata Sources

Filebeat reads Suricata logs from:

```
/var/log/suricata/eve.json
```

This JSON stream contains all the logs recovered by suricata. Which are later parsed and sent structured to Elasticsearch.

Filebeat Setup

Before starting the service, Filebeat runs a setup procedure:

```
filebeat setup
```

This command loads several important resources into Elasticsearch:

Component	Purpose
Index templates	Defines data strutures
Ingest pipelines	Parse incoming logs
Dashboards	Visualization templates

In this laboratory, the setup also requires the Elasticsearch administrator credentials to properly create and set up all previous components.

```
filebeat setup \
-E output.elasticsearch.username=$ELASTIC_USER \
-E output.elasticsearch.password=$ELASTIC_PASSWORD
```

Once completed, filebeat runs on the defined user specified in the cofiguration file and created on the `entrypoint.sh` script.

Security Considerations

Filebeat requires credentials to communicate with Elasticsearch and be able to download dashboards and services from the official Elastic servers.

> **Official dashboards**
>
> Security and credentials can be removed/disabled, but doing will prevent the environment from downloading dashboards and other services.

For this reason, a **dedicated internal user** is created:

filebeat_internal : pswd_vntd

This user has enough permissions to manage and create all necessary elements to parse data into Elasticsearch.

> **Credentials**
>
> The credentials are intentionally weak for this laboratory environment to ease the configuration process.

> **Role + Permissions**
>
> All recommended permissions suggested by the official Elastic documentation plus additional ones recommended by online forums have been added to ensure propper communication between services. Please consider study into greater detail which privileges to give the user before copying these settings into a real environment.

Starting Filebeat

Once configuration and setup are completed, Filebeat is started with:

```
    filebeat -e -c /etc/filebeat/filebeat.yml &
```

These parameters allow: - **`-e`:** Logs errors to stderr. This way, logs can be seen using the `docker logs -f <container>` command. - **`-c`:** Specified the configuration file to use.

Once events reach Elasticsearch, they can be visualized in **Kibana**.

> **Remember**
>
> Dashboards are automatically installed during the `filebeat setup` phase, no need to create them manually.

Verification

Check Filebeat logs:

```
filebeat test output
```

Check whether the configuration is correct:

```
filebeat test config
```

Additional Resources

For additional documentation, check the official documentation:

- Docs - Filebeat

March 11, 2026

12.5 Kibana

Kibana provides the **browser-based interface** used to searching, viewing, and interacting with the security data stored in Elasticsearch.

It connects to **Elasticsearch** to visualize the data stored in it through dashboards and other graphical elements for easy analysis.

Therefore, *Kibana* is responsible for: - Search network events. - Inspect IDS registers. - Visualize dashboards. - investigate suspicious activity.

Architecture

Kibana works on top of **Elasticsearch** and provides visualization capabilities.

```
flowchart LR

    FILEBEAT[Filebeat]
    ELASTIC[Elasticsearch]
    KIBANA[Kibana]

    FILEBEAT --> KIBANA
    ELASTIC --> KIBANA
```

The system allows users to explore network activity captured by the IDS.

Service Activation

Kibana is started only when the following variable is set:

```
env:
    ELASTIC_STACK: 1
```

If the variable is not set, none of the Elastic components (Elasticsearch, Filebeat, Kibana) start.

This allows disabling the full monitoring stack when running the laboratory on lower resource systems

> ⚠ **Elasticsearch Dependency**
>
> Kibana cannot function without Elasticsearch. If the logwatch node reports a "yellow" or "red" status on it's health, the Kibana interface may be unavailable.

Startup Behaviour

The monitoring container `entrypoint` performs the followign operations:

1. Launch Kibana service.
2. Wait for Elasticsearch availability.
3. Wait until Kibana API returns a healthy status.

Example startup command:

```
    su -s /bin/bash kibana -c "/usr/share/kibana/bin/kibana" &
```

The service must be executed using the **kibana** user. This profile is automatically created by the elastic installer and needs no additional manual configuration to use. Hence, the use of `su` (switch user) and the `-s` (use bash as the shell) and `-c` (which operation to execute) parameters.

File Structure and Configuration

Kibana configuration is provided through binds.

Kibana primary configuration file:

```
/etc/kibana/kibana.yml
```

This file defines: - Elasticsearch source definition and credentials. - Network service properties. - Addresses to provide service to.

Bind example:

```
binds:
  - ./config/logwatch/kibana/kibana.yml:/etc/kibana/kibana.yml
  - ./config/logwatch/kibana/node.options:/etc/kibana/node.options
```

See node.options file in the next point

This allows modifying the Kibana service behavior without rebuilding the container image.

Heap Options

Kibana consumes a significant amount of RAM. By default, it allocates around **4GB** of the memory to Kibana.

Although it does not consume as many resources as its counterpart (Elasticsearch) it still consumes significantly.

Therefore, the amount of RAM consumed by the service can be limited in the `node.options` file.

```
/etc/kibana/node.options
```

This file defines the maximum amount of RAM for the Kibana web node. In this laboratory environment, the RAM consumption has been limited to **2GB**, although it can be further reduced to **1GB** given the scale of the environment.

```
--max-old-space-size=2048
```

The value is represented in `MB`.

Security Considerations

Kibana requires credentials to communicate with Elasticsearch and be able to request data to be displayed to the users.

> **Security**
>
> To view the reason why credentials have been enabled take a look at Filebeat.

A **dedicated internal user** is used. This user is already provided by the system.

> **Kibana user**
>
> For more information regarding this existing Kibana user, take a look at Elasticsearch.

kibana_system | pswd_vntd

> **Credentials**
>
> The credentials are intentionally weak for this laboratory environment to ease the configuration process.

Web Access & Verification

The Kibana service publishes a web interface.

Default port:

5601 #HTTP

Default endpoint:

http://localhost:5601

```
http://<monitoring-node-ip>:5601
```

The site is intended to be used from a web explorer.

> **monitoring-node-ip**
>
> To connect to the web site for testing the environment and it's accessibility, use the Containerlab generated IP management address.

Authentication

Users authenticate using credentials created during container initialization.

For more information on this process, check Elasticsearch.

Default credentials:

admin | 12345aA

This account has `superuser` privileges.

> **Weak credentials**
>
> These credentials are intentionally weak and should only be used in isolated and controlled environments.

Dashboards

Dashboards are automatically installed by **Filebeat** **setup**.

These dashboards include suricata statistics and easy register analysis.

> **Dashboards availability**
>
> Dashboards are only available if both suricata service and elastic stack are enabled. These are automatically created by **Filebeat**.

Additional Resources

For additional documentation, check the official documentation:

- Docs - Kibana

March 11, 2026

13. Attacks

13.1 Attack Simulations

This section documents the attack scenarios used to generate malicious traffic within the VNTD environment.

These simulations allow the monitoring infrastructure to generate real detection events, which can later be analyzed through the monitoring stack.

Objectives

Attack scenarios serve several purposes:

- Validate IDS detection capabilities.
- Generate labeled benign and malignant events
- Evaluate monitoring stack.
- Provide datasets for machine learning analysis.

Planned Scenarios

The following attack simulations will be implemented:

- Port Scanning.
- Denial of Service.
- ...

Each attack will include:

- Execution methodology.
- Expected network behavior.
- IDS detection results.
- Generated logs.

March 11, 2026

14. Operational Commands Reference

This document provides a repository of useful commands for managing, troubleshooting, and interacting with the nodes in the VNTD laboratory.

14.1 Container Management

These commands are executed from the **host machine** to control the virtual environment.

- **View Device Logs:** Check the output and initialization errors for a specific node:

```
docker logs clab-virtual-env-internal_server
```

- **Connect to a Device:** Open a terminal inside a running node:

```
docker exec -it clab-virtual-env-pc_vlan50_1 bash
```

14.2 Services

To view the services running on a concrete device, use:

```
service --status-all
```

```
ps aux
```

14.3 Traffic Inspection Tools

tcpdump

Use `tcpdump` to capture and analyze packets on a specific interface. This is critical for validating that the **DHCP Relay** or **Firewall** is forwarding traffic correctly.

- **Monitor DHCP Traffic:** In which `port` is the physical port to analyze. For this example, DHCP traffic can be viewed to detect any possible related issues given changes are made and the service is no longer working as intended.

```
tcpdump -i <port> -f 'port 67 or port 68'
```

14.4 Connectivity and Web Verification

Test if web services are reachable and serving content through the network policies.

Web Service Check

```
# Internal DMZ Website
curl http://enterprise.local

# External ISP Website
curl http://internet.com
```

Expected output: "Hello from Nginx on the web server"

14.5 DHCP Troubleshooting

To request an IP address again from a client device with access to the DHCP server:

```
dhcpcd -4 -d eth1
```

14.6 DNS Troubleshooting

Tools to verify the name resolution service between the enterprise network and the internet.

DNS Infrastructure

Zone	DNS Known IP Addresses
Internal DNS	192.168.10.10, 192.168.40.10
External DNS	172.16.100.100, 172.16.30.2

Resolution Diagnostic Tools

```
# Which DNS to query
cat /etc/resolv.conf
```

```
# Query DNS records for a specific name
nslookup enterprise.local
nslookup 192.168.10.10
```

```
# More structured and detailed output
dig www.enterprise.local
```

Example Output

```
pc_enterprise:/# nslookup enterprise.local
Server:     192.168.10.10
Address:    192.168.10.10#53

Name:   enterprise.local
Address: 192.168.10.10
```

```
pc_enterprise:/# nslookup enterprise.com
Server:     192.168.10.10
Address:    192.168.10.10#53

Name:   enterprise.com
Address: 192.168.10.10
```

```
pc_enterprise:/# nslookup internet.com
Server:     192.168.10.10
Address:    192.168.10.10#53

Name:   internet.com
Address: 172.16.100.100
```

14.7 Mutt

End-user systems use the Mutt mail client for sending and retrieving emails.

- **Launch:** Type in the terminal:

```
mutt
```

- **Editor Usage:** When setting up mail, use the built-in editor. To start writing use the `i` button. To save changes and exit the editor, use the command `:wq`.
- **Note:** The client does not detect new emails in real-time. You must exit and restart the program to refresh the inbox.

15. WORKING ON THIS:

tcpdump -i eth1 icmp -nn

Testing suricata configuration: suricata -T -c /etc/suricata/suricata.yaml

To see the suricata logs perfectly: docker logs -f clab-virtual-env-ids

View the logs data as it grows docker exec -it clab-virtual-env-ids tail -f /var/log/suricata/eve.json tail -f /var/log/suricata/eve.json

View the logs that only contain certain words: docker logs -f clab-virtual-env-ids 2>&1 | grep -Ei 'elastic|filebeat|elasticsearch|kibana'

Health: curl http://192.168.20.11:9200/_cluster/health?pretty

See machines usage cponsumption: docker stats

curl -s http://localhost:5066/stats

curl -s "http://192.168.20.11:9200/_cat/indices?v" curl -X GET "http://192.168.20.11:9200/_cat/indices?v" curl -X GET "http://192.168.20.11:9200/_cat/shards?v" curl 192.168.20.11:9200/_cat/indices?v curl 192.168.20.11:9200/_data_stream?pretty

All traffic: tcpdump -i eth1 -nn

Suricata logs: tail -f /var/log/suricata/suricata.log

docker stats

new token? /usr/share/elasticsearch/bin/elasticsearch-create-enrollment-token -s kibana

From suricata i'll need: Flow metadata Timing Bytes Ports Protocol Flags DNS queries HTTP hostnames TLS fingerprints

March 11, 2026