Let's Build The Internet In a Weekend

"Let's Build The Internet In a Weekend" is a hands-on guide to creating a fully virtualized Internet ecosystem, emulating real-world ISP infrastructure on a compact, scalable platform. Using Proxmox, VyOS, OPNsense, and a range of core services, this project showcases the setup of dynamic routing, DNS, DHCP, and live monitoring across multiple virtual ISPs—all within a self-contained environment. Combining practical skills with advanced technology, this book is a testament to what passion, expertise, and a DIY spirit can achieve, offering a detailed blueprint for anyone eager to master networking from the ground up.

Introduction

Introduction

Welcome to *Let's Build The Internet In a Weekend*, a journey into the heart of networking where we'll create a fully virtualized Internet environment from the ground up. This isn't just another lab setup or a simple home network—this is about building a complete digital ecosystem that mirrors the complexity and functionality of real-world ISPs, all within a controlled, virtual space. Throughout this guide, you'll see how to transform a collection of virtual machines into a robust, dynamic network that can handle core services, routing, monitoring, and much more.

Why undertake a project like this? The answer is simple: hands-on experience trumps theory every time. In the real world of IT, it's not just about knowing how things should work; it's about making them work, solving problems on the fly, and integrating diverse systems into a seamless whole. This project is a bold statement that you don't need a degree or a stack of certifications to understand and build complex networks. What you need is the willingness to dig in, experiment, and learn by doing. With over 20 years of experience in rural telecom and IT infrastructure, this book distills that practical knowledge into a weekend project that will leave you with a much deeper understanding of networking fundamentals and advanced configurations.

What You'll Learn

We'll take you through every step, from setting up Proxmox to configuring core routers, managing DNS and DHCP, and implementing real-time monitoring with tools like VyOS, OPNsense, and Grafana. By the end, you'll have created multiple virtual ISPs, each with its own set of core services and routing protocols. You'll understand the basics of dynamic routing, the intricacies of managing DHCP pools for different types of clients, and how to keep an eye on the whole setup with effective monitoring tools. This isn't just theory—it's real, practical knowledge you can use to build, troubleshoot, and optimize networks in the real world.

Who This Book Is For

This book is for anyone passionate about networking, whether you're a seasoned IT professional looking to sharpen your skills or a home lab enthusiast ready to take things to the next level. It's also for those who want to bypass traditional educational routes and prove what real-world skills can accomplish. If you're the kind of person who believes in the DIY ethos of IT, then this project will resonate with you. It's for those who prefer hands-on problem solving, who aren't afraid to get their hands dirty, and who see technology as a playground for creativity and innovation.

The Vision Behind the Project

This project is more than a technical exercise; it's a statement. It's about proving that practical, hands-on experience can accomplish what traditional paths might overlook. It's a celebration of the skills honed through years of working in the trenches of IT, managing everything from DHCP and RADIUS servers to complex routing and monitoring setups. The ecosystem you're about to build is a living demonstration of what's possible when you combine passion, knowledge, and a willingness to experiment. By documenting this process step-by-step, we're giving you not just a guide but a blueprint that you can adapt, modify, and use to build your own networks—whether at home, in a lab, or in a real-world IT environment.

Let's get started and build the internet, one virtual machine at a time.

Chapter 1: Getting Started

Section 1: Requirements

Recommended VM Specifications

Since we're not simulating the scale of a full-blown ISP, we can afford to keep the VMs relatively light:

1. **VyOS VM**:
   - **CPU**: 2 cores
   - **RAM**: 1 GB (2 GB recommended)
   - **Disk**: 8 GB
   - **Network**: 1 NIC connected to the real-world bridge, 1 NIC to the core network bridge, and additional NICs for each virtual ISP bridge.
2. **DNS Root VM**:
   - **CPU**: 1 core
   - **RAM**: 512 MB (1 GB recommended)
   - **Disk**: 4 GB
   - **Network**: 1 NIC connected to the core network bridge.
3. **Stratum 1 Time Server VM**:
   - **CPU**: 1 core
   - **RAM**: 512 MB (1 GB recommended)
   - **Disk**: 4 GB
   - **Network**: 1 NIC connected to the core network bridge.
   - *Hint*: There's something unique planned for this VM that will add a touch of realism to the time synchronization setup. Stay tuned.

Knowledge Prerequisites

To get the most out of this project, you should be comfortable with:

- **Basic Networking**: IP addressing, routing concepts, and network segmentation.
- **Linux Command Line**: Many configurations will require command-line interaction, especially when setting up services on VMs.
- **Virtualization**: Familiarity with creating and managing VMs within Proxmox.

Time and Resource Estimates

1. **Time Commitment**:
    *   **Initial Setup**: Approximately 1-2 hours for configuring Proxmox, setting up bridges, and creating initial VMs.
    *   **Configuration and Testing**: Expect to spend 3-4 hours on setting up core services and ensuring everything is communicating correctly.
    *   **Full Deployment**: Over the course of the weekend, you should be able to bring the entire virtual internet ecosystem to life.
2. **Resource Management Tips**:
    *   **Use Snapshots**: Take snapshots of your VMs at key configuration points. This allows you to roll back quickly if something goes wrong.
    *   **Monitor Resource Usage**: Keep an eye on CPU, RAM, and disk usage through Proxmox's interface. Adjust allocations as needed.

Conclusion

With these requirements in hand, you're ready to start building. This section ensures you have a clear understanding of what's needed and how each piece will fit together as we proceed. The next step is to lay out the network architecture, mapping out how our virtual ISPs will interact and how the core services will keep everything running smoothly. Let's get into the design phase!

Section 2: High-Level Network Architecture and Design

Before we start configuring VMs and services, it's essential to understand the overall architecture of the virtualized internet ecosystem we are building. This section provides a high-level view of how everything will be designed, including the key components, network topology, and security considerations. Think of it as the blueprint that will guide us through the technical setup in the following chapters.

1. Network Topology Overview

Our virtual internet ecosystem will mimic the real-world structure of an ISP network, but on a smaller, more manageable scale. At its core, the system will consist of:

- **Core Services**: These include our primary router, DNS root server, and Stratum 1 time server.
- **Network Bridges**: The connections between internal VMs, virtual ISPs, and the real world. Each service and ISP will have a dedicated segment, ensuring clean, isolated networking.
- **Virtual ISPs**: Independent network segments, each functioning as its own isolated ISP environment with its own routing, DHCP pools, and configurations.

Here's a breakdown of how the network will be organized:

- **Core Network (vmbr-core)**: The heart of our ecosystem, connecting all core services.
- **Real-World Bridge (vmbr0)**: Provides connectivity between our virtual environment and the external internet, allowing certain services to be accessible publicly.
- **ISP-Specific Bridges (vmbr-isp1, vmbr-isp2, etc.)**: Each virtual ISP will have its own bridge, creating isolated segments that can be routed and managed independently.

2. Key Components and Their Roles

1. **Core Router (VyOS)**

- The VyOS router will serve as the backbone of the network. It will handle:
    - **Dynamic Routing (BGP)**: Enabling communication between different segments of the network and simulating real-world routing behavior.
    - **Network Segmentation**: Isolating traffic between virtual ISPs, core services, and external networks.
    - **Security Policies**: Acting as a firewall and enforcing rules on how data moves between segments.

2. **DNS Root Server**
   - This VM will manage domain name resolution within the virtual ecosystem.
   - **Internal DNS Management**: Each virtual ISP will rely on this DNS root for resolving domain names, ensuring smooth communication within the network.
   - **Scalability**: Configured to handle both internal services and external DNS requests, simulating real-world DNS infrastructure.

3. **Stratum 1 Time Server**
   - A Stratum 1 server is a precise time source, usually directly connected to a GPS or atomic clock. In our setup, this VM will:
       - **Provide Accurate Time Synchronization**: Ensuring all services and VMs are in sync, which is crucial for network operations, security protocols, and troubleshooting.
       - **Hint**: There's a special feature we'll implement to give this server a realistic touch, making our virtual ecosystem's time synchronization as precise as possible.

4. **Root Certificate Authority**

- The Root Certificate Authority (CA) will serve as the central entity for issuing and managing digital certificates within our virtual ecosystem. It ensures that all internal services can communicate securely by providing trusted certificates for encryption protocols like HTTPS and TLS.
- **Purpose**: Setting up our own Root CA allows us to control the security infrastructure, enabling encrypted connections across the network. This is essential for protecting data, establishing trust between services, and mimicking real-world ISP security practices.

1. **Virtual ISPs**
   - Each virtual ISP will be an isolated environment, complete with its own:
       - **Routing and DHCP**: Independent configurations, simulating how different ISPs operate separately yet can be managed centrally.
       - **Custom Services**: DNS and DHCP servers for each ISP will be configured to allow for unique setups, providing a more realistic ISP simulation.

3. Bridges and Virtual Network Segmentation

1. **Core Network Bridge (vmbr-core)**

- Central to internal communication between core services, such as the VyOS router, DNS root server, and time server.
  - Acts as the backbone, ensuring all core components are tightly integrated and efficiently communicating.
2. **Real-World Bridge (vmbr0)**
   - This bridge connects the virtual environment to the external internet.
   - **External Access**: Necessary for functions like downloading software updates, public-facing monitoring, and allowing selected external services to interface with the internal network.
   - **Secure Routing**: Traffic between the real-world network and internal systems will be strictly managed via VyOS and OPNsense.
3. **ISP-Specific Bridges (vmbr-isp1, vmbr-isp2, etc.)**
   - Each virtual ISP gets its own bridge, creating isolated network segments.
   - **Isolation and Security**: This ensures that traffic is contained within its segment unless explicitly routed through the core network.
   - **Scalability**: Easy to add new ISP segments by creating additional bridges, enabling future expansion without disrupting the existing setup.

4. Security Considerations and Traffic Flow

1. **Traffic Isolation and Management**
   - **NAT and Firewalls**: VyOS will manage Network Address Translation (NAT) and firewall rules, ensuring that internal segments are secure and external access is controlled.
   - **Traffic Filtering**: Rules will be configured to allow or deny traffic between virtual ISPs, core services, and the external world based on defined security policies.
2. **OPNsense and HAProxy Integration**
   - **OPNsense**: Will function as a secondary layer of security, handling more advanced firewall and filtering tasks. It will also provide VPN services and further segment internal traffic.
   - **HAProxy**: Responsible for load balancing and managing traffic to and from the public-facing services hosted within the ecosystem, ensuring smooth, reliable access for monitoring tools like Grafana and Zabbix.

5. Visualization and Monitoring

1. **Internal and External Monitoring**
   - Tools like **Zabbix**, **Grafana**, and **OpenSearch** will be used to track key metrics across the network, including traffic flows, resource usage, and performance indicators.

- **Live Dashboards**: Accessible both internally and (selectively) from the external internet, giving a real-time view of the network's health and activity.
2. **Secure Public Access via NGINX Proxy**
   - **Traffic Routing**: Using **NGINX** at a remote server (e.g., Linode) to securely route external requests to the internal monitoring systems.
   - **Data Security**: Ensures that only specific data and metrics are exposed, while maintaining a strong security posture across the network.

Conclusion

This architecture lays out the groundwork for everything we're about to build. By understanding the high-level design and purpose of each component, you'll have a clearer path forward when it's time to configure the VMs, set up routing, and deploy services. The network topology and segmentation we've planned ensure scalability, security, and realistic simulation of real-world ISP environments. Now that we've covered what's needed and how it will all fit together, it's time to start building in Chapter 2.

We'll begin by setting up the core services, starting with the VyOS router, DNS root server, and our special Stratum 1 time server. This foundation will pave the way for the rest of the project, so let's get ready to dive in.

Raw Notes

These are the raw notes from building this project.

Internal DNS Configuration for isp1.net in a Private Network Environment

Objective

This guide describes the steps to configure Core DNS to forward all requests for **isp1.net** to **ISP DNS 1** within an isolated network. The goal is to set up an authoritative, internal-only DNS for **isp1.net**, ensuring local queries are resolved internally without reaching external DNS servers.

Requirements

- Core DNS as the main DNS resolver for internal clients
- ISP DNS 1 as authoritative for **isp1.net**
- Internal domain **isp1.net** that only resolves within the private network, avoiding external DNS lookups

Setup Steps

1. Define the Zone for isp1.net on ISP DNS 1

First, configure ISP DNS 1 to serve as the authoritative DNS for **isp1.net**.

1. Edit the ISP DNS 1 configuration file (typically located at `/etc/bind/named.conf.local`) to add the zone for **isp1.net**. Define the zone as a master and specify `allow-query` to any IP and `allow-transfer` permissions for Core DNS (10.1.0.10).

```
zone "isp1.net" {

    type master;

    file "/etc/bind/zones/db.isp1.net";

    allow-query { any; };

    allow-transfer { 10.1.0.10; };

};
```

Create the zone file, typically at `/etc/bind/zones/db.isp1.net`. This file should include the SOA record, NS record, and A records for all devices within **isp1.net**.

```
$TTL 86400
@   IN  SOA isp-dns1.isp1.net. admin.isp1.net. (
        2023102701 ; Serial
        3600      ; Refresh
        1800      ; Retry
        1209600   ; Expire
        86400     ; Minimum TTL
    )
; Nameserver
    IN  NS  isp-dns1.isp1.net.
; A records
isp-router1    IN  A   10.10.0.2
isp-dns1       IN  A   10.10.1.10
isp-gateway    IN  A   10.10.2.1
isp-business   IN  A   10.10.3.1
```

Restart BIND on ISP DNS 1 to apply the changes:

```
sudo systemctl restart bind9
```

2. Configure Core DNS to Use ISP DNS 1 for isp1.net

On Core DNS, define a **stub zone** for **isp1.net** that points to **ISP DNS 1** as the authoritative DNS server for this domain.

1. Add a stub zone entry for **isp1.net** to the Core DNS configuration file, typically located at `/etc/bind/named.conf.local`. In this entry, specify the type as `stub`, set the `masters` to ISP DNS 1's IP (10.10.1.10), and add a `forwarders` directive with empty braces to prevent

forwarding to external servers.

```
zone "isp1.net" {
    type stub;
    masters { 10.10.1.10; };
    forwarders {};  # Prevents external forwarding for isp1.net
};
```

- Explanation of the `forwarders {};` Directive: By setting `forwarders {};`, we stop Core DNS from forwarding requests for **isp1.net** to any external DNS servers. This directive is crucial to ensure Core DNS exclusively queries ISP DNS 1 for this internal-only domain.
- Restart BIND on Core DNS to load the new configuration:

```
sudo systemctl restart bind9
```

3. Verifying the Configuration

Use the following steps to confirm that the configuration is working correctly.

1. Run a direct query to ISP DNS 1 from Core DNS to confirm that ISP DNS 1 is serving the **isp1.net** records correctly:

```
dig @10.10.1.10 isp-router1.isp1.net
```

Test forwarding from Core DNS by querying **isp1.net** records without specifying ISP DNS 1, confirming that Core DNS is forwarding queries correctly to ISP DNS 1:

```
dig isp-router1.isp1.net @10.1.0.10
```

Use tcpdump or a similar tool to verify that DNS requests for **isp1.net** are reaching ISP DNS 1 and returning the expected responses:

```
sudo tcpdump -i eth0 host 10.10.1.10 and port 53
```

Troubleshooting and Common Issues

1. **REFUSED Errors**: If Core DNS receives REFUSED responses, ensure that ISP DNS 1 has `allow-query` and `allow-transfer` settings configured to allow access from Core DNS (10.1.0.10).
2. **allow-query-cache Denials**: If cache queries are denied, add `allow-query-cache { 10.1.0.10; localhost; };` to ISP DNS 1 to permit Core DNS to access cached entries for faster responses.

3. **No Matching 'Forwarders' Statement**: The `forwarders {};` directive is necessary in this configuration to prevent Core DNS from forwarding **isp1.net** queries to global DNS servers. Adding this directive in the stub zone settings ensures exclusive forwarding to ISP DNS 1.

Building the ISP Mail System with Roundcube

For a setup emulating around 100 users accessing a Roundcube webmail server, you'll want a balance of resources to handle concurrent webmail sessions, email delivery, and storage for messages. Here's a guideline for hardware resources to support this experimental environment effectively:

Recommended Hardware Resources

1. **CPU**:
   - **4-6 cores**: This will allow you to handle concurrent connections, especially as users access the webmail interface and the server handles SMTP/IMAP transactions.
   - Emulated users tend to put moderate stress on the CPU due to encryption (TLS/SSL for IMAP, SMTP) and web traffic, so having multiple cores will ensure responsiveness.
2. **Memory (RAM)**:
   - **8-12 GB RAM**: Roundcube itself is lightweight, but Dovecot and Postfix will need memory to handle simultaneous IMAP connections. Each user session can consume a small portion of RAM, especially with multiple concurrent mail retrievals.
   - Consider **16 GB** if you want extra room for scalability or plan to run additional monitoring tools.
3. **Storage**:
   - **200-500 GB SSD**: Storage needs can vary, but this range should be adequate for emails and attachments. SSDs are preferable to HDDs for faster I/O performance, which will help with message retrieval and database access.
   - **Database Storage**: Consider at least **50 GB** allocated for the MariaDB database, especially if you plan to retain a large amount of mail data.
4. **Network Bandwidth**:
   - **1 Gbps Network Interface**: Internal experiments typically don't need heavy outbound bandwidth, but a reliable internal network interface with 1 Gbps is sufficient for handling mail traffic and simultaneous IMAP sessions.
5. **Additional Considerations**:
   - **Load Testing and Monitoring Tools**: Include tools like **Grafana, Prometheus**, or simple **Linux system metrics** to monitor CPU, memory, and network usage as you emulate load. This can help you identify bottlenecks and tune the configuration as needed.

- **Virtual Machine Sizing**: If running in a VM, allocate resources as close to physical hardware specs as possible, avoiding overcommitted resources.

This setup will support your experiment with enough headroom to simulate user activity realistically. Let me know if you want further details on specific configurations for any of these components!

Kea DHCP Configuration Documentation

Overview

This configuration file is set up to provide DHCP services on a **10.10.1.0/24** network, specifically for an internal ISP user services network. It includes specific IP reservations, DNS settings, and comprehensive logging configurations to track DHCP activity at a detailed level.

Configuration Details

```
{
  "Dhcp4": {
    "interfaces-config": {
      "interfaces": [ "ens18" ]
    },
    "match-client-id": false,
    "subnet4": [
      {
        "subnet": "10.10.1.0/24",
        "pools": [],
        "option-data": [
          {
            "name": "domain-name-servers",
            "data": "10.10.1.10"
          },
          {
            "name": "domain-name",
            "data": "isp1.net"
          }
        ],
        "reservations": [
```

```
      {
        "hw-address": "BC:24:11:07:88:16",
        "ip-address": "10.10.1.100"
      }
    ]
  }
],
"loggers": [
  {
    "name": "kea-dhcp4",
    "severity": "DEBUG",
    "output_options": [
      {
        "output": "/var/log/kea/dhcp4.log",
        "maxver": 10
      }
    ]
  },
  {
    "name": "kea-dhcp4.dhcpsrv",
    "severity": "DEBUG",
    "output_options": [
      {
        "output": "/var/log/kea/dhcp4-dhcpsrv.log",
        "maxver": 10
      }
    ]
  },
  {
    "name": "kea-dhcp4.leases",
    "severity": "DEBUG",
    "output_options": [
      {
        "output": "/var/log/kea/dhcp4-leases.log",
        "maxver": 10
      }
    ]
  }
]
```

```
    }
  }
```

Explanation of Each Section

1. **"interfaces-config"**:
   - **Purpose**: Defines the network interfaces Kea should listen on for DHCP requests.
   - **Setting**: `"interfaces": ["ens18"]` binds Kea to `ens18`, the network interface for the **10.10.1.0/24** subnet.
2. **"match-client-id": false**:
   - **Purpose**: Instructs Kea to ignore the DHCP Client ID and rely on the hardware (MAC) address for client identification. This can be helpful if clients do not consistently provide a Client ID or if MAC-based reservations are used.
3. **"subnet4"**:
   - **Purpose**: Defines the subnet settings for the **10.10.1.0/24** network.
   - **Details**:
     - `"subnet": "10.10.1.0/24"` specifies the address range.
     - `"pools": []` indicates no dynamic IP pool is configured, meaning only reservations will be assigned.
     - `"option-data"` includes DHCP options:
       - `"domain-name-servers": "10.10.1.10"` specifies the DNS server.
       - `"domain-name": "isp1.net"` sets the DNS search domain for the network.
     - `"reservations"` contains static IP reservations:
       - **Example Reservation**: A device with MAC `BC:24:11:07:88:16` receives the reserved IP address `10.10.1.100`.
4. **"loggers"**:
   - **Purpose**: Enables detailed logging at the **DEBUG** level, splitting logs across different categories.
   - **Details**:
     - `"kea-dhcp4"`: General DHCP activity log file at `/var/log/kea/dhcp4.log`.
     - `"kea-dhcp4.dhcpsrv"`: DHCP server-specific events logged at `/var/log/kea/dhcp4-dhcpsrv.log`.
     - `"kea-dhcp4.leases"`: Lease assignment and release events logged at `/var/log/kea/dhcp4-leases.log`.
   - **Log Rotation**: Each log file has `maxver: 10`, allowing up to 10 rotated log files.

Usage Notes

- **Reservations**: This configuration only serves devices with reservations due to the empty pool configuration. Ensure all required devices have reservations.
- **Logging**: The **DEBUG** level will produce detailed logs useful for troubleshooting and monitoring DHCP activity.

This setup should provide reliable, reservation-based IP assignment with comprehensive logging for monitoring DHCP activity on the **10.10.1.0/24** network. Let me know if there are additional features or details you'd like to incorporate!