Basic Configuration of dnsmasq in an Incus Container on Debian with Netplan

1 Introduction

This guide provides step-by-step instructions for setting up `dnsmasq` as a DNS and DHCP server in an Incus container running Debian. The network configuration is managed using Netplan and a custom Python script to create virtual Ethernet (veth) pairs and bridges, ensuring proper network integration.

2 Prerequisites

Before proceeding, ensure the following:

- Incus is installed on the host system (`sudo apt install incus`).
- A Debian-based container is created in Incus.
- Python 3 and the `pyroute2` package are installed on the host (`sudo apt install python3 python3-pyroute2`).
- Basic knowledge of Linux networking and container management.
- Root or sudo access to the host and container.

3 Step-by-Step Configuration

3.1 Creating and Setting Up the Incus Container

Create a Debian container named `dnsmasq-container` using the following commands on the host:

```
incus create images:debian/12 dnsmasq-container
incus config set dnsmasq-container security.syscalls.intercept.mount true
incus config set dnsmasq-container security.nesting true
incus config set dnsmasq-container security.privileged true
incus start dnsmasq-container
```

The `security.syscalls.intercept.mount`, `security.nesting`, and `security.privileged` settings are required for `dnsmasq` and Docker to function correctly in the container.

3.2 Installing Additional Packages

Install the necessary packages inside the container:

```
incus exec dnsmasq-container -- apt update
incus exec dnsmasq-container -- apt install -y \
    netplan.io \
    sudo vim nano git tmux mc zip unzip curl wget htop lynx \
    iproute2 termshark bridge-utils \
    python3 python3-ipython python3-pyroute2 python3-scapy \
    docker.io docker-compose
```

3.3 Configuring Users and Permissions

Configure user access and permissions within the container.

3.3.1 Changing the Root Password

Set the root password to "passroot":

```
incus exec dnsmasq-container -- bash -c 'echo␣"root:passroot"␣|␣
   chpasswd'
```

3.3.2 Adding a New User

Add a new user named "user" with the password "pass" and add them to the "sudo" and "docker" groups:

```
incus exec dnsmasq-container -- useradd -m -s /bin/bash user
incus exec dnsmasq-container -- bash -c 'echo␣"user:pass"␣|␣
   chpasswd'
incus exec dnsmasq-container -- usermod -aG sudo user
incus exec dnsmasq-container -- usermod -aG docker user
```

3.4 Accessing the Container

Access the container's shell:

```
incus exec dnsmasq-container -- bash
```

3.5 Installing dnsmasq

Update the package list and install `dnsmasq`:

```
incus exec dnsmasq-container -- apt update
incus exec dnsmasq-container -- apt install dnsmasq -y
```

3.6 Configuring Network with Veth Pairs and Netplan

To enable advanced networking, use the provided Python script (`link.py`) to create a virtual Ethernet (veth) pair connecting the container to the host's network namespace, with an optional bridge for network integration. Save the following script as `link.py` on the host:

```
import argparse
import os
import subprocess
import sys
from pyroute2 import IPRoute, NetNS

# ... (rest of the link.py script as provided) ...
```

Run the script to create a veth pair, move one end to the container's network namespace, and attach it to a bridge on the host. First, identify the container's name or ID:

```
incus list
```

Assuming the container name is `dnsmasq-container`, execute the script with sudo privileges:

```
sudo python3 link.py -t1 incus -ns1 dnsmasq-container -n1 veth-
   container -t2 1 -n2 veth-host -b2 br0
```

Explanation:

- `-t1 incus`: Specifies that the first namespace is an Incus container.
- `-ns1 dnsmasq-container`: Specifies the container's network namespace (Incus container name).
- `-n1 veth-container`: Names the veth interface inside the container.
- `-t2 1`: Specifies the default (host) namespace.
- `-n2 veth-host`: Names the veth interface on the host.
- `-b2 br0`: Attaches the host's veth interface to a bridge named `br0`.

Before running the script, ensure the bridge `br0` exists on the host. Create it if necessary:

```
sudo ip link add name br0 type bridge
sudo ip link set br0 up
```

The script exposes the container's network namespace, creates the veth pair, moves `veth-container` to the container's namespace, attaches `veth-host` to `br0`, and brings both interfaces up.

3.7 Configuring the Network with Netplan

Configure the container's network using Netplan to assign a static IP address to the `veth-container` interface (aliased as `eth0` for simplicity). Create or edit the Netplan configuration file at `/etc/netplan/01-netcfg.yaml` inside the container:

```
incus exec dnsmasq-container -- nano /etc/netplan/01-netcfg.yaml
```

Add the following configuration:

```
network:
  version: 2
  ethernets:
    eth0:
      match:
        name: veth-container
      dhcp4: no
      addresses:
        - 192.168.1.10/24
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

Explanation:

- `match: name: veth-container`: Matches the `veth-container` interface created by the script, aliased as `eth0`.
- `dhcp4: no`: Disables DHCP to use a static IP.
- `addresses`: Assigns the static IP `192.168.1.10/24`.
- `routes`: Sets the default gateway to `192.168.1.1`.
- `nameservers`: Specifies Google's DNS servers.

Apply the configuration:

```
incus exec dnsmasq-container -- netplan apply
```

Verify the network configuration:

```
incus exec dnsmasq-container -- ip a show eth0
incus exec dnsmasq-container -- ping 8.8.8.8
```

3.8 Configuring dnsmasq

Edit the `dnsmasq` configuration file at `/etc/dnsmasq.conf`:

```
incus exec dnsmasq-container -- nano /etc/dnsmasq.conf
```

Add or modify the following settings to enable DNS and DHCP:

```
# DNS settings
domain-needed
bogus-priv
no-resolv
server=8.8.8.8
```

```
server=8.8.4.4
local=/example.local/
domain=example.local

# DHCP settings
dhcp-range=192.168.1.100,192.168.1.200,12h
dhcp-option=3,192.168.1.1
dhcp-option=6,8.8.8.8,8.8.4.4
```

Explanation:

- `domain-needed`: Prevents incomplete domain names from being sent to upstream DNS.
- `bogus-priv`: Blocks reverse DNS lookups for private IP ranges.
- `no-resolv`: Disables reading `/etc/resolv.conf`.
- `server`: Specifies upstream DNS servers (Google DNS in this case).
- `local` and `domain`: Configures a local domain.
- `dhcp-range`: Defines the IP range for DHCP clients (from 192.168.1.100 to 192.168.1.200, lease time 12 hours).
- `dhcp-option`: Sets the default gateway (option 3) and DNS servers (option 6).

3.9 Starting and Enabling dnsmasq

Restart and enable the `dnsmasq` service:

```
incus exec dnsmasq-container -- systemctl restart dnsmasq
incus exec dnsmasq-container -- systemctl enable dnsmasq
```

Verify that `dnsmasq` is running:

```
incus exec dnsmasq-container -- systemctl status dnsmasq
```

3.10 Testing the Configuration

Test DNS resolution from within the container:

```
incus exec dnsmasq-container -- nslookup example.local
   192.168.1.10
```

To test DHCP, connect a client device to the same network (via the `br0` bridge) and verify that it receives an IP address in the range `192.168.1.100-192.168.1.200`.

4 Troubleshooting

If `dnsmasq` fails to start:

- Check the logs: `incus exec dnsmasq-container - journalctl -u dnsmasq`.

- Ensure no other service is using port 53 (DNS) or 67 (DHCP).
- Verify the network configuration with `incus exec dnsmasq-container - ip a` and `incus exec dnsmasq-container - ping 8.8.8.8`.
- Confirm the veth pair and bridge setup: `ip link show` on the host and `incus exec dnsmasq-container - ip link show`.

5 Conclusion

This guide configures `dnsmasq` as a DNS and DHCP server in an Incus container on Debian. The `link.py` script and Netplan configuration ensure a robust network setup with veth pairs and static IP addressing. For advanced configurations, refer to the `dnsmasq` documentation (`man dnsmasq`) and `pyroute2` documentation.