Homelab

- Heimdall
  - GitLab vs. Gitea
- Oracle VPS as IPv4 Proxy
  - Pterodactyl
  - Teamspeak
  - RelayDock Dokumentation
- Nginx Proxy Manager

Heimdall

GitLab vs. Gitea

Oracle VPS as IPv4 Proxy

Pterodactyl

Zielsetzung

Das Routing von einkommendem IPv4-Traffic (z.B. von Steam) über einen Oracle Cloud Server, weiter per IPv6 durch das Internet zum lokalen Raspberry Pi (Internet Point für Server Traffic), und schließlich per IPv4 ins Heimnetzwerk zum eigentlichen Game-Server (Debian/Pterodactyl) oder zu Server wie Teamspeak auf dem Raspberry Pi.
Dieses Setup minimiert Latenz (Ping) durch den **Verzicht auf VPNs wie Tailscale/Wireguard** und nutzt natives Netzwerk-Routing via `socat`.

1. Übersicht & IPs (Beispiel Icarus Server)

- Oracle VPS (Public IPv4)
  - IP Adressen
  - Der Server, mit dem sich die Spieler verbinden
- Raspberry Pi (Public IPv6)
  - IP Adressen
  - Das Gateway im Heimnetz
- Game-Server Debian (Local IPv4)
  - IP Adressen
  - Der Rechner, auf dem Pterodactyl/Docker läuft
- Benötigte Ports
  - Icarus: `17777 UDP`
  - Steam: `27015 UDP`

2. Pterodactyl (Game-Server) Vorbereitung

Damit der Game-Server in Docker die weitergeleiteten Pakete von fremden IPs akzeptiert:

1. Im Pterodactyl-Panel unter Network sicherstellen, dass dem Server die IP `0.0.0.0` zugewiesen ist (nicht die lokale `192.168.x.x` Adresse).
2. Unter Startup bei `SERVER_IP` (falls vorhanden) ebenfalls `0.0.0.0` eintragen.
3. Server neustarten.

---

3. Oracle Cloud Setup (IPv4 -> IPv6)

Der Oracle-Server fängt den IPv4-Traffic der Spieler ab und leitet ihn per IPv6 an den Raspberry Pi zu Hause weiter.

Voraussetzungen auf Oracle

Sicherstellen, dass in der Oracle Cloud Web-Oberfläche (Dashboard) die benötigten Ports (`17777` und `27015 UDP`) in den Ingress Rules (Eingehend) der Virtual Cloud Network Security List geöffnet sind.
Zusätzlich müssen die Ports in der Ubuntu-Firewall geöffnet sein:

```
sudo iptables -I INPUT -p udp -m udp --dport 17777 -j ACCEPT
sudo iptables -I INPUT -p udp -m udp --dport 27015 -j ACCEPT
```

Portmapping mit socat (Oracle)

Alte socat-Prozesse beenden:

```
sudo pkill socat
```

Neue Routen für Game- und Query-Port setzen (Die IPv6 des Pi muss in eckigen Klammern stehen):

```
nohup sudo socat UDP4-LISTEN:17777,fork,reuseaddr UDP6:[IPv6-Raspberry-Pi]:17777 &
nohup sudo socat UDP4-LISTEN:27015,fork,reuseaddr UDP6:[IPv6-Raspberry-Pi]:27015 &
```

> Hinweis: Die Ausgabe nohup: ignoring input and appending output to 'nohup.out' ist normal und bedeutet, dass der Prozess erfolgreich im Hintergrund läuft.

>

> If you want to debug incoming and outcoming trafic:

```
sudo tcpdump -i any udp port 27015 -n
```

---

4. Raspberry Pi Setup (IPv6 -> Lokale IPv4)

Der Raspberry Pi empfängt die IPv6-Pakete von Oracle und leitet sie als lokale IPv4-Pakete an den Game-Server weiter.

Voraussetzungen auf dem Pi / Heimnetz

- In der FritzBox muss eine "IPv6 Portfreigabe" (oder "Ping6 erlauben") für den Raspberry Pi für die entsprechenden UDP-Ports (`17777`, `27015`) existieren.
- `socat` muss auf dem Pi installiert sein: `sudo apt install socat`

Portmapping mit socat (Raspberry Pi)

Alte socat-Prozesse beenden:

```
sudo pkill socat
```

Neue Routen setzen (Achtung: Hier steht `UDP6-LISTEN` vorne. Die lokale IPv4 benötigt keine eckigen Klammern):

```
nohup sudo socat UDP6-LISTEN:17777,fork,reuseaddr UDP4:192.168.x.x:17777 &
nohup sudo socat UDP6-LISTEN:27015,fork,reuseaddr UDP4:192.168.x.x:27015 &
```

---

5. Neue Game-Server hinzufügen (Checkliste)

Wenn ein neues Spiel (z.B. Minecraft) auf Port `25565 TCP` gehostet werden soll:

1. Pterodactyl-Binding auf `0.0.0.0` setzen.
2. Oracle Cloud Dashboard: Port `25565 TCP` Ingress freigeben.
3. Oracle Terminal: iptables INPUT für `25565 TCP` erlauben.
4. Oracle Terminal: Neuen socat-Befehl ausführen (Achtung: Bei TCP das `UDP4` zu `TCP4` und `UDP6` zu `TCP6` ändern).
5. FritzBox: IPv6 Port `25565 TCP` für den Raspberry Pi freigeben.
6. Raspberry Pi Terminal: Neuen socat-Befehl ausführen (TCP6 -> TCP4).

Teamspeak

1. Übersicht & IPs

- Oracle VPS (Public IPv4)
  - IP Adressen
  - Der Server, mit dem sich die Spieler verbinden
- Raspberry Pi (Public IPv6)
  - IP Adressen
  - Das Gateway im Heimnetz
- Game-Server Debian (Local IPv4)
  - IP Adressen
  - Der Rechner, auf dem Pterodactyl/Docker läuft
- Benötigte Ports
  - Voice: `9987 UDP`
  - Filetransfer: `30033 UDP`

---

2. Cloudflare Records

You probaly have a IPv6 (AAAA) Record for your domain: `ts.yourdomain.com`

You now create a new IPv4 (A) Record and add your Oracle VPS IPv4.

---

3. Oracle VPS

Open Ubuntu Firewall:

Voice:

```
sudo iptables -I INPUT -p udp -m udp --dport 9987 -j ACCEPT
```

Filetransfer:

```
sudo iptables -I INPUT -p tcp -m tcp --dport 30033 -j ACCEPT
```

Add new Route for Port:

Voice:

```
nohup sudo socat UDP4-LISTEN:9987,fork,reuseaddr UDP6:[IPv6-Raspberry-Pi]:9987 &
```

Filetranfer:

```
nohup sudo socat TCP4-LISTEN:30033,fork,reuseaddr TCP6:[IPv6-Raspberry-Pi]:30033 &
```

RelayDock Dokumentation

Überblick

RelayDock ist ein kleines Verwaltungsprojekt für Port-Weiterleitungen mit `socat`, `systemd`-Template-Units und einer Weboberfläche. Das Ziel ist, IPv4- und IPv6-Weiterleitungen als einzelne Dienste verwaltbar zu machen, Regeln zentral in JSON zu speichern und Änderungen über eine Weboberfläche oder per Script zu synchronisieren. [1][2]

Das Projekt ist absichtlich generisch benannt und nicht an Pterodactyl gebunden. Dadurch eignet es sich für Game-Server, TeamSpeak, Query-Ports, TCP-Dienste und andere selbst gehostete Services mit festen Portweiterleitungen. [1]

Architektur

RelayDock besteht aus fünf Hauptbausteinen:

- Eine JSON-Datei als zentrale Konfiguration für Rolle, Ports und Zieladressen.
- Ein `systemd`-Template wie `relaydock@.service`, damit jede Weiterleitung als eigene Instanz läuft. [1][2]
- Ein Instanz-Script, das für genau einen Regel-Namen die passende `socat`-Weiterleitung startet.
- Ein Reconcile-Script, das JSON und vorhandene `systemd`-Instanzen abgleicht.
- Eine Weboberfläche, die Regeln anlegt, löscht, startet, stoppt und synchronisiert.

Das Template-Prinzip ist sinnvoll, weil `systemd` Instanzen über `%i` oder `%I` parametrieren kann. Dadurch werden Dienste wie `relaydock@icarus-game.service` oder `relaydock@steam-query.service` möglich, was einzelne Neustarts, Statusabfragen und isolierte Fehlerdiagnosen erlaubt. [1][2][3]

Voraussetzungen

Für RelayDock werden mindestens `socat`, `jq`, `python3` und `python3-flask` benötigt. `jq` ist ein üblicher Weg, JSON in Shell-Scripts sauber auszuwerten, während Flask als einfacher Webdienst unter `systemd` betrieben werden kann. [4][5][6]

Empfohlene Installation auf Debian oder Ubuntu:

```
sudo apt update
sudo apt install -y socat jq python3 python3-flask nano
```

Wenn `visudo` beim Bearbeiten der `sudoers`-Dateien keinen Editor findet, liegt das meist daran, dass `/usr/bin/editor` nicht korrekt gesetzt ist oder kein Editor installiert wurde. In dem Fall hilft häufig `sudo apt install nano` und danach `sudo EDITOR=nano visudo`. [7][8]

Verzeichnisstruktur

Eine sinnvolle Zielstruktur sieht so aus:

```
/opt/relaydock/
├── app.py
├── templates/
├── static/
├── relaydock@.service
├── relaydock.target
├── relaydock-instance.sh
├── relaydock-reconcile.sh
└── README.md

/etc/relaydock/
└── config.json

/usr/local/bin/
├── relaydock-instance.sh
└── relaydock-reconcile.sh

/etc/systemd/system/
├── relaydock@.service
├── relaydock.target
└── relaydock-web.service
```

Diese Trennung hält Anwendung, Konfiguration und Systemdateien sauber auseinander. Das erleichtert Updates, Backups und spätere Härtung. [5][9]

Konfiguration per JSON

Die JSON-Datei ist die zentrale Datenquelle. Sie definiert, ob der Host als `oracle` oder `pi` arbeitet und welche Weiterleitungsregeln existieren.

Beispiel:

```
{
  "role": "oracle",
  "rules": [
    {
      "name": "icarus-game",
      "proto": "udp",
      "listen_port": 17777,
      "target_host": "2001:db8::1234",
      "target_port": 17777
    },
    {
      "name": "steam-query",
      "proto": "udp",
      "listen_port": 27015,
      "target_host": "2001:db8::1234",
      "target_port": 27015
    }
  ]
}
```

Mit `jq` kann diese Datei robust geprüft und geparst werden. Ein einfacher Syntaxcheck ist mit `jq empty /etc/relaydock/config.json` möglich. [4][10][11]

Bedeutung der Rolle

Die Rolle bestimmt, wie das Instanz-Script `socat` startet:

- `oracle`: eingehend auf IPv4, ausgehend auf IPv6.
- `pi`: eingehend auf IPv6, ausgehend auf lokales IPv4.

Für UDP auf Oracle wäre das typischerweise `UDP4-LISTEN:PORT` nach `UDP6:[IPv6]:PORT`, auf dem Raspberry Pi entsprechend `UDP6-LISTEN:PORT` nach `UDP4:192.168.x.x:PORT`. Diese Trennung spiegelt

die in RelayDock hinterlegte Logik wider. [12]

Installation

1. Pakete installieren

```
sudo apt update
sudo apt install -y socat jq python3 python3-flask nano
```

2. Dateien kopieren

```
sudo mkdir -p /opt/relaydock /etc/relaydock
sudo cp -r relaydock/* /opt/relaydock/
sudo cp /opt/relaydock/config.json /etc/relaydock/config.json
sudo cp /opt/relaydock/relaydock@.service /etc/systemd/system/
sudo cp /opt/relaydock/relaydock.target /etc/systemd/system/
sudo cp /opt/relaydock/relaydock-web.service /etc/systemd/system/
sudo cp /opt/relaydock/relaydock-instance.sh /usr/local/bin/
sudo cp /opt/relaydock/relaydock-reconcile.sh /usr/local/bin/
sudo chmod +x /usr/local/bin/relaydock-instance.sh /usr/local/bin/relaydock-reconcile.sh
```

3. `systemd` neu laden

```
sudo systemctl daemon-reload
```

`systemd` verwaltet Units und Unit-Dateien getrennt; nach neuen oder geänderten Service-Dateien ist `daemon-reload` der normale Schritt. [9][13]

4. Webdienst aktivieren

```
sudo systemctl enable --now relaydock-web.service
```

5. Regeln synchronisieren

```
sudo /usr/local/bin/relaydock-reconcile.sh /etc/relaydock/config.json
```

Danach sollten die passenden Instanzen wie `relaydock@icarus-game.service` erzeugt und gestartet sein. [1][9]

Arbeiten mit `systemd`-Instanzen

Wichtige Befehle:

```
systemctl list-units 'relaydock@*.service' --all
systemctl list-unit-files 'relaydock@*.service'
systemctl status relaydock@icarus-game.service
journalctl -u relaydock@icarus-game.service -n 100
```

`list-units` zeigt geladene oder aktive Units, während `list-unit-files` zeigt, welche Unit-Dateien bekannt sind und welchen Enable-Status sie haben. Diese Unterscheidung ist wichtig, wenn gelöschte Regeln scheinbar noch in `systemctl` auftauchen. [14][15][16]

Weboberfläche

Die Weboberfläche soll folgende Aufgaben übernehmen:

- Rolle `oracle` oder `pi` setzen.
- Neue Regeln anlegen.
- Einzelne Regeln löschen.
- Einzelne Instanzen starten, stoppen oder neu starten.
- Die JSON mit den `systemd`-Instanzen synchronisieren.
- Optional `daemon-reload` auslösen.

Flask eignet sich gut für so ein kleines internes Admin-Panel und kann über eine eigene `systemd`-Unit als Dienst laufen. Das ist ein üblicher Weg, Flask produktionsnah in Linux-Diensten einzubinden. [17][5][6]

Sicherheit

Für erste Tests kann die Weboberfläche als root laufen, weil das die Komplexität reduziert. Langfristig ist das aber riskant, weil jede Schwachstelle in der Webanwendung dann direkten Root-Zugriff ermöglicht. Stattdessen wird üblicherweise empfohlen, den Webdienst als eingeschränkten

Benutzer auszuführen und nur gezielte Verwaltungsbefehle über `sudoers` oder einen Wrapper freizugeben. [18][19]

Wenn `sudoers` verwendet wird, ist eine separate Datei unter `/etc/sudoers.d/` sauberer als direkt in `/etc/sudoers` zu schreiben. `visudo -f /etc/sudoers.d/relaydock` ist dafür der empfohlene Weg, und `visudo -c` prüft anschließend die Syntax. [20][21][22]

Wildcard-Regeln in `sudoers` sind praktisch, können aber riskant sein, weil sie mehr matchen können als erwartet. Deshalb ist ein kleines, fest validierendes Wrapper-Script oft sicherer als sehr breite Wildcard-Freigaben. [23][24]

Typische Probleme und Lösungen

`visudo: no editor found (editor path = /usr/bin/editor)`

Dieses Problem bedeutet, dass kein nutzbarer Editor gefunden wurde. Übliche Lösung:

```
sudo apt install nano
sudo EDITOR=nano visudo
```

Alternativ kann dauerhaft mit `sudo update-alternatives --config editor` ein Standard-Editor gesetzt werden. [7][25][8]

Gelöschte Services erscheinen noch in `systemctl`

Wenn Regeln in der Weboberfläche gelöscht werden, aber unter `systemctl` noch sichtbar sind, wurde meist nur der JSON-Eintrag entfernt. Die zugehörige `systemd`-Instanz wurde dann nicht gestoppt oder deaktiviert. `systemctl list-units --all` zeigt solche Units weiterhin an, solange sie noch geladen, aktiv oder fehlgeschlagen sind. [14][15][13]

Abhilfe:

```
sudo systemctl stop relaydock@alte-regel.service
sudo systemctl disable relaydock@alte-regel.service
sudo systemctl daemon-reload
sudo systemctl reset-failed
```

Das Reconcile-Script sollte gelöschte Regeln deshalb nicht nur aus der JSON entfernen, sondern alte Instanzen explizit mit `stop` und `disable` aufräumen. [26][27]

Weboberfläche läuft als root

Das ist für einen schnellen Prototyp technisch in Ordnung, aber nicht die empfohlene Endlösung. Die Weboberfläche sollte dann zumindest nicht offen im Internet erreichbar sein, sondern nur intern oder über eine restriktive Firewall. Ein Root-Webdienst erhöht die Auswirkungen jeder Sicherheitslücke erheblich. [18][28]

`systemd`-Instanzen reagieren nicht auf JSON-Änderungen

Nach JSON-Änderungen muss das Reconcile-Script ausgeführt werden. Wenn nur die Datei geändert wird, aber kein Abgleich stattfindet, laufen bestehende Dienste mit alter Konfiguration weiter. Das ist normal, weil `systemd` laufende Prozesse nicht automatisch an Dateiänderungen bindet. [9][27]

UDP funktioniert unzuverlässig

UDP-Weiterleitungen mit `socat` können problematischer sein als TCP, insbesondere wenn Antworten, Quelladressen oder Sitzungszuordnung nicht so verlaufen wie das Spielprotokoll es erwartet. `fork` und die zustandslose Natur von UDP können dabei zu Verhalten führen, das im Test teilweise funktioniert, aber in echten Spielszenarien instabil wirkt. [12][29]

Empfohlener Betriebsablauf

1. Regeln in der Weboberfläche anlegen oder anpassen.
2. JSON speichern.
3. Reconcile ausführen.
4. Status der betroffenen Instanzen mit `systemctl status relaydock@NAME.service` prüfen.
5. Bei Problemen Logs mit `journalctl -u relaydock@NAME.service -n 100` ansehen.
6. Vor externen Tests Firewall, Cloud-Ingress und Router-Freigaben prüfen.

Diese Reihenfolge reduziert Fehler, weil sie Konfigurationsänderung, Dienstabgleich und Statusprüfung klar trennt. [9][13]

Betrieb auf Oracle und Raspberry Pi

Auf Oracle müssen die benötigten Ports zusätzlich in der Cloud-Firewall und lokal freigegeben sein. Auf dem Raspberry Pi müssen die IPv6-Freigaben am Router und die lokalen Ziele im Heimnetz korrekt gesetzt sein. RelayDock löst nur die lokale Prozessverwaltung der Weiterleitungen, nicht die vorgelagerten Firewall- oder Router-Regeln. [13]

Verbesserungsvorschläge

Folgende Erweiterungen sind für produktiven Einsatz sinnvoll:

- Bearbeiten bestehender Regeln im UI statt nur Hinzufügen und Löschen.
- Eingabevalidierung für IPv4, IPv6 und doppelte Listen-Ports.
- Authentifizierung für die Weboberfläche.
- Betrieb des Webdienstes als eigener User statt root. [18][19]
- Wrapper-Script statt breiter `sudoers`-Wildcards. [23][24]
- Eine Log-Ansicht im UI für `journalctl`.
- Sichtbare Hinweise, ob eine Regel nur in JSON existiert oder bereits aktiv synchronisiert wurde.

Diagnosebefehle

Nützliche Befehle für die Fehlersuche:

```
sudo systemctl daemon-reload
sudo systemctl status relaydock-web.service
sudo systemctl status relaydock@icarus-game.service
sudo journalctl -u relaydock-web.service -n 100
sudo journalctl -u relaydock@icarus-game.service -n 100
sudo jq empty /etc/relaydock/config.json
sudo tcpdump -i any udp port 27015 -n
```

`tcpdump` ist besonders hilfreich, wenn geprüft werden soll, ob UDP-Pakete überhaupt am richtigen Interface ankommen oder das System wieder verlassen. [13][12]

Fazit

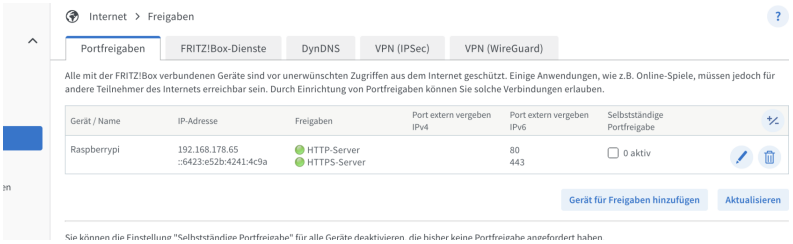
RelayDock kombiniert eine zentrale JSON-Konfiguration mit `systemd`-Template-Units und einer Weboberfläche zu einem flexiblen Verwaltungsansatz für Portweiterleitungen. Die größte Stärke liegt darin, dass jede Weiterleitung als eigene Instanz behandelt werden kann, während die Weboberfläche als bequeme Verwaltungsoberfläche dient. [1][2]

Für einen ersten funktionalen Aufbau reicht ein direkter Root-Betrieb der Weboberfläche aus. Für dauerhaften Einsatz sollten jedoch Härtung, Authentifizierung, sauber begrenzte Privilegien und ein robusteres Aufräumen gelöschter Instanzen ergänzt werden. [18][19]

Nginx Proxy Manager

Portfreigaben

Wird nur http und https Traffic weitergeleitet müssen folgende Portfreigaben in der FritzBox gemacht werden auch wenn der HTTP Host intern auf einem anderen Port läuft



In diesem Fall leiten wir an die IPv6 des Proxys Managers weiter es ist wichtig dass die IPv6 des Host mit der in der FritzBox übereinstimmt da dies aus unerklärlichen Gründen nicht immer automatisch der Fall ist

DNS weiterleitung in Cloudflare

In Cloudflare muss einmal die root Domain freigeben werden und mit einem * *alle anderen Wildcard domains es ist wichtig dass auch die root Domain freigeben wird da dass * für die* Wildcard domains sonst nicht in kraft tritt!