WebSocket-Stabilität in Caddy optimieren

https://gemini.google.com/app/988f2a81ee801ae5

User prompt: Ich stelle fest, dass Websockets nicht stabil die Verbindung behalten. Wie kann ich Candy als Reverse Proxy optimieren?

Response: Instabile WebSocket-Verbindungen hinter Caddy lassen sich meistens auf Timeouts, Load-Balancer-Einstellungen oder Timeouts bei Inaktivität (TCP-Keepalive) zurückführen. Da WebSockets langlebige TCP-Verbindungen sind, brechen sie oft ab, wenn Router, Firewalls oder Proxies sie wegen mangelnder Aktivität für tot erklären.

Hier sind die wichtigsten Stellschrauben in der **Caddyfile**, um WebSockets dauerhaft stabil zu halten:

1. Das Caddyfile optimieren

Fügen Sie in Ihrer `Caddyfile` explizite Transport- und Timeout-Regeln für den Reverse Proxy hinzu:

```
example.com {
    reverse_proxy localhost:3000 {
        # Verhindert, dass Caddy den Upstream zu früh als tot betrachtet
        transport http {
            dial_timeout 5s
            response_header_timeout 30s
            read_timeout 0s
            write_timeout 0s
        }

        # Falls Verbindungen nach einer bestimmten Zeit ohne Daten,
        # aber ohne erkennbaren Grund abbrechen:
        # stream_timeout 24h
    }
}
```

- **`read_timeout 0s` / `write_timeout 0s`**: Standardmäßig unbegrenzt, aber das explizite Setzen (oder Erhöhen auf hohe Werte) verhindert, dass Caddy stille WebSocket-Streams wegen vermeintlicher Inaktivität abbricht.

2. TCP-Keepalive auf globaler Ebene anpassen

Wenn Verbindungen nach exakt derselben Zeitspanne (z. B. nach einigen Minuten Stille) abbrechen, liegt das oft daran, dass NAT-Gateways oder Firewalls dazwischen die Verbindung kappen. Sie können Caddy dazu bringen, regelmäßig TCP-Keepalive-Pakete zu senden:

Fügen Sie ganz oben in Ihre `Caddyfile` (in den globalen Block) folgendes ein:

```
{
    servers {
        # Sendet TCP-Keepalive-Pakete alle 15 Sekunden
        keepalive_interval 15s
        keepalive_idle 30s
    }
}
```

- **Auswirkung:** Dies hält die Verbindung auf Netzwerkebene aktiv, sodass Stateful Firewalls

oder Router den WebSocket nicht wegen Inaktivität droppen.

3. Client-seitiges Heartbeat (Ping/Pong) implementieren

Caddy selbst leitet WebSocket-Frames einfach nur transparent weiter. Die robusteste Lösung gegen instabile Verbindungen bleibt jedoch ein **Applikations-Ebene-Ping/Pong** (z. B. im Code Ihrer Web-App oder Ihres Clients implementiert). Wenn der Client oder Server alle 20–30 Sekunden einen winzigen Ping-Frame sendet, bleibt die Verbindung unabhängig von Proxy-Timeouts garantiert offen.

Verifizierung

Nachdem Sie die Änderungen in der Caddyfile vorgenommen haben, laden Sie die Konfiguration neu:

```
caddy reload --config /etc/caddy/Caddyfile
```

- **Erfolgsprüfung:** Öffnen Sie die Entwicklertools Ihres Browsers (F12) → Reiter **Network** → Filter **WS**. Prüfen Sie, ob die Verbindung nun über einen längeren Zeitraum (auch bei Inaktivität) im Status `101 Switching Protocols` verbleibt, anstatt nach kurzer Zeit mit `Closed` oder `Aborted` abzubrechen.

Sources

1. Module http.reverse_proxy.transport.http - Caddy Documentation
2. Global options (Caddyfile) — Caddy Documentation
