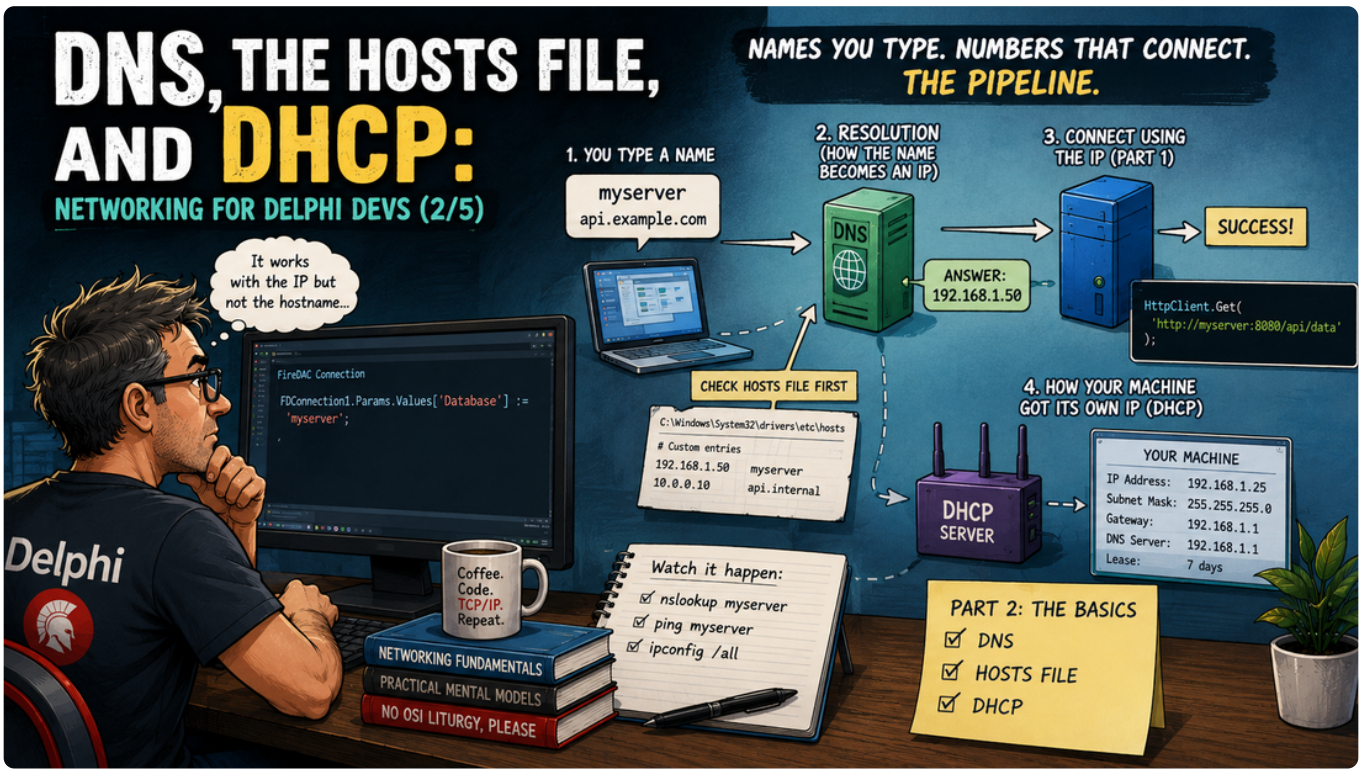
NEWS · JULY 11, 2026

DNS, die hosts-Datei und DHCP: Netzwerk-Grundlagen für Delphi-Entwickler (2/5)

By Dr. Holger Flick



In Teil 1 haben wir die Zahlen geklärt: Jede Maschine im Netzwerk hat eine IP-Adresse, `127.0.0.1` ist immer *diese* Maschine, und ein Port wählt das konkrete Programm hinter dieser Adresse aus. Zahlen, wohin man schaut.

Aber genau da liegt der Punkt — Sie *tippen* diese Zahlen fast nie. Sie schreiben `myserver` in einen FireDAC-Connection-String. Sie richten einen REST-Client auf `api.example.com`. Sie öffnen `https://www.embarcadero.com` im Browser. Überall Namen — und irgendwie werden aus allen am Ende die Zahlen aus Teil 1.

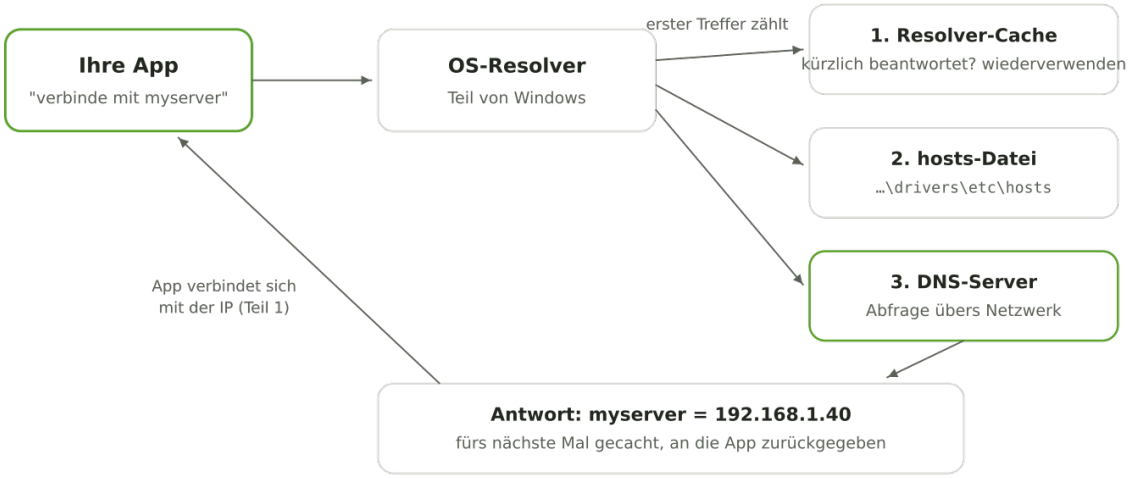
Diese Lücke — zwischen dem Namen, den Sie eintippen, und der Verbindung, die sich öffnet — ist der Lebensraum einer ganzen Klasse von "it works on my machine"-Bugs. *"Mit der IP geht es, mit dem Hostnamen nicht." "Gestern ging es noch, und am Server wurde nichts geändert." "Auf dem Server selbst funktioniert es, vom Client-PC aus nicht."* Jeder dieser Fälle ist ein Namensauflösungs-Problem, und jeder davon ist in Minuten diagnostizierbar, sobald Sie die Pipeline kennen.

Dieser Teil gibt Ihnen genau diese Pipeline: wie aus einem Namen eine Adresse wird (DNS und die hosts-Datei) — und, in die andere Richtung, wer Ihrer Maschine überhaupt ihre eigene Adresse gegeben hat (DHCP). Am Ende können Sie die Auflösung mit drei Kommandos live auf Ihrem eigenen Rechner beobachten.

Vom Namen zur Nummer: die Auflösungs-Pipeline

Wenn Ihre Delphi-App — oder irgendein Programm — eine Verbindung zu `myserver` anfordert, erledigt sie die Detektivarbeit nicht selbst. Sie übergibt den Namen an den **Resolver** des Betriebssystems, einen eingebauten Dienst, dessen einzige Aufgabe es ist, Namen in IP-Adressen zu verwandeln. Der Resolver arbeitet dann eine kurze Checkliste ab, und der erste Treffer gewinnt.

Unter Windows ist die dokumentierte Reihenfolge im Kern: erst der lokale Cache und die **hosts-Datei**, dann die Anfrage an einen **DNS-Server**, mit älteren NetBIOS-Mechanismen als Legacy-Fallback (Microsofts Reihenfolge der Hostnamen-Auflösung). Hier der gesamte Ablauf in einem Bild.



Namensauflösung auf Ihrer Maschine: erst lokale Antworten, dann das Netzwerk — und das Ergebnis wird gecacht

Lesen Sie das Diagramm auf der rechten Seite von oben nach unten, und Sie haben die ganze Geschichte: Der Resolver prüft zuerst, ob er die Antwort schon kennt (der Cache), schaut dann in eine schlichte Textdatei auf Ihrer eigenen Festplatte (die hosts-Datei) — und geht erst danach ins Netzwerk, um einen DNS-Server zu fragen. Die Antwort, die zurückkommt, wird gecacht, an Ihre App übergeben, und Ihre App verbindet sich mit der IP, genau wie in Teil 1. Daraus folgen sofort zwei Dinge, und beide sind fürs Debugging wichtig:

1. **Ihre App spricht nie direkt mit DNS.** FireDAC, `TNetHTTPClient`, Ihr Browser — sie alle fragen denselben OS-Resolver. Deshalb schlägt ein Name, der in Ihrer Delphi-App scheitert, auf derselben Maschine auch im Browser fehl — und deshalb ist der Test im Browser ein völlig legitimer Diagnoseschritt.
2. **Lokale Antworten schlagen Netzwerk-Antworten.** Ein Eintrag in der hosts-Datei überschreibt stillschweigend alles, was DNS gesagt hätte. Das ist ein Feature — und gelegentlich der Bug.

Die hosts-Datei: Ihr persönliches Telefonbuch-Override

Bevor es DNS gab, gab es buchstäblich eine gemeinsam gepflegte Textdatei mit Namen und Adressen — und ein Nachfahre dieser Datei liegt bis heute jedem Betriebssystem bei. Unter Windows findet sie sich unter `C:\Windows\System32\drivers\etc\hosts` (formal `%WinDir%\System32\drivers\etc`, laut Microsofts Dokumentation zur hosts-Datei); sie hat keine Dateiendung, und jede Zeile besteht schlicht aus einer IP-Adresse gefolgt von einem oder mehreren Namen:

```
192.168.1.40    myserver
192.168.1.40    api.mycompany.test
```

Weil der Resolver diese Datei konsultiert, *bevor* er irgendeinen DNS-Server fragt, ist sie auf sehr spezielle Weise der Freund des Entwicklers: **Sie können jeden Namen alles bedeuten lassen — nur auf Ihrer Maschine, ohne einen einzigen Server anzufassen.** Ein paar Anwendungen aus dem Alltag:

- Lassen Sie `api.mycompany.test` auf die IP eines Testservers zeigen, damit sich die Konfiguration Ihrer App zwischen Umgebungen nicht ändert — nur die hosts-Datei tut es.
- Lassen Sie einen Produktions-Hostnamen auf eine Staging-Maschine zeigen, um "die echte Konfiguration" gefahrlos von einem PC aus zu testen.
- Geben Sie der Maschine eines Kollegen einen stabilen Namen, solange Sie gegen sie integrieren.

> **Zum Bearbeiten der hosts-Datei brauchen Sie Administratorrechte**
> Die Datei ist geschützt. Öffnen Sie Notepad (oder Ihren Editor) **als Administrator** und dann `C:\Windows\System32\drivers\etc\hosts`. Zeilen, die mit `#` beginnen, sind Kommentare. Und denken Sie an die Kehrseite von "lokale Antworten gewinnen": Ein vergessener hosts-Eintrag ist ein Klassiker unter den Ursachen für "diese Maschine verhält sich anders als jede andere". Wenn die Namensauflösung wie verhext wirkt, prüfen Sie zuerst die hosts-Datei — das dauert zehn Sekunden.

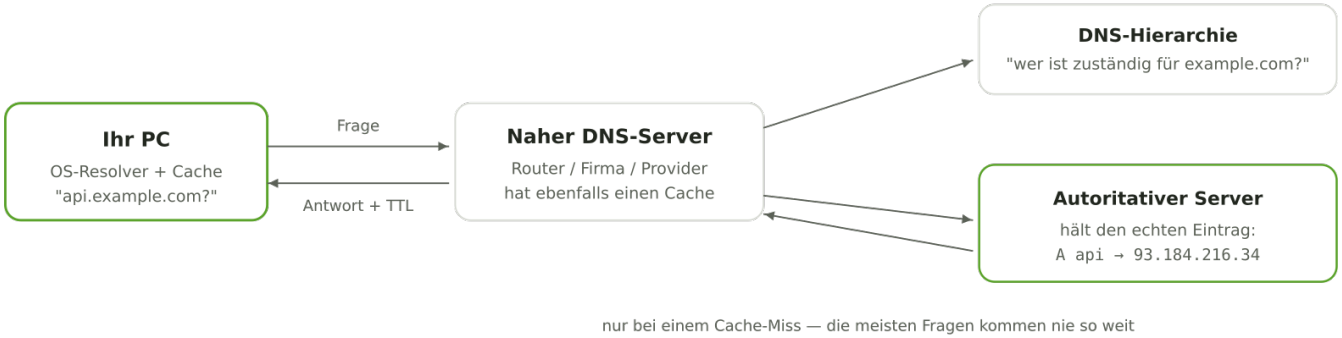
DNS: das Telefonbuch, das niemandem ganz gehört

Für jeden Namen, der nicht in Ihrer hosts-Datei steht, fragt der Resolver das Domain Name System — DNS —, definiert in den RFCs 1034 und 1035 und konzeptionell seit 1987 im Wesentlichen unverändert in Betrieb.

Das Denkmodell, das Ihnen am meisten hilft: **DNS ist ein verteiltes Telefonbuch.** Keine einzelne Maschine hält das ganze Buch; stattdessen wird Verantwortung delegiert. Jemand, der für `example.com` autoritativ ist, veröffentlicht die Einträge für Namen darunter, jemand anderes ist für `embarcadero.com` autoritativ, und eine Hierarchie von Servern weiß, wen man wonach fragt.

Der Eintragstyp, der Sie in neunzig Prozent der Fälle interessiert, ist der **A-Record**: eine simple Zuordnung *Name* ’*IPv4-Adresse*, z. B. `api.example.com ’ 93.184.216.34`. Sein Geschwister, der **AAAA-Record**, tut dasselbe für IPv6. Es gibt viele weitere Record-Typen — Mail-Routing, Aliase, Text-Metadaten —, registriert bei der IANA, der Organisation, die die globalen Identifikatoren des Internets koordiniert. Sie müssen sie jetzt nicht lernen; Sie müssen nur wissen, dass es sie gibt, damit die Begriffe Sie nicht erschrecken, wenn ein Server-Admin einen "CNAME" erwähnt.

Und welchen DNS-Server fragt *Ihre* Maschine? Fast immer einen, der ihr automatisch mitgeteilt wurde: Im Heim- oder Büronetzwerk verkündet typischerweise Ihr Router oder der Firmenserver beim Verteilen der Adressen "nimm mich (oder diese Adresse) für DNS" — diesen Mechanismus (DHCP) lernen wir gleich kennen. Hinter diesem ersten Server sitzt der Resolver des Providers oder der Firma, und dahinter die öffentliche Hierarchie. Ihren konfigurierten DNS-Server sehen Sie jederzeit mit `ipconfig /all`.



Das verteilte Telefonbuch: Ihr Resolver fragt einen nahen DNS-Server, der die autoritative Antwort findet — und cacht

Die zentrale Erkenntnis in diesem Bild ist das Caching auf *jeder* Ebene. Ihr PC cacht Antworten, der nahe DNS-Server cacht Antworten, und jede gecachte Antwort trägt eine **TTL** ("time to live") — eine Dauer, festgelegt von demjenigen, der den Record veröffentlicht hat, nach deren Ablauf die Antwort verworfen und frisch geholt werden muss (die TTL ist Teil jedes Records in RFC 1035). Caching ist der Grund, warum das System schnell ist. Es ist auch die Antwort auf eine Frage, die jeder Entwickler irgendwann stellt:

> **Ich habe den DNS-Eintrag geändert — warum verbindet sich meine App noch mit der alten Adresse?**
> Weil irgendwo zwischen Ihnen und dem autoritativen Server ein Cache noch die alte Antwort ausliefert — und das bis zum Ablauf seiner TTL auch darf. Nichts ist kaputt; das Telefonbuch ist schlicht so entworfen, dass es leicht veraltete Seiten toleriert und dafür Geschwindigkeit gewinnt. Lokal können Sie den Cache *Ihrer* Maschine mit `ipconfig /flushdns` leeren — aber die Caches dazwischen können Sie nicht leeren.
>
> Wenn jemand einen Serverumzug plant und "einen Tag vorher die TTL senkt", ist es genau das, was er damit steuert.

Warum localhost nie jemanden fragt

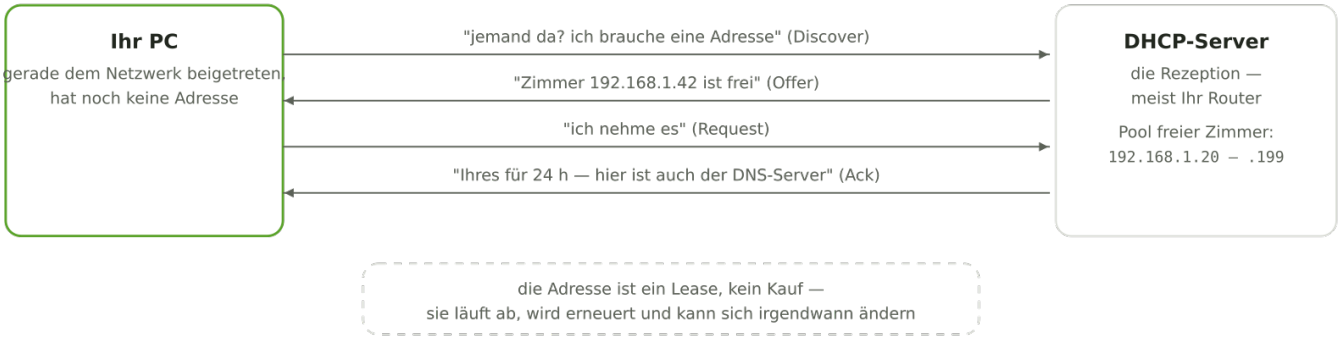
Ein Name aus Teil 1 verdient eine besondere Erwähnung: `localhost`. Er löst ohne jede DNS-Abfrage zur Loopback-Adresse auf (`127.0.0.1`, bzw. `::1` bei IPv6) — der Name ist in RFC 6761 formal reserviert, wo festgelegt ist, dass Resolver ihn selbst mit Loopback beantworten und nie ins Netzwerk schicken sollen. Windows macht das auf charmante Weise sichtbar: Die Standard-hosts-Datei *kommentiert* die localhost-Einträge *aus*, mit dem Hinweis "`localhost name resolution is handled within DNS itself`" — sprich: Der Resolver erledigt das intern, kein Dateieintrag und kein Server nötig (Microsofts Standard-hosts-Datei).

Praktische Konsequenz: `localhost` funktioniert auf einer Maschine ohne Netzwerkkabel, ohne DNS-Server, ganz ohne Konfiguration. Deshalb verwendet es jedes Tutorial für lokale Entwicklung — es ist der eine Name, den keines der Themen dieses Beitrags kaputt machen kann.

DHCP: Wer hat meinem Rechner überhaupt seine Adresse gegeben?

Bisher haben wir die Namen *anderer* Maschinen aufgelöst. Drehen wir die Frage um: Ihr eigener PC hat eine IP-Adresse — woher kommt sie? Sie haben sie mit ziemlicher Sicherheit nie eingetippt. Sie kam von DHCP, dem Dynamic Host Configuration Protocol (RFC 2131), und das passende Denkmodell ist eine **Hotelrezeption**.

Wenn Ihre Maschine einem Netzwerk beitritt — hochfährt, aus dem Ruhezustand erwacht, sich mit dem WLAN verbindet —, geht sie im Grunde zur Rezeption und sagt: "Ich hätte gern ein Zimmer." Der DHCP-Server (zu Hause meist auf Ihrem Router, in der Firma ein dedizierter Server) wählt eine freie Adresse aus seinem Pool und übergibt sie — zusammen mit den anderen Essentials des Netzwerks, unter anderem *welcher DNS-Server zu verwenden ist*. Damit ist der lose Faden aus dem DNS-Abschnitt verknüpft: Die Rezeption gibt Ihnen nicht nur eine Zimmernummer, sie drückt Ihnen auch die Adresse des Telefonbuchs in die Hand.



DHCP als Hotelrezeption: Ihre Maschine fragt, der Server bietet ein Zimmer an — für begrenzte Zeit

Die vier Pfeile sind die tatsächliche DHCP-Konversation (Discover, Offer, Request, Acknowledge — alle in RFC 2131 definiert), aber die gestrichelte Box unten ist der Teil, der Entwickler beißt: Die Adresse ist ein **Lease** mit Ablaufzeit. Eine laufende Maschine erneuert ihren Lease normalerweise still und behält dieselbe Adresse. Aber lassen Sie eine Maschine über ein langes Wochenende ausgeschaltet, oder lassen Sie den Lease verfallen, während der Pool gut gefüllt ist mit Gästen — und sie kann mit einer *anderen* Adresse zurückkommen. Ganz bewusst so entworfen.

Verbinden Sie das nun mit einem Bug, den Sie vielleicht schon einmal ausgeliefert haben: Sie schreiben die IP eines Kollegen — `192.168.1.42` — fest in eine Konfigurationsdatei, weil "das ist Annas Maschine, da läuft die Testdatenbank". Es funktioniert wochenlang. Dann verbindet sich eines Morgens nichts mehr, niemand hat etwas geändert, und Sie verbrennen einen halben Tag — weil Annas Maschine beim Erneuern ein anderes Zimmer bekommen hat. **Die Lösung ist die ganze These dieses Beitrags: Sprechen Sie Maschinen mit Namen an, und lassen Sie die Auflösung die aktuelle Nummer finden.**

Für Maschinen, die *Dienste anbieten* — der Datenbankserver, die Build-Maschine, der Lizenzserver —, lösen Netzwerke das andersherum: Sie bekommen eine **statische Adresse** (von Hand konfiguriert, außerhalb des Pools) oder eine **DHCP-Reservierung** (die Rezeption hält dasselbe Zimmer dauerhaft für genau diesen Gast frei). So oder so: Server bekommen stabile Adressen, Clients bekommen geleaste — und Namen kaschieren den Unterschied für alle.

Der Werkzeugkasten: Namensauflösung live beobachten

Genug Theorie — Windows liefert alles mit, was Sie brauchen, um dieser Pipeline bei der Arbeit zuzusehen, und es dauert fünf Minuten. Alle Kommandos laufen in einer normalen Eingabeaufforderung oder PowerShell; nur Schritt 5 verändert etwas.

Schritt 1 — Eigenen Lease und DNS-Server ansehen

Führen Sie `ipconfig /all` aus und suchen Sie Ihren aktiven Adapter. Sie sehen Ihre IP-Adresse, ob "DHCP Enabled" auf Yes steht, den **DHCP Server**, der sie Ihnen geleast hat, die Zeitstempel "Lease Obtained"/"Lease Expires" und die **DNS Servers**, die Ihr Resolver fragt. Dieser eine Bildschirm ist die gesamte erste Hälfte dieses Beitrags — live.

```
ipconfig /all
```

Schritt 2 — DNS direkt eine Frage stellen

`nslookup` schickt eine DNS-Abfrage und zeigt Ihnen die rohe Antwort — welcher Server geantwortet hat und welche Adressen zurückkamen:

```
nslookup www.embarcadero.com
```

Beachten Sie, was dieses Tool *nicht* tut: `nslookup` spricht direkt mit dem DNS-Server und umgeht Ihre hosts-Datei und den lokalen Cache. Das macht es zum Skalpell: Liefert `nslookup` die richtige Adresse, aber Ihre App verbindet sich mit der falschen, liegt das Problem lokal (hosts-Datei oder Cache), nicht beim DNS-Server.

Schritt 3 — Den lokalen Cache inspizieren

Führen Sie `ipconfig /displaydns` aus, um jede Antwort auszugeben, die Ihr Resolver gecacht hat — jede mit ihrer herunterzählenden Rest-TTL. Besuchen Sie erst eine Website, führen Sie es erneut aus, und sehen Sie zu, wie der neue Eintrag auftaucht.

```
ipconfig /displaydns
```

Schritt 4 — Beweisen, dass die hosts-Datei gewinnt

Fügen Sie eine Zeile in `C:\Windows\System32\drivers\etc\hosts` ein (Editor als Administrator gestartet), z. B. `127.0.0.1 testbox`, und dann `ping testbox`. Der Name löst sofort zu Loopback auf — kein DNS-Server wurde befragt. Entfernen Sie die Zeile, wenn Sie fertig sind.

Schritt 5 — Den Cache leeren

Wenn Sie einen hosts-Eintrag oder einen DNS-Record geändert haben und Ihre Maschine veraltete Antworten vergessen soll, leeren Sie den Cache (Eingabeaufforderung als Administrator):

```
ipconfig /flushdns
```

> **Plattformübergreifende und Open-Source-Alternativen**
>
> Für alles oben gibt es überall freie Gegenstücke: Unter Windows ist PowerShells `Resolve-DnsName` ein moderneres `nslookup`; unter Linux und macOS ist das Open-Source-Werkzeug `dig` (Teil von ISCs BIND, dem klassischen Open-Source-DNS-Server) das Standardtool, und die hosts-Datei liegt unter `/etc/hosts`. Die Konzepte dieses Beitrags sind auf jeder Plattform identisch — nur Pfade und Toolnamen ändern sich.

Was das für Ihren Delphi-Code bedeutet

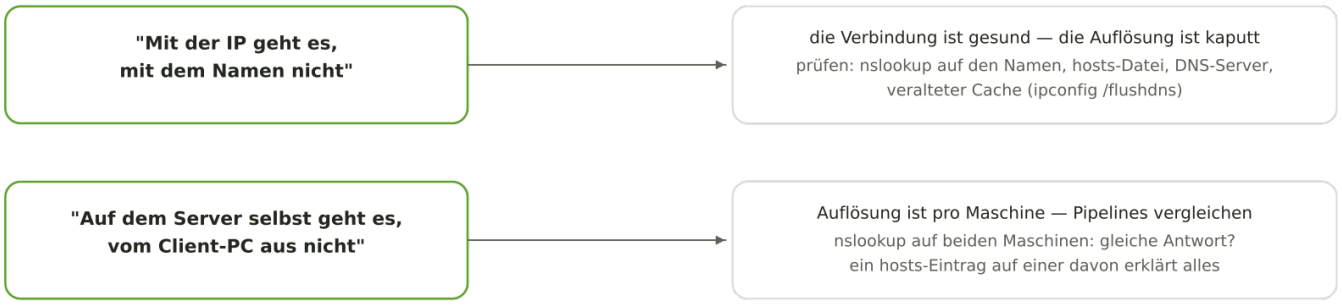
Hier kommt die Pointe für Delphi-Entwickler: **Ihr Code löst Namen nie selbst auf — und das ist ein Feature.**

Wenn FireDAC mit `Server=dbserver01` in seinen Parametern verbindet, oder wenn Sie einen Request wie diesen absetzen —

```
uses
  System.Net.HttpClient;

var
  LClient: THTTPClient;
begin
  LClient := THTTPClient.Create;
  try
    // "api.mycompany.test" durchläuft exakt die Pipeline von oben:
    // Cache -> hosts-Datei -> DNS -> Verbindung zur ermittelten IP.
    var LResp := LClient.Get('http://api.mycompany.test:8080/status');
    ShowMessage(LResp.ContentAsString);
  finally
    LClient.Free;
  end;
end;
```

— dann übergibt der `THTTPClient` aus der modernen RTL (sein Komponenten-Wrapper ist `TNetHTTPClient`) den Hostnamen einfach an den OS-Resolver. Genau wie FireDAC, genau wie jeder REST-Client, genau wie die gesamte RTL des aktuellen Delphi 13 Florence. Es gibt kein "Delphi-DNS" — es gibt nur die Auflösungs-Pipeline der Maschine, und damit lassen sich die klassischen Support-Anrufe mechanisch entschlüsseln.



Zwei klassische Support-Anrufe, entschlüsselt durch die Pipeline

Die Erkenntnis aus diesem Diagramm: Beide "Mysterien" hören auf, Mysterien zu sein, sobald Sie sich erinnern, dass die Auflösung **pro Maschine und vor der Verbindung** stattfindet. "Mit der IP geht es" heißt: Alles aus

Teil 1 ist gesund, und nur der Schritt Name’Nummer ist krank. "Dort geht es, hier nicht" heißt: Zwei Maschinen bekommen auf dieselbe Frage unterschiedliche Antworten — und `nslookup` auf beiden plus ein Blick in beide hosts-Dateien findet den Unterschied fast jedes Mal. Auf dem Server selbst löst sich `localhost` oder der eigene Maschinenname trivial auf; vom Client aus muss die volle Pipeline mitspielen. Allein diese Asymmetrie erklärt einen bemerkenswerten Anteil der Überraschungen am Deployment-Tag.

> **Diese Serie: Netzwerk-Grundlagen für Delphi-Entwickler**

> 1. IP-Adressen, localhost und Ports 2. **Namen: DNS, die hosts-Datei und DHCP** — Sie sind hier 3. TCP-Verbindungen und Sockets 4. Wie ein Webserver funktioniert: HTTP und WebBroker 5. Web Services: REST, JSON und TMS XData

Das Wichtigste in Kürze

Namen sind die Art, wie Menschen und Konfigurationsdateien Maschinen bezeichnen; die Auflösung ist die Pipeline, die daraus die Zahlen aus Teil 1 macht — und diese Pipeline beherrschen Sie jetzt von Anfang bis Ende.

- Ihre App fragt den **OS-Resolver**; der Resolver prüft seinen **Cache**, dann die **hosts-Datei** (`C:\Windows\System32\drivers\etc\hosts`), dann einen **DNS-Server**. Der erste Treffer gewinnt, und lokale Antworten überschreiben das Netzwerk.
- **DNS** ist ein verteiltes, gecachtes Telefonbuch; ein **A-Record** bildet einen Namen auf eine IPv4-Adresse ab, und jede gecachte Antwort verfällt mit ihrer **TTL** — weshalb DNS-Änderungen Zeit brauchen, um "anzukommen".
- **localhost** ist besonders: reserviert für Loopback, aufgelöst ohne jeden Server, unkaputtbar.
- **DHCP** leased Ihrer Maschine ihre Adresse (und sagt ihr, welchen DNS-Server sie verwenden soll); Leases laufen ab — genau deshalb verrotten hartkodierte Client-IPs über Nacht, und genau deshalb bekommen Server statische oder reservierte Adressen, während alles andere über Namen läuft.
- Mit `ipconfig /all`, `nslookup`, `ipconfig /displaydns` und `ipconfig /flushdns` können Sie jede Stufe davon aus einer normalen Eingabeaufforderung beobachten und zurücksetzen.

> Wenn die Regel aus Teil 1 "Adresse plus Port findet das Programm" war, dann lautet die Regel aus Teil 2: **Sprechen Sie immer in Namen — und kennen Sie die Pipeline, die daraus Nummern macht. Denn in dieser Pipeline wohnen die "mysteriösen" Fehler.**

An diesem Punkt können wir die Maschine und die Tür finden. In Teil 3 öffnen wir sie: was eine TCP-Verbindung eigentlich *ist*, was ein Socket ist und was zwischen "connecting…" und "connected" wirklich passiert.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —