# curl:// for Network Debugging — curlup 2017

2017-03-18

Talk from curlup 2017: how to use curl as a network debugger — telnet replacement, –resolve for SNI debugging, and why -v and -k are essential options.

---

Talk I gave at curlup 2017 in Nürnberg. The slides cover three practical curl techniques that are useful for everyday network debugging.

The slides are available as PDF and the video recording (MP4) is hosted on curl.se.

## Why curl?

curl supports a large number of protocols, is installed by default on almost every OS, and needs no extra binary. Two options that are immediately useful for debugging:

- **`-v`** — verbose output: shows the full request/response exchange including TLS handshake details
- **`-k`** — skip certificate verification: necessary in most enterprise environments with a private CA/PKI

## Telnet replacement

Most modern systems no longer ship with a telnet binary, but telnet-style connection testing is still useful. curl has telnet built in:

```
curl -v telnet://www.google.com:80/
```

To send an actual HTTP request over the telnet connection:

```
echo -e 'GET / HTTP/1.0\n' | curl -v telnet://www.google.com:80/
```

This gives you the raw HTTP response — useful for verifying that a port is reachable and the server is responding before adding TLS or application-layer complexity.

## `--resolve`: poor man's DNS override

Changing the system resolver for a single test is cumbersome. `--resolve` lets you override DNS for a specific host:port combination inline:

```
curl -v --resolve HOST:PORT:ADDRESS http://example.com
```

**Original request** (resolves via DNS):

```
curl -v http://me2digital.online -o /dev/null
# Connected to me2digital.online (192.0.78.25) port 80
```

**Force a specific IP** (e.g. to test behind a CDN or load balancer):

```
curl -v --resolve me2digital.online:80:2607:f8b0:4009:813::2004 http://me
2digital.online -o /dev/null
# Added me2digital.online:80:2607:f8b0:4009:813::2004 to DNS cache
# Connected to me2digital.online (2607:f8b0:4009:813::2004) port 80
```

## Why `--resolve` matters for SNI

For HTTPS, connecting directly to an IP address is not enough. The TLS layer uses the hostname (Server Name Indication) to select the correct virtual host and certificate. Connecting to the IP directly gives a certificate mismatch:

```
curl -v https://54.71.185.21/templates/print-headers.tmpl
# curl: (51) SSL: certificate subject name '*.f0b6.ded-stg2-
aws.openshiftapps.com'
#             does not match target host name '54.71.185.21'
```

With `--resolve`, curl sends the correct SNI hostname while routing to the IP you specify:

```
curl -v --resolve caddy-template-usage-caddy-test001.f0b6.ded-stg2-
aws.openshiftapps.com:443:54.71.185.21 \
  https://caddy-template-usage-caddy-test001.f0b6.ded-stg2-aws.
openshiftapps.com/templates/print-headers.tmpl
# SSL connection using ECDHE-RSA-AES128-GCM-SHA256
```

This is the reliable way to test a specific backend node behind a load balancer or to verify a certificate before DNS has propagated.