ADMIN:DOCS

Windows Server 2025 Core: Setting Up and Managing a DNS Server with PowerShell

Deploy a standalone DNS server on Windows Server 2025 Core without Active Directory: install the role, manage file-backed zones and records via PowerShell, and verify resolution systematically.

Category: Windows Server **Published:** 12.11.2026

Once a freshly deployed Windows Server 2025 Core instance is reachable on the network, a practical operational hurdle quickly surfaces: internal services, test VMs, and storage targets should no longer be addressed by changing or hard-to-remember IP addresses, but reliably by dedicated hostnames. In a pure workgroup environment without Active Directory, there is no central directory service for name resolution—administrators are left either maintaining local `hosts` files across every individual machine or setting up a dedicated DNS service on the network.

Building directly on the `WIN-CORE01` host configured in Windows Server 2025 Core: Installation and Administration via SSH and PowerShell, the server takes on exactly that role. Because Server Core intentionally omits a local desktop interface and graphical tools such as DNS Manager (`dnsmgmt.msc`), the entire deployment and ongoing maintenance take place over SSH directly in Windows PowerShell.

To achieve this, you first install the Windows Server DNS role together with its management tools. This provides the actual `DNS` system service as well as the `DnsServer` PowerShell module. Using that module, you create file-backed primary zones, manage resource records such as `A`, `AAAA`, `CNAME`, and `PTR`, and cleanly separate authoritative answers from recursive forwarding to upstream resolvers. Equally critical is verification: every configuration change must be validated with targeted queries directly against the listening port before other systems on the network

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

rely on the service.

Target Audience and Required Background

This guide is written for administrators with solid operational fundamentals who want to deploy and operate a standalone Windows DNS server without Active Directory from the command line.

Prerequisites:

- Working knowledge of IPv4, IPv6, subnets, and TCP/IP communication
- Familiarity with basic DNS concepts (hostnames, IP mapping, caching, TTL)
- A running Windows Server 2025 Core system (`WIN-CORE01`) with administrative PowerShell access over SSH
- Even if you normally administer Linux systems, you can run every step in this guide directly over SSH in Windows PowerShell without installing a Windows GUI

> □□ **Scope of this guide:** Everything in this article targets a standalone **workgroup DNS server** with file-backed primary zones. Active Directory Domain Services (AD DS), AD-integrated zones, DNSSEC, DNS over HTTPS (DoH), clustering, and graphical tools are intentionally out of scope.

Understanding Windows DNS Server

Before installing the role and creating your first zones, it helps to separate the moving parts: the Windows service, the PowerShell management layer, zone storage, and query resolution logic. Keeping these layers distinct allows you to pinpoint immediately whether a failure stems from the zone database, recursion, or network filtering.

DNS Server Role, System Service, and PowerShell Module

Windows Server 2025 cleanly separates the installable server role (`DNS`), the background service, and the administrative tooling:

- **DNS Server Role (`DNS`):** Provides the server binaries that listen on port 53 (UDP and TCP), hold zones in memory, and answer or forward DNS queries.
- **System Service (`DNS`):** The Windows service `DNS` (`dns.exe`) loads configured zones and server parameters at startup. If this service is stopped, no name resolution takes place on the host—even if the role is installed.
- **PowerShell Module `DnsServer`:** Through the DnsServer module, you manage server settings, zones, and resource records entirely from the command line on Server Core. Including `-IncludeManagementTools` during role installation installs this module automatically.

During operation, `dns.exe` keeps all active zones in RAM and serves queries directly from memory. Changes made via PowerShell take effect immediately in memory and are flushed to disk periodically, on clean service stop, or via an explicit writeback command. Editing a `.dns` zone file manually in a text editor while the `DNS` service is running therefore risks having your edits overwritten when the service next writes its in-memory state to disk.

Primary Zones and File-Backed Storage

A DNS zone represents a contiguous portion of the DNS namespace for which a server is responsible. A **primary zone** is the writable master copy for that namespace—only in a primary zone can you directly add, modify, or delete resource records.

Because `WIN-CORE01` has no Active Directory, Windows DNS Server stores each primary zone as a plain text file in `%SystemRoot%\System32\dns\` (by default `<zonename>.dns`). Unlike Active Directory-integrated zones, there is no multi-master replication across domain controllers; all zone data lives on this single server, as documented in Manage DNS Zones using DNS Server in Windows Server.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

File-Backed Primary Zone

Storage location
: Local zone file (`%SystemRoot%\System32\dns\`)

Requirement
: No domain needed (workgroup-ready)

Write access
: Single primary server (single-master)

Secure updates
: Not available

AD-Integrated Zone

Storage location
: Active Directory database (AD DS)

Requirement
: Domain controllers and AD DS required

Write access
: Any writable DC (multi-master)

Secure updates
: Controllable via AD ACLs

Shared: Zones and resource records are managed through the `DnsServer` module

Authoritative Answers vs. Recursive Queries

A DNS server performs two fundamentally different tasks on the network, and distinguishing them is essential during troubleshooting:

- **Authoritative DNS answers:** When your server hosts `example.test` as a primary zone, it is the binding source of truth for every name inside that zone. If a client queries `web01.example.test`, the server answers directly from its own zone data and sets the `AA` (*Authoritative Answer*) flag. If the name does not exist in the zone, it immediately returns `NXDOMAIN`

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

.

- **Recursive DNS queries:** When the server receives a query for an external name outside its local zones (such as `learn.microsoft.com`), it holds no authoritative data for it. If recursion is enabled, the DNS server resolves the name on the client's behalf—either by forwarding the request to configured **forwarders** or by resolving iteratively starting from **root hints**.

Forward Lookup and Reverse Lookup

In daily administration, you work with two opposite query directions:

- **Forward lookup:** The standard case. A hostname (FQDN such as `app01.example.test`) is resolved to an IPv4 address via `A` records or an IPv6 address via `AAAA` records.
- **Reverse lookup:** The opposite direction. Given a known IP address, the client queries for the corresponding hostname. DNS uses dedicated reverse zones (under `in-addr.arpa` for IPv4) containing `PTR` records. Reverse lookups are routinely needed for log correlation, diagnostic utilities like `nslookup`, and service checks.

Prerequisites and Network Planning

With the architecture of roles, zones, and query paths clear, you can lock down the network prerequisites on `WIN-CORE01` before the first DNS packet arrives. DNS is core infrastructure: if the server's own IP changes or a firewall drops responses, name resolution fails across the entire connected subnet.

System Baseline and Administrative Access

This guide assumes the working baseline established in the preceding Server Core installation guide:

- **Operating system:** Windows Server 2025 Standard Evaluation (Server Core, x64), hostname `WIN-CORE01`, workgroup membership without Active Directory.
- **Management access:** Active OpenSSH connection (port `22/TCP`) running an elevated Windows PowerShell 5.1 session.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

- **Network connectivity:** Direct reachability inside the lab network between the administrative client, `WIN-CORE01`, and the upstream router or resolver.

Static IPv4 Address and Server DNS Client Configuration

A DNS server strictly requires a **static IPv4 address**. If the host obtains its address via DHCP without a reservation and changes IP after a reboot, every DNS client configuration pointing to it breaks. Inspect the current interface configuration first:

```
# Display current IPv4 addresses and interface indices
Get-NetIPAddress -AddressFamily IPv4 |
    Where-Object { $_.InterfaceAlias -notlike "*Loopback*" } |
    Format-Table InterfaceIndex, InterfaceAlias, IPAddress, PrefixLength, PrefixOrigin

# Check the server's current DNS client configuration
Get-DnsClientServerAddress -AddressFamily IPv4

# Pre-flight check: verify TCP reachability to the future upstream forwarder on port 53
Test-NetConnection -ComputerName 192.0.2.1 -Port 53
```

Schematic example output (not executed live on the target system):

```
InterfaceIndex InterfaceAlias IPAddress  PrefixLength PrefixOrigin
-------------- ------------- ---------  ------------ ------------
             6 Ethernet       192.0.2.10           24 Manual

InterfaceAlias               InterfaceIndex AddressFamily ServerAddresses
--------------               -------------- ------------- ---------------
Ethernet                                  6 IPv4          {192.0.2.1}
```

Verify that `PrefixOrigin` reads `Manual` (statically assigned, not `Dhcp`) and note the exact `InterfaceAlias` (such as `Ethernet` or `vEthernet`) for later changes with `Set-DnsClientServerAddress`. If `Test-NetConnection` already shows `TcpTestSucceeded : True` toward `192.0.2.1:53`, you know the lab network permits outbound DNS traffic.

In workgroup environments, also check the active Windows network profile right away: after a fresh install, interfaces often default to `Public`, which prevents standard firewall rules scoped to `Private` or `Domain` from matching. Switch the profile to `Private` if needed:

```
# Check network profile and set it to Private on the internal lab network
Get-NetConnectionProfile | Select-Object InterfaceAlias, NetworkCategory
Set-NetConnectionProfile -InterfaceAlias "Ethernet" -NetworkCategory Private

# Only if IP is still assigned via DHCP: configure static IPv4 192.0.2.10/24
# New-NetIPAddress -InterfaceAlias "Ethernet" -IPAddress "192.0.2.10" -PrefixLength 24 -
DefaultGateway "192.0.2.1"
```

Once the DNS service is operational, you can point `WIN-CORE01`'s own DNS client to itself using Set-DnsClientServerAddress.

> ⚠ **Self-resolution feedback loop:** Do not switch `WIN-CORE01`'s own DNS client setting to `127.0.0.1` or `192.0.2.10` before the DNS service is running and forwarders are configured. Without a working local resolver, the server itself immediately loses external name resolution—breaking Windows Update, PowerShell module lookups, and time sync.

Network Topology and Port Matrix for DNS Operations

Compared to initial server provisioning (which involved Proxmox console access, SMB, or WinRM), a dedicated DNS server exposes a very small set of ports. Clients send queries to `WIN-CORE01`, while the server forwards unknown names outbound to a defined upstream resolver:

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management



Account for the following ports on Windows Defender Firewall as well as any upstream network firewalls:

Port	Protocol	Direction	Purpose in DNS Operations
22	TCP	Inbound (Admin net)	OpenSSH management into PowerShell
53	UDP	Inbound (Clients)	Standard DNS queries and compact responses
53	TCP	Inbound (Clients)	Fallback for large responses (> 512 bytes without EDNS0) and zone transfers
53	UDP/TCP	Outbound (to Forwarder)	Recursive forwarding of external DNS queries
49152–65535	UDP/TCP	Outbound / Return	Ephemeral ports for responses to outbound DNS queries

In practice, administrators frequently open **53/UDP** and forget **53/TCP**. When a DNS response exceeds the UDP payload limit, the server sets the *Truncated* (`TC`) bit and the client automatically retries over TCP port 53. If TCP 53 is blocked, those specific queries fail with

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

seemingly random timeouts.

Test Zone, Address Ranges, and Separation of Resolution Paths

To keep every example in this guide reproducible without colliding with real domains or production networks, we use reserved documentation values throughout:

- **Test zone (RFC 6761):** `example.test` (the `.test` TLD is permanently reserved for testing and is not resolved on the public internet).
- **IPv4 documentation range (RFC 5737):** `192.0.2.0/24` (with `192.0.2.10` assigned to `WIN-CORE01`).
- **IPv6 documentation prefix (RFC 3849):** `2001:db8::/32`.

> **Adapt documentation addresses to your own lab:** `192.0.2.0/24` and `2001:db8::/32` are official documentation prefixes and are not routed on the internet. Replace these placeholders with the actual private IP addresses of your lab or test segment when deploying in your environment.

Plan a strict separation of resolution paths before creating your first zone: `WIN-CORE01` answers all hostnames within `example.test` and its corresponding reverse zone **authoritatively** from local zone files. Everything else (such as public domains) goes through explicitly defined **forwarders** to an upstream resolver. This prevents internal test names from leaking onto the internet and avoids uncontrolled direct root queries.

Installing the DNS Server Role

With the IP address, target subnet, and port requirements confirmed, you install the DNS Server role on `WIN-CORE01` over SSH in an elevated PowerShell session. As Microsoft outlines in Install and Configure DNS Server, installation cleanly separates server binaries from management tools.

Checking Initial State and Installing the Role

First verify whether the role is already present so you avoid unnecessary changes:

```
# Initial state: check installation status of the DNS Server role and RSAT tools
Get-WindowsFeature -Name DNS, RSAT-DNS-Server
```

On a clean system, both features report `Available`. To install the DNS Server role along with the `DnsServer` PowerShell module, run Install-WindowsFeature with `-IncludeManagementTools`:

```
# Install DNS Server role including management tools and DnsServer module
Install-WindowsFeature -Name DNS -IncludeManagementTools
```

- **`-Name DNS`:** Installs the DNS Server role and the `DNS` system service (`dns.exe`).
- **`-IncludeManagementTools`:** Installs the corresponding management tools (`RSAT-DNS-Server`), which include the `DnsServer` PowerShell module. Without this flag, Server Core lacks the PowerShell cmdlets required for configuration.

On Windows Server 2025, this installation does not require a reboot (`Restart Needed: No`). Verify the feature state right after completion:

```
# Confirm both role and management tools report "Installed"
Get-WindowsFeature -Name DNS, RSAT-DNS-Server | Format-Table Name, InstallState
```

Schematic example output (not executed live on the target system):

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
Name            InstallState
----            ------------
DNS                Installed
RSAT-DNS-Server    Installed
```

Checking Service State, Autostart, and PowerShell Module

An installed Windows feature does not guarantee that the service is running with automatic startup or that the cmdlets are ready. Check service status, startup type, and module availability in two focused steps:

```
# Check status and startup type of the DNS service
Get-Service -Name DNS | Select-Object Name, Status, StartType
```

Schematic example output (not executed live on the target system):

```
Name  Status StartType
----  ------ ---------
DNS  Running Automatic
```

The `DNS` service must show `Running` and `Automatic`. If it is stopped or set to `Manual`, fix it explicitly:

```
# Run only if Status is not Running or StartType is not Automatic
Set-Service -Name DNS -StartupType Automatic
Start-Service -Name DNS
```

Next, confirm that the `DnsServer` module is available in your PowerShell session:

```
# Verify the DnsServer module is available
Get-Module -ListAvailable -Name DnsServer

# Optional: check the number of cmdlets provided by the module
(Get-Command -Module DnsServer).Count
```

Checking Port 53 Listeners and Firewall Rules

When the `DNS` service runs, it binds to local interfaces on port 53. Verify at the socket level that both UDP and TCP are listening on port 53:

```
# Check active UDP and TCP listeners on port 53
Get-NetUDPEndpoint -LocalPort 53 | Format-Table LocalAddress, LocalPort, OwningProcess
Get-NetTCPConnection -LocalPort 53 -State Listen | Format-Table LocalAddress, LocalPort,
State, OwningProcess
```

Schematic example output (not executed live on the target system):

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
LocalAddress LocalPort OwningProcess
------------ --------- ------------
::                  53          2480
0.0.0.0             53          2480

LocalAddress LocalPort  State OwningProcess
------------ ---------  ----- ------------
::                  53 Listen          2480
0.0.0.0             53 Listen          2480
```

- **`Get-NetUDPEndpoint -LocalPort 53`:** Shows bound IP addresses (such as `0.0.0.0`, `::`, or specific interface IPs) for standard DNS queries.
- **`Get-NetTCPConnection -LocalPort 53 -State Listen`:** Confirms the server is also listening for TCP fallbacks and zone transfers. The matching `OwningProcess` ID points to `dns.exe`.

Immediately afterward, inspect the active firewall rules for port 53. Querying via `Get-NetFirewallPortFilter` works regardless of OS display language:

```
# Inspect firewall rules for port 53 (UDP/TCP) independently of OS language
Get-NetFirewallPortFilter |
    Where-Object { $_.LocalPort -eq 53 } |
    Get-NetFirewallRule |
    Select-Object DisplayName, Direction, Action, Enabled, Profile
```

Schematic example output (not executed live on the target system):

```
DisplayName             Direction Action Enabled Profile
-----------             --------- ------ ------- -------
DNS (UDP-In)              Inbound  Allow    True Domain, Private
DNS (TCP-In)              Inbound  Allow    True Domain, Private
```

Pay close attention to `Enabled = True`, `Direction = Inbound`, `Action = Allow`, and the active network profile (`Domain`, `Private`, or `Public`). If `WIN-CORE01`'s interface sits in `Public` while the built-in rule only allows `Domain`/`Private`, Windows Defender Firewall silently drops incoming client queries even though the `DNS` service is running.

If a hardened Server Core image lacks matching port-53 rules—or if you want to restrict access strictly to your client subnet `192.0.2.0/24`—create explicit UDP and TCP rules:

```
# Only if no matching rule exists: allow DNS exclusively from lab subnet 192.0.2.0/24
New-NetFirewallRule -DisplayName "DNS-Server (UDP-In, Lab-Subnet)" -Direction Inbound -
Protocol UDP -LocalPort 53 -RemoteAddress "192.0.2.0/24" -Profile Private -Action Allow
New-NetFirewallRule -DisplayName "DNS-Server (TCP-In, Lab-Subnet)" -Direction Inbound -
Protocol TCP -LocalPort 53 -RemoteAddress "192.0.2.0/24" -Profile Private -Action Allow
```

> ⚠ **An installed role is not proof of function:** Completing `Install-WindowsFeature`, `Get-Service`, and the port-53 socket check only confirms that the base service is up. It does not prove that `WIN-CORE01` can serve authoritative answers or forward external names—both local zones and forwarders are still missing.

Cleanly Removing the Role (Optional Lab Rollback)

If you ever need to tear down a lab setup on `WIN-CORE01`, remove the DNS Server role with `Uninstall-WindowsFeature`. Note that Windows preserves the physical zone files under

`%SystemRoot%\System32\dns\` during uninstallation so data is not lost accidentally; archive or clean them manually if needed:

```
# Cleanly uninstall the DNS Server role and management tools
Uninstall-WindowsFeature -Name DNS -IncludeManagementTools -Restart
```

Creating a Forward Lookup Zone

The `DNS` service is listening on port 53 and the `DnsServer` module is ready, yet the server owns no namespace. Next, you create your first authoritative forward lookup zone (`example.test`) so `WIN-CORE01` can answer queries for internal hostnames locally and authoritatively.

Choosing a Zone Name and Understanding Zone Types

Follow a clear rule for internal test and lab zones: never use public production domains you do not own, and avoid legacy pseudo-TLDs like `.local` (reserved for Multicast DNS / mDNS under RFC 6762). The `.test` top-level domain is permanently reserved for testing by RFC 6761 and never collides with public DNS or local mDNS traffic.

Windows Server DNS distinguishes three primary zone types:

- **Primary zone (`Primary`):** Holds the single writable master copy of zone data. Only in a primary zone can you create or modify resource records.
- **Secondary zone (`Secondary`):** Holds a read-only copy pulled regularly via zone transfer (`AXFR` / `IXFR`) from a primary DNS server for redundancy or load distribution.
- **Stub zone (`Stub`):** Stores only the records needed to locate authoritative name servers (`SOA` , `NS` , and glue `A` / `AAAA` records) for delegated child zones.

Property	Primary Zone	Secondary Zone	Stub Zone

Write access	Yes (master)	No (read-only)	No (referrals only)
Stored records	All zone records	All zone records	Only `SOA`, `NS`, glue `A` / `AAAA`
Data source	Local file (or AD)	Zone transfer (`AXFR` / `IXFR`)	Name server list transfer
Typical use case	Main server such as `WIN-CORE01`	Second DNS for redundancy	Fast resolution of delegated subzones

For our standalone workgroup server, we create a **file-backed primary forward lookup zone** .

Checking Existing Zones

Before creating the new zone, inspect what zones already exist on the freshly installed server with Get-DnsServerZone:

```
# Initial state: list existing DNS zones on the server
Get-DnsServerZone
```

On a brand-new server, only automatically created built-in helper zones (such as `TrustAnchors` or loopback reverse zones, depending on OS configuration) appear—there is no user-defined forward zone yet.

Creating a Primary File-Backed Zone with PowerShell

To create the standalone zone `example.test`, use Add-DnsServerPrimaryZone with `-ZoneFile` and `-DynamicUpdate "None"`. Because this is a workgroup server, do not pass Active Directory parameters such as `-ReplicationScope`:

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# Create primary file-backed forward lookup zone without dynamic updates
Add-DnsServerPrimaryZone -Name "example.test" -ZoneFile "example.test.dns" -DynamicUpdate
"None" -PassThru
```

- `-Name "example.test"`**:** Sets the DNS domain name of the zone.
- `-ZoneFile "example.test.dns"`**:** Instructs Windows Server DNS to store the zone as `%SystemRoot%\System32\dns\example.test.dns`.
- `-DynamicUpdate "None"`**:** Disables dynamic client updates so only administrators can add or modify records via PowerShell.
- `-PassThru`**:** Returns the newly created zone object immediately for verification.

> ⚠ **No AD-integrated zones without AD DS:** Passing `-ReplicationScope "Domain"` or `-DynamicUpdate "Secure"` on a workgroup server fails immediately because there is no Active Directory domain partition or Kerberos trust to store or authenticate the zone.

Understanding Dynamic Update Modes

The `-DynamicUpdate` parameter controls whether clients may register or overwrite records automatically:

- `None` **(Recommended here):** Completely disables dynamic updates. In a workgroup environment without Kerberos authentication, this is the safest default because it prevents unauthorized clients from overwriting records.
- `NonsecureAndSecure`**:** Accepts unauthenticated dynamic updates from any host able to reach port 53. Only use this in isolated, trusted test segments where automatic registration is explicitly required.
- `Secure`**:** Requires Active Directory Kerberos authentication and is available **exclusively** for AD-integrated zones.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Inspecting Zone Properties and the Zone File

Verify both the logical zone configuration in the service and the physical file on disk immediately after creation:

```
# Inspect detailed properties of the new zone
Get-DnsServerZone -Name "example.test" | Format-List ZoneName, ZoneType, IsAutoCreated,
IsDsIntegrated, DynamicUpdate, ZoneFile
```

Schematic example output (not executed live on the target system):

```
ZoneName       : example.test
ZoneType       : Primary
IsAutoCreated  : False
IsDsIntegrated : False
DynamicUpdate  : None
ZoneFile       : example.test.dns
```

The key values are `ZoneType : Primary`, `IsDsIntegrated : False`, and `DynamicUpdate : None`, confirming that the zone is maintained as a local file independent of Active Directory.

Now inspect the generated zone file on disk. Because the `DNS` service buffers changes in memory and writes them to disk periodically, on service stop, or after an explicit `dnscmd /ZoneWriteBack example.test`, flush the buffer before reading the file directly after fresh edits:

```
# Optional: flush in-memory zone buffer to example.test.dns and inspect its contents
```

```
dnscmd.exe /ZoneWriteBack example.test
Get-Content -Path "$env:SystemRoot\System32\dns\example.test.dns"
```

A freshly created zone file follows the standard text format (schematic example):

```
;
;  Database file example.test.dns for example.test zone.
;      Zone version:  1
;

@                       IN  SOA win-core01.  hostmaster.  (
                                1            ; serial number
                                900          ; refresh
                                600          ; retry
                                86400        ; expire
                                3600       ) ; default TTL

;
;  Zone NS records
;

@                       NS  win-core01.
win-core01.             A   192.0.2.10
```

Note two important syntax details in this file:

1. **Trailing dot on FQDNs:** Names ending in a dot (`win-core01.`) are absolute; names without a trailing dot have the zone name appended automatically by the parser.
2. **Name server glue entry:** Windows often populates the server's short hostname (`win-core01.`) as the initial `NS` record. When you want a clean FQDN within the zone (such as `win-core01.example.test.`), align the `NS` and `SOA` records accordingly so clients can resolve the authoritative name server inside the zone.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Understanding and Checking `SOA` and `NS` Records

Whenever you create a primary zone, Windows automatically generates one `SOA` (*Start of Authority*) record and at least one `NS` (*Name Server*) record at the zone apex (`@`). Inspect all records with `Get-DnsServerResourceRecord`:

```
# List all initial resource records in the zone
Get-DnsServerResourceRecord -ZoneName "example.test"
```

Schematic example output (not executed live on the target system):

```
HostName  RecordType Type       Timestamp           TimeToLive      RecordData
--------  ---------- ----       ---------           ----------      ----------
@         NS         2          0                   01:00:00        win-core01.
@         SOA        6          0                   01:00:00        [1][win-
core01.][hostmaster.]
```

It is worth inspecting the `SOA` fields with `Format-List`, because `MinimumTimeToLive` in particular governs negative caching:

```
# Display detailed SOA properties
(Get-DnsServerResourceRecord -ZoneName "example.test" -RRType SOA).RecordData | Format-
List
```

Schematic example output (not executed live on the target system):

```
SerialNumber      : 1
PrimaryServer     : win-core01.
ResponsiblePerson : hostmaster.
RefreshInterval   : 00:15:00
RetryDelay        : 00:10:00
ExpireLimit       : 1.00:00:00
MinimumTimeToLive : 01:00:00
```

- **`PrimaryServer`:** The hostname of the authoritative primary DNS server for the zone.
- **`ResponsiblePerson`:** Contact email in DNS notation (the `@` sign is replaced by a dot, e.g., `hostmaster.example.test.`).
- **`SerialNumber`:** Incremental zone version. Windows DNS increments this number automatically on every change so secondary servers know when to pull updates.
- **`RefreshInterval`, `RetryDelay`, `ExpireLimit`:** Timers (by default 15 minutes, 10 minutes, 1 day) controlling how often secondary servers poll for updates and when stale copies expire.
- **`MinimumTimeToLive`:** Default negative-caching TTL (often 1 hour / `01:00:00`) specifying how long resolvers may cache `NXDOMAIN` responses.

To update `PrimaryServer` and `ResponsiblePerson` to clean FQDNs inside your zone, clone the existing `SOA` record in memory and write it back with `Set-DnsServerResourceRecord`:

```
# Read existing SOA record and create a mutable copy
$oldSoa = Get-DnsServerResourceRecord -ZoneName "example.test" -RRType SOA
$newSoa = $oldSoa.Clone()

# Set authoritative server and contact address to FQDNs in example.test
$newSoa.RecordData.PrimaryServer = "win-core01.example.test."
```

```
$newSoa.RecordData.ResponsiblePerson = "hostmaster.example.test."

# Commit updated SOA record to the zone
Set-DnsServerResourceRecord -NewInputObject $newSoa -OldInputObject $oldSoa -ZoneName
"example.test"
```

Verifying Authoritative Zone Responses

Seeing the zone in `Get-DnsServerZone` is only the server's internal view. Verify with an actual DNS query against `127.0.0.1` that the service answers authoritatively over port 53:

```
# Query SOA and NS records directly over the local DNS socket
Resolve-DnsName -Name "example.test" -Type SOA -Server 127.0.0.1 -DnsOnly -NoHostsFile
Resolve-DnsName -Name "example.test" -Type NS -Server 127.0.0.1 -DnsOnly -NoHostsFile
```

Schematic example output for `Type SOA` (not executed live on the target system):

```
Name         Type   TTL   Section    PrimaryServer             NameAdministrator
SerialNumber
----         ----   ---   -------    -------------             -----------------
------------
example.test SOA    3600  Answer     win-core01.example.test   hostmaster.example.test  2
```

In the `Resolve-DnsName` output, a valid authoritative answer returns directly in `Section : Answer` with `Type : SOA` or `Type : NS` and your configured data (`win-core01.example.test`), without forwarding externally. Note also that `SerialNumber` incremented cleanly to `2` after the `SOA` update.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

> ⚠ **Common zone creation pitfalls:** If `Add-DnsServerPrimaryZone` fails in a workgroup, the cause is almost always a missing `-ZoneFile` parameter or an AD-only argument such as `-ReplicationScope` or `-DynamicUpdate "Secure"`. If the zone shows up in `Get-DnsServerZone` but `Resolve-DnsName` times out, either the firewall is blocking port 53 or the `DNS` service is not listening on the queried IP.

Outlook: Adding a Secondary DNS Zone for Redundancy

In a pure workgroup environment without Active Directory replication, you initially have a single writable master (`WIN-CORE01`). If that server reboots during maintenance, internal name resolution pauses unless a second server is present. Once you add a second Server Core host (for example `WIN-CORE02` at `192.0.2.11`), configure it as a read-only **secondary zone** that pulls updates via zone transfer (`AXFR` / `IXFR` on TCP port 53) from `WIN-CORE01`:

```
# On secondary server WIN-CORE02 (192.0.2.11): create secondary zone
Add-DnsServerSecondaryZone -Name "example.test" -ZoneFile "example.test.dns" -
MasterServers "192.0.2.10" -PassThru

# On primary server WIN-CORE01 (192.0.2.10): allow transfers exclusively to 192.0.2.11
Set-DnsServerPrimaryZone -Name "example.test" -SecureSecondaries "TransferToSecureServers"
-SecondaryServers "192.0.2.11" -Notify "NotifyServers" -NotifyServers "192.0.2.11"
```

This keeps `WIN-CORE01` as the single source of truth while `WIN-CORE02` is notified automatically whenever the `SOA` serial number increments.

Managing DNS Resource Records

With the active forward lookup zone `example.test`, the DNS server knows its namespace—but so far it only contains `SOA` and `NS` metadata. To make actual hosts and services resolvable for clients, you now populate the zone with resource records.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Separating Server Mutations from Queries

Administrators new to the `DnsServer` module often conflate two distinct layers:

- **Zone database mutations (`Add-`, `Set-`, `Remove-DnsServerResourceRecord*`):** These cmdlets write directly to the zone, increment the `SOA` serial number, and change what every client on the network receives.
- **Read-only checks (`Get-DnsServerResourceRecord` and `Resolve-DnsName`):** `Get-DnsServerResourceRecord` reads the stored zone inventory on the server, while `Resolve-DnsName` sends a genuine DNS query over the network socket like any client. Neither command modifies zone data.

`A` Records (IPv4 Host Entries)

An `A` **record** (*Address Record*) maps a hostname within the zone to an IPv4 address. Required inputs are the hostname (`-Name`), target zone (`-ZoneName`), and a valid IPv4 address (`-IPv4Address`). Optionally, `-TimeToLive` (`TTL` in `hh:mm:ss` format) defines how long clients may cache the answer.

To register `web01.example.test` at `192.0.2.20` with a one-hour TTL, use Add-DnsServerResourceRecordA:

```
# Create A record for web01.example.test pointing to 192.0.2.20
Add-DnsServerResourceRecordA -Name "web01" -ZoneName "example.test" -IPv4Address
"192.0.2.20" -TimeToLive 01:00:00
```

Adding `-CreatePtr` also creates the matching reverse `PTR` record—**only** if the corresponding reverse lookup zone already exists on the server (covered in the next section).

Verify the new record both in the zone inventory and via a live query:

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# Check zone inventory
Get-DnsServerResourceRecord -ZoneName "example.test" -Name "web01" -RRType "A"

# Test live DNS query against the local service
Resolve-DnsName -Name "web01.example.test" -Type A -Server 127.0.0.1 -DnsOnly -NoHostsFile
```

Schematic example output of `Resolve-DnsName` (not executed live on the target system):

```
Name                      Type   TTL   Section    IPAddress
----                      ----   ---   -------    ---------
web01.example.test        A      3600  Answer     192.0.2.20
```

`AAAA` Records (IPv6 Host Entries)

An **`AAAA` record** (*Quad-A Record*) performs the same mapping for 128-bit IPv6 addresses using `-IPv6Address`. A single hostname can hold both an `A` and an `AAAA` record simultaneously for dual-stack operation.

Add the IPv6 entry for `web01.example.test` using `Add-DnsServerResourceRecordAAAA`:

```
# Create AAAA record for web01.example.test using documentation prefix 2001:db8::/32
Add-DnsServerResourceRecordAAAA -Name "web01" -ZoneName "example.test" -IPv6Address
"2001:db8::20" -TimeToLive 01:00:00
```

Verify both the zone entry and the live IPv6 query:

```
# Verify AAAA entry in the zone and via DNS query
Get-DnsServerResourceRecord -ZoneName "example.test" -Name "web01" -RRType "AAAA"
Resolve-DnsName -Name "web01.example.test" -Type AAAA -Server 127.0.0.1 -DnsOnly -
NoHostsFile
```

Schematic example output of `Resolve-DnsName` (not executed live on the target system):

```
Name                      Type   TTL   Section    IPAddress
----                      ----   ---   -------    ---------
web01.example.test        AAAA   3600  Answer     2001:db8::20
```

`CNAME` Records (Aliases)

A **`CNAME` record** (*Canonical Name*) points an alias hostname to another canonical FQDN (`-HostNameAlias`). This is ideal when one host serves multiple application names (for example, `portal.example.test` pointing to `web01.example.test`): if `web01` 's IP changes later, you only update `web01` 's `A` / `AAAA` record and all aliases follow automatically.

> ⚠ **DNS standard rules for `CNAME` :** A hostname that has a `CNAME` record cannot coexist with any other resource records (`A` , `AAAA` , `MX` , `TXT` , or `NS`)—and a `CNAME` is strictly invalid at the zone apex (`example.test` itself), because `SOA` and `NS` already reside there.

Create the alias with Add-DnsServerResourceRecordCName and verify the resolution chain:

```
# Create CNAME alias portal.example.test -> web01.example.test
Add-DnsServerResourceRecordCName -Name "portal" -ZoneName "example.test" -HostNameAlias
"web01.example.test" -TimeToLive 01:00:00

# Query returns the CNAME first and then resolves the target host
Resolve-DnsName -Name "portal.example.test" -Server 127.0.0.1 -DnsOnly -NoHostsFile
```

Schematic example output (not executed live on the target system):

```
Name                      Type   TTL   Section    NameHost / IPAddress
----                      ----   ---   -------    --------------------
portal.example.test       CNAME  3600  Answer     web01.example.test
web01.example.test        A      3600  Answer     192.0.2.20
```

The response shows both hops together: `portal.example.test` resolves via `CNAME` to `web01.example.test`, and the DNS server includes the matching `A` record in the same answer.

`PTR` Records (Reverse Mapping)

A **`PTR` record** (*Pointer Record*) maps an IP address back to a canonical hostname (`-PtrDomainName`).

Crucially, while `A`, `AAAA`, and `CNAME` live in your forward lookup zone `example.test`, a `PTR` record **always** lives in a separate reverse lookup zone (such as `2.0.192.in-addr.arpa` for `192.0.2.20`). The `-Name` parameter is only the host portion (`"20"`), and `-PtrDomainName` is the full FQDN (`"web01.example.test."`).

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# Example call (requires reverse zone 2.0.192.in-addr.arpa to exist first)
Add-DnsServerResourceRecordPtr -Name "20" -ZoneName "2.0.192.in-addr.arpa" -PtrDomainName
"web01.example.test." -TimeToLive 01:00:00
```

In the next section, we create that reverse zone and verify `PTR` resolution end-to-end.

Safely Updating Existing Resource Records

In the `DnsServer` module, `Set-DnsServerResourceRecord` does not blindly overwrite by name alone—you pass both the unchanged original object (`-OldInputObject`) and a modified copy (`-NewInputObject`). This prevents accidentally overwriting the wrong entry when multiple records share the same name (such as round-robin `A` records).

□□ Practical example:

Suppose `web01.example.test` is moving from `192.0.2.20` to `192.0.2.25` and you want to lower its TTL to 5 minutes:

```
# 1. Read the exact existing A record
$oldRecord = Get-DnsServerResourceRecord -ZoneName "example.test" -Name "web01" -RRType
"A"

# 2. Clone in memory and update values
$newRecord = $oldRecord.Clone()
$newRecord.RecordData.IPv4Address = [System.Net.IPAddress]::Parse("192.0.2.25")
$newRecord.TimeToLive = [System.TimeSpan]::FromMinutes(5)

# 3. Apply the change on the DNS server
Set-DnsServerResourceRecord -OldInputObject $oldRecord -NewInputObject $newRecord -
ZoneName "example.test"

# 4. Verify via a live query against the server
```

```
Resolve-DnsName -Name "web01.example.test" -Type A -Server 127.0.0.1 -DnsOnly -NoHostsFile
```

When multiple `A` records exist under the same hostname, filter `$oldRecord` with `Where-Object` before calling `.Clone()`—otherwise `$oldRecord` is an array and `.Clone()` fails. The following example updates only the second round-robin IP for `app01.example.test`:

```
# Two A records under the same hostname (DNS round robin)
Add-DnsServerResourceRecordA -Name "app01" -ZoneName "example.test" -IPv4Address
"192.0.2.30" -TimeToLive 00:05:00
Add-DnsServerResourceRecordA -Name "app01" -ZoneName "example.test" -IPv4Address
"192.0.2.31" -TimeToLive 00:05:00

# Filter strictly to 192.0.2.31 before cloning and updating
$oldApp = Get-DnsServerResourceRecord -ZoneName "example.test" -Name "app01" -RRType "A" |
    Where-Object { $_.RecordData.IPv4Address.IPAddressToString -eq "192.0.2.31" }
$newApp = $oldApp.Clone()
$newApp.RecordData.IPv4Address = [System.Net.IPAddress]::Parse("192.0.2.32")

Set-DnsServerResourceRecord -OldInputObject $oldApp -NewInputObject $newApp -ZoneName
"example.test"
```

TTL Planning Ahead of Planned Server Migrations

With a `01:00:00` (3,600-second) TTL, clients and caching resolvers can hold an old `A` record for up to a full hour. Lower the affected record's TTL to `00:05:00` (300 seconds) **at least one full old TTL period before the cutover**, perform the IP switch, and raise the TTL back to the standard value only after verification succeeds.

Safely Removing Obsolete Records

Deleting a resource record takes effect immediately: `WIN-CORE01` begins returning `NXDOMAIN` for that name (while clients with cached entries keep using the old address until TTL expiry).

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Always run Remove-DnsServerResourceRecord with `-WhatIf` first and narrow the target with `-RecordData`:

```
# Step 1: Dry-run deletion without modifying the server
Remove-DnsServerResourceRecord -ZoneName "example.test" -RRType "CNAME" -Name "portal" -
RecordData "web01.example.test." -WhatIf

# Step 2: Execute after verifying target (PowerShell prompts for confirmation)
Remove-DnsServerResourceRecord -ZoneName "example.test" -RRType "CNAME" -Name "portal" -
RecordData "web01.example.test."

# Step 3: Confirm the record is gone
Get-DnsServerResourceRecord -ZoneName "example.test" -Name "portal" -ErrorAction
SilentlyContinue
```

Avoid blanket pipeline deletions with `-Force` unless you explicitly filter by both `-RRType` and `-RecordData`: accidentally removing an `NS` or primary `A` record immediately breaks dependent services and aliases.

Additional Useful Record Types: `SRV`, `TXT`, and `MX`

Alongside `A`, `AAAA`, `CNAME`, and `PTR`, three other record types appear regularly on standalone workgroup DNS servers:

- **`TXT` records:** Store arbitrary text metadata (such as internal service notes or verification strings).
- **`SRV` records:** Map a service name and protocol (for instance `_https._tcp`) to a target host, port, priority, and weight.
- **`MX` records:** Designate a domain's mail exchanger (`MailExchange`) with a preference value.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# Add TXT record for internal zone documentation
Add-DnsServerResourceRecord -ZoneName "example.test" -Name "_meta" -Txt -DescriptiveText
"admindocs-lab-zone" -TimeToLive 01:00:00

# Add SRV record for an internal HTTPS service on web01.example.test (port 443)
Add-DnsServerResourceRecord -ZoneName "example.test" -Name "_https._tcp" -Srv -DomainName
"web01.example.test." -Port 443 -Priority 10 -Weight 100 -TimeToLive 01:00:00

# Add MX record with preference 10 (if an internal mail relay exists)
Add-DnsServerResourceRecordMX -ZoneName "example.test" -Name "." -MailExchange
"mail01.example.test." -Preference 10 -TimeToLive 01:00:00
```

Review the full zone inventory after major edits:

```
# Display full zone inventory in table format
Get-DnsServerResourceRecord -ZoneName "example.test" | Format-Table HostName, RecordType,
TimeToLive, RecordData
```

Schematic example output (not executed live on the target system):

```
HostName     RecordType TimeToLive RecordData
--------     ---------- ---------- ----------
@            NS         01:00:00   win-core01.example.test.
@            SOA        01:00:00   [5][win-core01.example.test.][hostmaster.example.test.]
_https._tcp  SRV        01:00:00   [10][100][443][web01.example.test.]
_meta        TXT        01:00:00   admindocs-lab-zone
app01        A          00:05:00   192.0.2.30
app01        A          00:05:00   192.0.2.32
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
web01        A          00:05:00   192.0.2.25
web01        AAAA       01:00:00   2001:db8::20
```

After running `dnscmd.exe /ZoneWriteBack example.test`, the physical zone file `%SystemRoot%\System32\dns\example.test.dns` reflects this exact state in plain text (schematic example):

```
;
;  Database file example.test.dns for example.test zone.
;      Zone version:  5
;

@                       IN  SOA win-core01.example.test.  hostmaster.example.test.  (
                                5            ; serial number
                                900          ; refresh
                                600          ; retry
                                86400        ; expire
                                3600       ) ; default TTL

;
;  Zone NS records
;

@                       NS      win-core01.example.test.

;
;  Zone records
;

_https._tcp             SRV     10 100 443 web01.example.test.
_meta                   TXT     ( "admindocs-lab-zone" )
app01                   300 A   192.0.2.30
app01                   300 A   192.0.2.32
web01                   300 A   192.0.2.25
web01                   AAAA    2001:db8::20
win-core01              A       192.0.2.10
```

Notice two details in the written zone file:

- **Non-default TTLs:** Records with a custom TTL (`300` seconds for `app01` and `web01`) include the value explicitly before the record type (`300 A`), while records matching the zone default (`3600` seconds) omit it.
- **Automatic `SOA` serial number:** Every PowerShell edit increments `serial number` automatically. If you ever edit the zone file manually in an emergency, you must increment this number yourself (often using `YYYYMMDDnn`, provided it is numerically greater than the previous value) so secondary servers and caches pick up the change.

Configuring a Reverse Lookup Zone

At the end of the previous section, adding a `PTR` record hit a natural boundary: until `WIN-CORE01` has a matching reverse lookup zone, both `Add-DnsServerResourceRecordPtr` and `-CreatePtr` on `Add-DnsServerResourceRecordA` have nowhere to write. Now we build that reverse resolution cleanly.

Purpose of Reverse Name Resolution in Operations

At first glance, reverse lookup may seem optional because clients connect to services via hostnames (`A` / `AAAA`). In daily operations and log analysis, however, you constantly encounter raw IP addresses—and that is where reverse DNS provides clarity:

- **Diagnostic tools (`nslookup`):** Whenever an administrator queries a DNS server with `nslookup`, the tool first tries to reverse-resolve the DNS server's own IP address. Without a `PTR` record for `192.0.2.10`, `nslookup` prints `Server: UnKnown`, which is frequently misread as a DNS outage.
- **Logs and monitoring:** Firewalls, SSH logs, proxies, and monitoring agents record source IPs first. Consistent `PTR` records let you map events directly to `web01.example.test` rather than anonymous numbers.
- **Forward-Confirmed Reverse DNS (FCrDNS):** Many services and security checks verify that a `PTR` record resolves to a hostname whose `A` record points back to the exact same IP address.

Why `in-addr.arpa` Reverses the Octets

DNS is a strict hierarchy where the broadest level sits on the right (`.` , then `.test` , then `example.test`) and the most specific host label sits on the left (`web01`). An IPv4 address works in the opposite order: the network prefix sits on the left (`192.0.2`) and the host portion sits on the right (`.25`).

To allow hierarchical delegation by subnet, the `in-addr.arpa` domain reverses the octet order:

- The IPv4 subnet **`192.0.2.0/24`** becomes the zone name **`2.0.192.in-addr.arpa`** .
- The full reverse name for **`192.0.2.25`** becomes **`25.2.0.192.in-addr.arpa`** .
- Inside `2.0.192.in-addr.arpa` , a `PTR` record only needs the remaining host octet `"25"` as `-Name` .

Checking Existing Zones and Creating an IPv4 Reverse Lookup Zone

First verify with `Get-DnsServerZone` whether a reverse zone for `192.0.2.0/24` already exists on `WIN-CORE01` :

```
# Initial state: check existing reverse lookup zones
Get-DnsServerZone | Where-Object { $_.IsReverseLookupZone -eq $true }
```

On a freshly provisioned server, this returns no user-defined zones (only built-in loopback/link-local helper zones if enabled).

To create the file-backed IPv4 reverse zone, use Add-DnsServerPrimaryZone again—this time with `-NetworkID` instead of `-Name` . Windows Server DNS derives the `in-addr.arpa` zone name automatically:

```
# Create file-backed IPv4 reverse lookup zone for 192.0.2.0/24
Add-DnsServerPrimaryZone -NetworkID "192.0.2.0/24" -ZoneFile "2.0.192.in-addr.arpa.dns" -
DynamicUpdate "None" -PassThru
```

- **`-NetworkID "192.0.2.0/24"`:** Passes the IPv4 subnet in CIDR notation; the cmdlet derives `2.0.192.in-addr.arpa` automatically.
- **`-ZoneFile "2.0.192.in-addr.arpa.dns"`:** Stores zone data independently of Active Directory in `%SystemRoot%\System32\dns\2.0.192.in-addr.arpa.dns`.
- **`-DynamicUpdate "None"`:** Keeps the reverse zone under manual administrative control so clients cannot inject arbitrary `PTR` records.

Verify the new reverse zone's properties immediately:

```
# Verify properties of the new reverse zone
Get-DnsServerZone -Name "2.0.192.in-addr.arpa" | Format-List ZoneName, ZoneType,
IsReverseLookupZone, IsDsIntegrated, DynamicUpdate, ZoneFile
```

Schematic example output (not executed live on the target system):

```
ZoneName            : 2.0.192.in-addr.arpa
ZoneType            : Primary
IsReverseLookupZone : True
IsDsIntegrated      : False
DynamicUpdate       : None
ZoneFile            : 2.0.192.in-addr.arpa.dns
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Windows also creates initial `SOA` and `NS` records in the reverse zone. Align `PrimaryServer` and `ResponsiblePerson` to `win-core01.example.test.` just as you did for the forward zone.

Subnet Size and `PTR` Uniqueness

Make sure `-NetworkID` matches your actual subnet boundary: a `/16` prefix such as `192.0.0.0/16` creates `0.192.in-addr.arpa`, where `PTR` names must use two octets (`"25.2"` for `192.0.2.25`), whereas a `/24` prefix creates `2.0.192.in-addr.arpa` with a single host octet (`"25"`). Also avoid assigning multiple `PTR` records to the same IP address unless specifically required—diagnostic tools and log parsers will otherwise return inconsistent hostnames.

Creating `PTR` Records for the Server and Existing Hosts

With `2.0.192.in-addr.arpa` active, add `PTR` records for existing IPv4 hosts—especially for `WIN-CORE01` (`192.0.2.10`) itself and for `web01.example.test` (`192.0.2.25`, updated in the previous section):

```
# Create PTR record for WIN-CORE01 (192.0.2.10)
Add-DnsServerResourceRecordPtr -Name "10" -ZoneName "2.0.192.in-addr.arpa" -PtrDomainName
"win-core01.example.test." -TimeToLive 01:00:00

# Create PTR record for web01.example.test (192.0.2.25)
Add-DnsServerResourceRecordPtr -Name "25" -ZoneName "2.0.192.in-addr.arpa" -PtrDomainName
"web01.example.test." -TimeToLive 01:00:00
```

Always provide the **full FQDN with a trailing dot** (`"web01.example.test."`) for `-PtrDomainName`. Omitting the domain suffix produces incomplete hostnames in reverse responses.

Using `-CreatePtr` for Future Hosts

Once a matching reverse lookup zone exists, you can create both forward and reverse records in a single step for new IPv4 hosts. Passing `-CreatePtr` to `Add-DnsServerResourceRecordA` tells Windows DNS to locate the matching `in-addr.arpa` zone and insert the `PTR` record

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

automatically:

```
# Create forward A record for app01.example.test together with its PTR record
Add-DnsServerResourceRecordA -Name "app01" -ZoneName "example.test" -IPv4Address
"192.0.2.30" -CreatePtr -TimeToLive 01:00:00
```

Keep one operational limitation in mind: when you later change an existing `A` record's IP via `Set-DnsServerResourceRecord` or delete it with `Remove-DnsServerResourceRecord`, the `PTR` record is **not updated or removed automatically**. You must clean up orphaned `PTR` records explicitly in the reverse zone.

A short PowerShell comparison between `example.test` and `2.0.192.in-addr.arpa` spots orphaned `PTR` entries immediately:

```
# Collect all IPv4 addresses currently defined in example.test
$forwardIps = Get-DnsServerResourceRecord -ZoneName "example.test" -RRType "A" |
    ForEach-Object { $_.RecordData.IPv4Address.IPAddressToString }

# Check each PTR record to see whether a matching A record still exists
Get-DnsServerResourceRecord -ZoneName "2.0.192.in-addr.arpa" -RRType "PTR" |
    ForEach-Object {
        $ip = "192.0.2.$($_.HostName)"
        [PSCustomObject]@{
            IP         = $ip
            PtrTarget  = $_.RecordData.PtrDomainName
            HasARecord = ($forwardIps -contains $ip)
        }
    } | Format-Table -AutoSize
```

Rows showing `HasARecord : False` highlight orphaned `PTR` entries (or IPs whose `A` record intentionally lives in a different zone), ready for review with `Remove-DnsServerResourceRecord -`

`WhatIf`.

Verifying Reverse DNS Queries

Check the reverse zone inventory first and then run live reverse queries against `127.0.0.1`:

```
# 1. Inspect reverse zone PTR records
Get-DnsServerResourceRecord -ZoneName "2.0.192.in-addr.arpa" -RRType "PTR"

# 2. Reverse query using the IPv4 address directly with Resolve-DnsName
Resolve-DnsName -Name "192.0.2.25" -Server 127.0.0.1 -DnsOnly -NoHostsFile

# 3. Explicit PTR query using in-addr.arpa notation
Resolve-DnsName -Name "25.2.0.192.in-addr.arpa" -Type PTR -Server 127.0.0.1 -DnsOnly -
NoHostsFile
```

When passed an IPv4 address, `Resolve-DnsName` automatically translates `192.0.2.25` into a `PTR` query for `25.2.0.192.in-addr.arpa`. Schematic example output:

```
Name                           Type   TTL   Section    NameHost
----                           ----   ---   -------    --------
25.2.0.192.in-addr.arpa        PTR    3600  Answer     web01.example.test
```

Cross-check with `nslookup` (once `192.0.2.10` has its `PTR` record):

```
nslookup 192.0.2.25 192.0.2.10
```

Schematic example output with consistent forward and reverse DNS:

```
Server:  win-core01.example.test
Address:  192.0.2.10

Name:    web01.example.test
Address:  192.0.2.25
```

If `Server:` still reads `UnKnown`, the `PTR` record for the queried DNS server IP is missing (note that querying `127.0.0.1` looks up `127.0.0.1`, whereas querying `192.0.2.10` resolves cleanly to `win-core01.example.test`).

How IPv6 Reverse DNS (`ip6.arpa`) Works

In Section 6 we created an `AAAA` record (`2001:db8::20`) for dual-stack readiness. IPv6 reverse DNS follows the exact same principle as IPv4, except it reverses **nibble-by-nibble** (each 4-bit hex digit) under `ip6.arpa`.

For the `/32` prefix `2001:0db8::/32`, the eight hex digits are reversed and separated by dots:

- Full prefix notation: `2001:0db8`
- Reversed nibble zone: **`8.b.d.0.1.0.0.2.ip6.arpa`**

`Add-DnsServerPrimaryZone -NetworkID "2001:db8::/32" -ZoneFile "..."` handles the nibble reversal automatically. For `web01` (`2001:db8::20`), the configuration looks like this:

```
# Optional for dual-stack networks: create IPv6 reverse zone for 2001:db8::/32
Add-DnsServerPrimaryZone -NetworkID "2001:db8::/32" -ZoneFile
"8.b.d.0.1.0.0.2.ip6.arpa.dns" -DynamicUpdate "None" -PassThru
```

```
# Create PTR record for 2001:db8::20 (24 reversed host nibbles)
Add-DnsServerResourceRecordPtr -Name "0.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0" -
ZoneName "8.b.d.0.1.0.0.2.ip6.arpa" -PtrDomainName "web01.example.test." -TimeToLive
01:00:00
```

For clean **Forward-Confirmed Reverse DNS (FCrDNS)**, the same rule applies to both IPv4 and IPv6: a reverse query on the IP returns a `PTR` to `web01.example.test`, and a forward query on `web01.example.test` returns that exact IP address.

> ⚠ **Common reverse zone pitfalls:** If `-CreatePtr` silently fails or `Resolve-DnsName` returns `NXDOMAIN` for a local IP, the reverse zone prefix almost certainly does not match the subnet (such as a `/16` zone `0.192.in-addr.arpa` instead of `/24` `2.0.192.in-addr.arpa`, or an incorrect host label). Also note that Windows Server DNS does not automatically build RFC 2317 classless reverse delegations for subnets smaller than `/24` via `-NetworkID`.

Configuring Forwarders and Recursion

With `example.test` and `2.0.192.in-addr.arpa`, `WIN-CORE01` answers all internal forward and reverse queries **authoritatively**. When a client—or the server itself—queries an external domain such as `learn.microsoft.com`, no local zone holds that record. To resolve external names cleanly without timing out, you configure **forwarders** and **recursion** explicitly via PowerShell.

How the DNS Server Chooses Between Zone, Forwarder, and Root Hints

Before changing parameters, review the exact decision sequence Windows DNS follows for every incoming query:

1. **Client query arrives on port 53**Request reaches WIN-CORE01 locally or over the network.
2. **Local DNS cache**If a valid, unexpired answer exists in cache, the server responds immediately.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

3. **Authoritative zone check** If the name belongs to example.test or 2.0.192.in-addr.arpa, the server answers authoritatively.

4. **Recursion check** If recursion is disabled, the server refuses to resolve external names and forwards nothing.

5. **DNS forwarder** With recursion enabled, the server queries configured upstream resolvers in order.

6. **Root hints fallback** If all forwarders fail, iterative root resolution runs only when UseRootHint is True.

Forwarding vs. Direct Recursion

In daily operations, "recursion" and "forwarders" are frequently conflated, but they control two distinct mechanisms:

- **Recursion (`Set-DnsServerRecursion`):** The master switch determining whether `WIN-CORE01` may resolve external domains on behalf of clients at all. With recursion disabled (`-Enable $false`), the server operates purely authoritatively for its own zones and ignores configured forwarders.
- **DNS forwarders (`Set-DnsServerForwarder`):** Delegates recursive resolution for unknown domains to one or more upstream resolvers (such as your lab router, firewall, or ISP resolver). The upstream resolver walks the DNS hierarchy and returns the answer to `WIN-CORE01`.
- **Root hints:** A built-in list of global root name servers (`a.root-servers.net` through `m.root-servers.net`). If no forwarders are configured—or if all forwarders fail while `-UseRootHint` is `$true`—Windows DNS queries iteratively starting from the root (`.`).

Forwarder (`UseRootHint = False`)

Outbound port 53
: Only to the defined upstream resolver

Response latency
: Often benefits from warm upstream cache

Firewall control
: Strictly scoped to `192.0.2.1`

Failure behavior
: Clean `SERVFAIL` when forwarder is down

Direct Root Hints (`UseRootHint = True`)

Outbound port 53
: To arbitrary authoritative servers worldwide

Response latency
: Requires multiple iterative round-trips

Firewall control
: Requires broad outbound port-53 access

Failure behavior
: Attempts own resolution; times out in isolated labs

Shared: Both require recursion (`Enable : True`) to be enabled on `WIN-CORE01`

> **Upstream forwarder choice depends on your network:** There is no single universally "best" public resolver. In lab and corporate networks, forwarders normally point to the upstream router, perimeter firewall, or internal DNS filter so existing security, logging, and split-DNS policies remain intact. Direct external resolvers only apply if outbound port-53 traffic to the internet is permitted and intended.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Checking Initial Recursion and Forwarder State

Inspect the current recursion and forwarder settings on `WIN-CORE01` first:

```
# Initial state: check recursion settings
Get-DnsServerRecursion

# Initial state: check configured forwarders and timeout
Get-DnsServerForwarder

# Inspect built-in root hints (output truncated)
Get-DnsServerRootHint | Select-Object -First 5
```

On a fresh Windows Server DNS installation, recursion is enabled (`Enable : True`), the forwarder list (`IPAddress`) is empty, and `UseRootHint` defaults to `True`. Without adjustment, the server immediately tries to query internet root servers directly—causing long timeouts in isolated lab segments without direct outbound port-53 internet access.

Configuring Forwarders and Fallback Behavior

In our `192.0.2.0/24` example network, the upstream router/resolver at `192.0.2.1` handles external resolution (replace this IP with your actual gateway or upstream resolver). Use Set-DnsServerForwarder to set the forwarder, timeout, and root-hints fallback behavior:

```
# Set upstream forwarder, limit timeout to 3 seconds, and disable root-hints fallback
Set-DnsServerForwarder -IPAddress "192.0.2.1" -Timeout 3 -UseRootHint $false -PassThru
```

Schematic example output (not executed live on the target system):

```
UseRootHint        : False
Timeout(s)         : 3
EnableReordering   : True
IPAddress          : {192.0.2.1}
```

- **`-IPAddress "192.0.2.1"`:** Replaces the forwarder list with the specified upstream resolver. Use `Add-DnsServerForwarder` to append without overwriting, and `Remove-DnsServerForwarder` to remove one entry.
- **`-Timeout 3`:** Waits at most 3 seconds for a response from this forwarder before trying the next entry. A tight timeout prevents clients from hanging when the first forwarder is unreachable.
- **`-UseRootHint $false`:** Disables automatic fallback to root hints. If all forwarders are unreachable, `WIN-CORE01` fails fast with `ServerFailure` instead of attempting direct root queries.

Confirm that recursion is active (`Enable : True`) so the forwarder path is used:

```
# Verify recursion is enabled (enable if False)
Get-DnsServerRecursion
# If Enable is False:
# Set-DnsServerRecursion -Enable $true -PassThru
```

Schematic example output (not executed live on the target system):

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
Enable                : True
AdditionalTimeout(s)  : 4
RetryInterval(s)      : 1
Timeout(s)            : 8
SecureResponse        : True
```

Conditional Forwarders for Isolated Partner or Lab Zones

Sometimes only a specific partner or lab domain (such as `lab.example.test`) needs to be routed to another internal DNS server (`192.0.2.50`) while all other external queries continue using `192.0.2.1`. Instead of changing the global forwarder, add a conditional forwarder zone with `Add-DnsServerConditionalForwarderZone`:

```
# Forward queries for lab.example.test specifically to 192.0.2.50
Add-DnsServerConditionalForwarderZone -Name "lab.example.test" -MasterServers 192.0.2.50 -
ForwarderTimeout 3 -PassThru
```

Multiple Forwarders and Built-In Server Verification (`Test-DnsServer`)

If two upstream resolvers are available on your network (`192.0.2.1` and `192.0.2.2`), configure both as an ordered list and verify with `Test-DnsServer` that `WIN-CORE01` resolves through them cleanly:

```
# Configure two redundant forwarders with a 3-second timeout
Set-DnsServerForwarder -IPAddress "192.0.2.1", "192.0.2.2" -Timeout 3 -UseRootHint $false
-PassThru
```

```
# Run built-in DNS server self-test including forwarder context
Test-DnsServer -IPAddress "192.0.2.10"
Test-DnsServer -IPAddress "192.0.2.10" -Context Forwarder
```

Schematic example output of `Test-DnsServer -Context Forwarder` (not executed live on the target system):

```
IPAddress   Result     Message
---------   ------     -------
192.0.2.10  Success    DNS server 192.0.2.10 responded cleanly using configured
forwarders.
```

If an error appears instead of `Success`, `Message` points directly to unreachable forwarders or disabled recursion. To remove a secondary forwarder later, run `Remove-DnsServerForwarder -IPAddress "192.0.2.2" -Force`.

Note `EnableReordering` (`True` by default): when multiple forwarders are configured and the first becomes slow or unresponsive, Windows DNS dynamically reorders the list by response time. For a single-forwarder lab setup, keep only the reachable `192.0.2.1` address.

Behavior When Forwarders Are Unreachable

When `192.0.2.1` fails, Windows DNS processes the list strictly in sequence:

1. It sends the query to the first forwarder and waits up to `-Timeout` (3 seconds).
2. If no response arrives, it sends the query to the next configured forwarder.
3. If no forwarder responds, `-UseRootHint` decides the outcome: `$true` triggers iterative resolution via root hints; `$false` aborts immediately and returns `SERVFAIL` (`ServerFailure`) to the client.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Security Risks of Open Recursion

Whenever recursion is enabled (`Enable : True`), `WIN-CORE01` resolves arbitrary external domains for any client allowed through the firewall. That creates two classic risks if access is not restricted:

- **DNS amplification / reflection DDoS:** A recursive DNS server exposed on UDP 53 to the public internet acts as an **open resolver**, allowing attackers to spoof source IPs and amplify small queries into large response floods against victims.
- **Resource exhaustion and cache poisoning:** Unbounded external queries fill the DNS cache, consume CPU and bandwidth, and widen the surface for cache poisoning.

Always restrict inbound port-53 rules in Windows Defender Firewall and on perimeter firewalls strictly to trusted internal client subnets—never `Any` from the internet. On a purely authoritative public server, keep recursion permanently disabled (`Set-DnsServerRecursion -Enable $false`).

Switching the Server's Own DNS Client and Testing Forwarding

Now that both local zones and the upstream forwarder are in place, revisit the warning from Section 3: you can safely point `WIN-CORE01`'s own DNS client to `192.0.2.10` without breaking external name resolution:

```
# Set the server's own DNS client to 192.0.2.10 (adjust InterfaceAlias if needed)
Set-DnsClientServerAddress -InterfaceAlias "Ethernet" -ServerAddresses ("192.0.2.10")
```

Verify both resolution paths separately with `Resolve-DnsName`:

```
# 1. Inspect configured forwarders
Get-DnsServerForwarder
```

```
# 2. Test authoritative local zone (no forwarding)
Resolve-DnsName -Name "web01.example.test" -Server 192.0.2.10 -DnsOnly -NoHostsFile

# 3. Test external name resolution via forwarder (if upstream is reachable in lab)
Resolve-DnsName -Name "learn.microsoft.com" -Server 192.0.2.10 -DnsOnly -NoHostsFile
```

Schematic example output for the external forwarder test (not executed live on the target system):

```
Name                                   Type   TTL   Section    IPAddress
----                                   ----   ---   -------    ---------
learn.microsoft.com                    CNAME  300   Answer
learn.microsoft.com.edgekey.net
learn.microsoft.com.edgekey.net        CNAME  60    Answer     e1234.dscx.akamaiedge.net
e1234.dscx.akamaiedge.net              A      20    Answer     203.0.113.50
```

In a healthy setup, the local zone immediately returns `192.0.2.25`, while the external query resolves through `192.0.2.1` and enters `WIN-CORE01`'s local cache.

> ⚠ **Common forwarder pitfalls:** If external queries time out despite a valid forwarder IP, check whether an outbound firewall blocks port `53` (`UDP/TCP`) from `WIN-CORE01` to the upstream resolver, whether the upstream resolver refuses recursive queries from `192.0.2.10` (`REFUSED`), or whether recursion was disabled on the server (`Set-DnsServerRecursion -Enable $false`).

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

DNS Queries and Troubleshooting

Once local zones, `PTR` records, and forwarders are configured, real-world troubleshooting begins: a client reports a hostname is unreachable, or receives an outdated IP after a record update. Instead of blindly disabling the firewall or restarting the `DNS` service, start from the observed symptom and narrow down the exact layer systematically.

Clean Test Queries with `Resolve-DnsName` and `nslookup`

The biggest trap in Windows DNS troubleshooting is hidden fallbacks: when a regular lookup fails, Windows checks the local `hosts` file or falls back to LLMNR and NetBIOS. To inspect strictly what `WIN-CORE01` (`192.0.2.10`) returns over DNS, enforce two switches on Resolve-DnsName:

```
# Pure DNS query bypassing hosts file and LLMNR/NetBIOS directly against WIN-CORE01
Resolve-DnsName -Name "web01.example.test" -Server 192.0.2.10 -DnsOnly -NoHostsFile

# Targeted checks by record type (Forward, Reverse, SOA, and external via forwarder)
Resolve-DnsName -Name "web01.example.test" -Type AAAA -Server 192.0.2.10 -DnsOnly -
NoHostsFile
Resolve-DnsName -Name "192.0.2.25" -Type PTR -Server 192.0.2.10 -DnsOnly -NoHostsFile
Resolve-DnsName -Name "example.test" -Type SOA -Server 192.0.2.10 -DnsOnly -NoHostsFile
Resolve-DnsName -Name "learn.microsoft.com" -Type A -Server 192.0.2.10 -DnsOnly -
NoHostsFile
```

- **`-Server 192.0.2.10`:** Ignores the client NIC's configured DNS settings and queries your target server directly.
- **`-DnsOnly`:** Disables LLMNR and NetBIOS fallbacks.
- **`-NoHostsFile`:** Excludes entries from `%SystemRoot%\System32\drivers\etc\hosts`.
- **`-TcpOnly` (optional for size issues):** Forces TCP port 53 to test whether a firewall permits UDP 53 but drops TCP 53.

For a quick cross-check—including from a Linux admin workstation—use `nslookup` (or `dig @192.0.2.10 web01.example.test` on Linux):

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# Direct query against 192.0.2.10 with nslookup (forward and reverse)
nslookup web01.example.test 192.0.2.10
nslookup 192.0.2.25 192.0.2.10

# Query a nonexistent name (expected: immediate NXDOMAIN)
Resolve-DnsName -Name "does-not-exist.example.test" -Server 192.0.2.10 -DnsOnly -
NoHostsFile
```

Schematic example output for `NXDOMAIN` (not executed live on the target system):

```
Resolve-DnsName : does-not-exist.example.test : DNS name does not exist
```

For a nonexistent hostname inside your authoritative zone, the server must respond immediately with `DNS name does not exist` (`NXDOMAIN`)—**not** with a timeout. An instant `NXDOMAIN` proves that network routing, port 53, and the `DNS` service itself are all working cleanly and only the requested record is absent.

Interpreting Common DNS Response Codes (`RCODE`)

Before diving into the six failure modes, keep these four primary DNS status codes in mind:

Response Code	Meaning	Typical Real-World Cause
`NOERROR`	Query succeeded	Either records are returned—or the name exists (e.g., `web01` has an `A` record) while the requested type (`AAAA`) does not (`NODATA`).
`NXDOMAIN`	Name does not exist	The queried hostname does not exist in the authoritative zone (or a subdomain label is wrong).

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Response Code	Meaning	Typical Real-World Cause
`SERVFAIL`	Server failure	The DNS server could not complete the query—typically an unreachable forwarder, a paused zone (`IsPaused : True`), or a corrupt zone file.
`REFUSED`	Query refused	The target resolver rejected the query (for example, recursion denied for external client IPs or upstream ACL policy).

Narrowing Down Six Failure Modes from Observed Symptoms

Depending on what `Resolve-DnsName` or `nslookup` reports, you can isolate the root cause to one of six layers:

Symptom 1: Network or Routing Issue

- **Observation:** `Resolve-DnsName` fails with `This operation returned because the timeout period expired` ; queries for `example.test` or `192.0.2.10` never reach the server.
- **Check:** Verify basic IP reachability and the TCP handshake on port 53 from the client:

```
# Basic reachability and TCP port 53 check from the client
Test-NetConnection -ComputerName 192.0.2.10 -Port 53
```

Schematic example output when port 53 is reachable (not executed live on the target system):

```
ComputerName     : 192.0.2.10
RemoteAddress    : 192.0.2.10
RemotePort       : 53
InterfaceAlias   : Ethernet
SourceAddress    : 192.0.2.25
TcpTestSucceeded : True
```

If `PingSucceeded` fails (assuming ICMP is permitted) and `TcpTestSucceeded` is `False`, the issue lies in routing, VLAN configuration, or the static IP on `WIN-CORE01`.

Symptom 2: Unreachable DNS Service or Blocking Firewall

- **Observation:** `192.0.2.10` is reachable on the network, yet `Resolve-DnsName -Server 192.0.2.10` times out from the client and `TcpTestSucceeded` is `False` (while `127.0.0.1` locally on `WIN-CORE01` may work or also fail).
- **Check on `WIN-CORE01`:**

```
# 1. Is the service running?
Get-Service -Name DNS | Select-Object Name, Status, StartType

# 2. Is DNS listening on 192.0.2.10 (not just 127.0.0.1)?
Get-NetUDPEndpoint -LocalPort 53 | Format-Table LocalAddress, LocalPort, OwningProcess
Get-NetTCPConnection -LocalPort 53 -State Listen | Format-Table LocalAddress, LocalPort,
State

# 3. Which network profile is active and do port-53 rules allow it?
Get-NetConnectionProfile | Select-Object InterfaceAlias, NetworkCategory
Get-NetFirewallPortFilter | Where-Object { $_.LocalPort -eq 53 } | Get-NetFirewallRule |
    Select-Object DisplayName, Enabled, Profile, Action
```

If the service is stopped, inspect the event log for why it stopped before restarting it. If the service is listening on `0.0.0.0` or `192.0.2.10` and responds on `127.0.0.1`, the firewall is blocking external traffic—most commonly because the NIC sits in `Public` while the rule only permits `Private` / `Domain`. Fix the interface profile (`Set-NetConnectionProfile`) or rule scope rather than disabling the firewall.

Symptom 3: Missing, Misspelled, or Paused Zone

- **Observation:** Querying `web01.example.test` returns an external `NXDOMAIN`, `SERVFAIL`, or an unexpected external IP.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

- **Cause:** Either the zone does not exist under the exact spelling (`exampel.test`), or the zone is paused/shut down. Not recognizing itself as authoritative, the server forwards the query upstream.
- **Check on `WIN-CORE01`:**

```
# Check existence, type, and pause state of all zones
Get-DnsServerZone | Format-Table ZoneName, ZoneType, IsPaused, IsShutdown, ZoneFile
```

Schematic example output for an accidentally paused zone (not executed live on the target system):

```
ZoneName              ZoneType IsPaused IsShutdown ZoneFile
--------              -------- -------- ---------- --------
example.test          Primary      True      False example.test.dns
2.0.192.in-addr.arpa  Primary     False      False 2.0.192.in-addr.arpa.dns
```

Ensure `IsPaused : False` and `IsShutdown : False` and verify exact `ZoneName` spelling. Resume a paused zone with `Resume-DnsServerZone -Name "example.test"` .

Symptom 4: Incorrect or Missing Resource Record

- **Observation:** `Resolve-DnsName -Name "example.test" -Type SOA -Server 192.0.2.10` answers authoritatively, yet querying `web01.example.test` returns `NXDOMAIN` (or `NOERROR` with an empty `Answer` section for `-Type AAAA`), or a `CNAME` points to an unresolvable target.
- **Check on `WIN-CORE01`:**

```
# Inspect full zone inventory
Get-DnsServerResourceRecord -ZoneName "example.test" | Format-Table HostName, RecordType,
TimeToLive, RecordData
```

Schematic example output showing a double-qualified name typo or wrong alias target (not executed live on the target system):

```
HostName           RecordType TimeToLive RecordData
--------           ---------- ---------- ----------
portal             CNAME      01:00:00   web02.example.test.
web01.example.test A          00:05:00   192.0.2.25
```

Common causes include a missing `AAAA` record for IPv6-preferring clients, a typo in a `CNAME`'s `-HostNameAlias` (`web02` instead of `web01`), a stale `PTR` after an IP change, or accidentally passing `web01.example.test` as `-Name` (which creates `web01.example.test.example.test`). Spot double-qualified names quickly with:

```
# Check for accidentally double-qualified hostnames inside the zone
Get-DnsServerResourceRecord -ZoneName "example.test" |
    Where-Object { $_.HostName -like "*example.test*" } |
    Format-Table HostName, RecordType, RecordData
```

Symptom 5: Recursion or Forwarder Issue

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

- **Observation:** Internal lookups for `web01.example.test` and `192.0.2.25` succeed immediately, but all external lookups (`learn.microsoft.com`) time out or return `ServerFailure` (`SERVFAIL`).
- **Check on `WIN-CORE01`:**

```
# 1. Check recursion and forwarder settings on the server
Get-DnsServerRecursion
Get-DnsServerForwarder

# 2. Test directly from WIN-CORE01 whether forwarder 192.0.2.1 responds
Resolve-DnsName -Name "learn.microsoft.com" -Server 192.0.2.1 -DnsOnly -NoHostsFile
```

If `192.0.2.1` does not respond directly from `WIN-CORE01`, the problem is the upstream resolver or outbound port 53. If `192.0.2.1` responds directly while `192.0.2.10` fails, check whether `Enable : False` is set in `Get-DnsServerRecursion` or the forwarder timeout is too short.

Symptom 6: DNS Cache Effects

- **Observation:** You updated an `A` record from `192.0.2.20` to `192.0.2.25` (`Get-DnsServerResourceRecord` shows the new IP), yet the client—or `WIN-CORE01` itself for forwarded external records—still returns the old IP or a cached `NXDOMAIN`.
- **Cause:** Both the Windows client resolver service (`Dnscache`) and the `DNS` server service cache answers for the duration of the record TTL (or `SOA` `MinimumTimeToLive` for negative `NXDOMAIN` responses).
- **Check and targeted flush:**

```
# Check and flush local client cache
Get-DnsClientCache | Where-Object { $_.Entry -like "*example.test*" }
Clear-DnsClientCache
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# Check and flush DNS server cache on WIN-CORE01
Show-DnsServerCache | Where-Object { $_.HostName -like "*example.test*" }
Clear-DnsServerCache -Force
```

Schematic example output of `Get-DnsClientCache` prior to flushing (not executed live on the target system):

```
Entry              RecordName          RecordType Status    TimeToLive Data
-----              ----------          ---------- ------    ---------- ----
web01.example.test  web01.example.test  A          Success          248 192.0.2.20
```

While `TimeToLive` counts down, the Windows client serves `192.0.2.20` locally without even contacting `WIN-CORE01`. After flushing both caches, the next query immediately returns the updated zone data.

Inspecting Windows Event Logs

When the cause remains unclear—such as a service failing to start, a zone file syntax error, or unexpected configuration changes—two event channels provide clear diagnostics via `Get-WinEvent`:

- **Classic service log (`DNS Server`):** Records service starts, zone file load errors, permission issues, and socket binding failures.
- **Audit channel (`Microsoft-Windows-DNSServer/Audit`):** Enabled by default, logs changes to zones, forwarders, and resource records (such as Event ID `515` when a resource record is created, or Event ID `541` when a server/zone setting is modified).

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# Inspect the last 20 events from the operational DNS Server log
Get-WinEvent -LogName "DNS Server" -MaxEvents 20 |
    Select-Object TimeCreated, Id, LevelDisplayName, Message |
    Format-List

# Inspect configuration and record changes in the Audit log over the last 24 hours
Get-WinEvent -FilterHashtable @{
    LogName   = 'Microsoft-Windows-DNSServer/Audit'
    StartTime = (Get-Date).AddHours(-24)
} -MaxEvents 20 | Select-Object TimeCreated, Id, Message
```

Schematic example output from `Microsoft-Windows-DNSServer/Audit` (not executed live on the target system):

```
TimeCreated          Id Message
-----------          -- -------
12.11.2026 08:15:10 515 A resource record of type 1, name web01, TTL 3600 and RDATA
192.0.2.25 was created in scope Default of zone example.test.
12.11.2026 08:17:42 541 The setting SecureSecondaries on scope example.test has been set
to NoTransfer.
```

If you need temporary packet-level query/answer logging, check `Get-DnsServerDiagnostics` first, enable debug logging briefly via `Set-DnsServerDiagnostics`, and turn it right back off afterward to avoid unnecessary disk I/O:

```
# Check current diagnostic logging settings
Get-DnsServerDiagnostics
```

```
# Temporarily log queries and answers during active troubleshooting
New-Item -Path "C:\Logs" -ItemType Directory -Force | Out-Null
Set-DnsServerDiagnostics -Queries $true -Answers $true -EnableLoggingToFile $true -
LogFilePath "C:\Logs\dns-debug.log"

# Disable immediately once analysis is complete
Set-DnsServerDiagnostics -Queries $false -Answers $false -EnableLoggingToFile $false
```

> **No blind restarts during troubleshooting:** Restarting the `DNS` service flushes the server cache and reloads zone files, which neither fixes firewall blocks nor corrects broken records or forwarders—and can mask the underlying root cause. Always run `Resolve-DnsName`, socket/firewall checks, and `Get-WinEvent` before touching the service.

Complete 6-Point Verification Routine After Changes

After any major change—and before pointing clients to `192.0.2.10`—run a standardized check across all six operational layers:

```
# 1. Service state and autostart
Get-Service -Name DNS | Select-Object Name, Status, StartType

# 2. Port 53 listeners (UDP and TCP)
Get-NetUDPEndpoint -LocalPort 53 | Select-Object LocalAddress, LocalPort
Get-NetTCPConnection -LocalPort 53 -State Listen | Select-Object LocalAddress, LocalPort,
State

# 3. Loaded zones without pause or shutdown
Get-DnsServerZone | Where-Object { -not $_.IsAutoCreated } |
    Format-Table ZoneName, ZoneType, IsPaused, IsShutdown, DynamicUpdate,
SecureSecondaries
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# 4. Forward and reverse resolution against 192.0.2.10
Resolve-DnsName -Name "web01.example.test" -Type A -Server 192.0.2.10 -DnsOnly -
NoHostsFile
Resolve-DnsName -Name "192.0.2.25" -Type PTR -Server 192.0.2.10 -DnsOnly -NoHostsFile

# 5. Expected NXDOMAIN for unknown internal name
try {
    Resolve-DnsName -Name "nxdomain-check.example.test" -Server 192.0.2.10 -DnsOnly -
NoHostsFile -ErrorAction Stop
} catch {
    "Expected NXDOMAIN received: $($_.Exception.Message)"
}

# 6. External resolution via forwarder
Resolve-DnsName -Name "learn.microsoft.com" -Type A -Server 192.0.2.10 -DnsOnly -
NoHostsFile
```

EDNS0, UDP Fragmentation, and Split-DNS Pitfalls

Two subtle issues often confuse troubleshooting even when port 53 is generally reachable:

- **Large responses and TCP fallback:** Classic DNS over UDP was capped at 512 bytes; EDNS0 permits larger UDP payloads (typically 1,232 or 4,096 bytes). When a response exceeds the UDP limit or fragmented UDP packets are dropped by intermediate firewalls, clients fall back to `53/TCP`. If `Resolve-DnsName ... -DnsOnly` works while `Resolve-DnsName ... -TcpOnly` fails, a firewall is specifically blocking `53/TCP`.
- **Split-DNS collisions:** If you use the exact same domain name internally as on the public internet, internal clients only see records hosted on `WIN-CORE01`—any unmirrored public subdomain returns `NXDOMAIN` internally. Using `example.test` cleanly avoids that maintenance trap in a workgroup lab.

Operations, Security, and Maintenance

A DNS server is not production-ready simply because an initial `Resolve-DnsName` query succeeded. Between a working lab setup and a resilient infrastructure service lie explicit access boundaries, locked-down zone transfers, a tested backup and restore procedure, and a

repeatable verification routine after updates and reboots.

Permissions and Administrative Access

On the standalone workgroup server `WIN-CORE01`, there is no Active Directory for Group Policy or delegated AD permissions. DNS administration rests with local `Administrators` and the local `DnsAdmins` security group created during role installation:

- **Least privilege in daily operations:** Use dedicated administrative accounts over SSH rather than shared default logins. While `DnsAdmins` scopes zone-level management, changes to services, the Windows registry, and firewall rules remain the responsibility of local administrators in a workgroup.
- **Bind listening IP addresses explicitly:** On multi-homed hosts (such as separate client and management VLANs), do not let the DNS service listen indiscriminately on every interface. Restrict the binding to `192.0.2.10` and loopback `127.0.0.1`:

```
# Inspect current listening IP addresses
(Get-DnsServerSetting -All).ListeningIpAddress

# Bind DNS explicitly to 192.0.2.10 and loopback 127.0.0.1
$dnsSettings = Get-DnsServerSetting -All
$dnsSettings.ListeningIpAddress = @("192.0.2.10", "127.0.0.1")
Set-DnsServerSetting -InputObject $dnsSettings -PassThru |
    Select-Object -ExpandProperty ListeningIpAddress
```

Include `127.0.0.1` in the list if local scripts on `WIN-CORE01` still query loopback—otherwise the service stops accepting queries on `127.0.0.1`.

- **Scope firewall rules to client subnets:** In Section 4 we verified that port 53 is open. For production operation, confirm via `Get-NetFirewallAddressFilter` that inbound port-53 rules limit `-RemoteAddress` to your actual client and management subnets (such as `192.0.2.0/24`) rather than allowing arbitrary source addresses:

```
# Inspect existing port-53 firewall rules including source address filters
Get-NetFirewallPortFilter |
    Where-Object { $_.LocalPort -eq 53 } |
    Get-NetFirewallRule |
    ForEach-Object {
        $addr = $_ | Get-NetFirewallAddressFilter
        [PSCustomObject]@{
            DisplayName   = $_.DisplayName
            Enabled       = $_.Enabled
            Profile       = $_.Profile
            RemoteAddress = $addr.RemoteAddress
        }
    } | Format-Table -AutoSize
```

> ⚠ **SSH and DNS protection go hand in hand:** As covered in the Server Core baseline guide, port `22/TCP` is the administrative entry point to `WIN-CORE01`. Restrict SSH access to your management network and port `53/UDP` and `53/TCP` strictly to the subnets permitted to query this resolver.

Locking Down Zone Transfers and Dynamic Updates

Never leave zone transfers (`AXFR` or `IXFR`) open to arbitrary hosts—an unrestricted transfer allows any requester to enumerate every internal hostname, service, and IP layout in your zone.

Use <code class="badge-code">Set-DnsServerPrimaryZone to define exactly which peers may request transfers for each primary zone:

- `NoTransfer` : Disables zone transfers completely—the right default while `WIN-CORE01` operates standalone without a secondary DNS server.
- `TransferToSecureServers` : Permits transfers exclusively to IP addresses listed in `-SecondaryServers` (use this when adding a secondary DNS server for redundancy).
- `TransferToZoneNameServer` : Limits transfers to hosts listed in the zone's `NS` records.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

- `TransferAnyServer`: Allows transfers to any host and should be avoided in production.

```
# Disable zone transfers for both zones on WIN-CORE01
Set-DnsServerPrimaryZone -Name "example.test" -SecureSecondaries "NoTransfer" -PassThru |
    Select-Object ZoneName, SecureSecondaries, DynamicUpdate

Set-DnsServerPrimaryZone -Name "2.0.192.in-addr.arpa" -SecureSecondaries "NoTransfer" -
PassThru |
    Select-Object ZoneName, SecureSecondaries, DynamicUpdate
```

Schematic example output (not executed live on the target system):

```
ZoneName              SecureSecondaries DynamicUpdate
--------              ----------------- -------------
example.test          NoTransfer        None
2.0.192.in-addr.arpa  NoTransfer        None
```

Confirm `DynamicUpdate` remains `None` at the same time: in a workgroup without Active Directory, `Secure` is unavailable, and `NonsecureAndSecure` would allow any client on the network to overwrite records unauthenticated.

Separating Authoritative and Recursive Functions

As covered in Section 8, a server with recursion enabled must never be reachable on port 53 from the internet or untrusted networks. That yields two clean operating models:

- **Internal combined resolver (`WIN-CORE01`):** Answers internal zones authoritatively and resolves external names via `192.0.2.1`, strictly protected by the subnet firewall rules shown above.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

- **Authoritative-only DNS server (no client forwarding):** If a server only publishes its own zones to other resolvers, disable recursion explicitly (`Set-DnsServerRecursion -Enable $false`).

Backing Up and Restoring File-Backed DNS Zones

The architecture from Section 2 directly dictates how backups work: on AD-integrated servers, zones live in Active Directory and are backed up via System State or `Export-DnsServerZone` . On our standalone workgroup server (`IsDsIntegrated : False`), resource records **already reside as text files** under `%SystemRoot%\System32\dns\` , while zone metadata (`ZoneType` , `DynamicUpdate` , `SecureSecondaries`) and server parameters (forwarders, recursion, listening IPs) are stored in the Windows registry.

A complete, consistent backup therefore requires **both** layers:

1. All `.dns` zone files in `$env:SystemRoot\System32\dns\` (`example.test.dns` , `2.0.192.in-addr.arpa.dns` , and `cache.dns`).
2. Both DNS registry paths: `HKLM\SYSTEM\CurrentControlSet\Services\DNS\Parameters` and `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\DNS Server\Zones` .

```
# Reproducible DNS backup (zone files + registry + readable configuration)
$backupRoot = "C:\Backup\DNS\$(Get-Date -Format 'yyyy-MM-dd_HHmm')"
New-Item -Path $backupRoot -ItemType Directory -Force | Out-Null

# 1. Back up zone files
Copy-Item -Path "$env:SystemRoot\System32\dns\*.dns" -Destination $backupRoot -Force

# 2. Export DNS registry configuration
reg.exe export "HKLM\SYSTEM\CurrentControlSet\Services\DNS\Parameters" "$backupRoot\dns-
parameters.reg" /y
reg.exe export "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\DNS Server\Zones"
"$backupRoot\dns-zones.reg" /y

# 3. Export readable server configuration as a reference snapshot
Get-DnsServer | Export-Clixml -Path "$backupRoot\dns-server-config.clixml"
```

```
Get-ChildItem -Path $backupRoot | Select-Object Name, Length, LastWriteTime
```

Schematic example output of a complete backup set (not executed live on the target system):

```
Name                     Length LastWriteTime
----                     ------ -------------
2.0.192.in-addr.arpa.dns    612 12.11.2026 08:20:01
cache.dns                  3328 12.11.2026 08:20:01
dns-parameters.reg         4180 12.11.2026 08:20:01
dns-server-config.clixml  18420 12.11.2026 08:20:01
dns-zones.reg              2940 12.11.2026 08:20:01
example.test.dns            845 12.11.2026 08:20:01
```

> ⚠ **Never overwrite zone files while `DNS` is running:** Because `dns.exe` buffers zone data in RAM and flushes it later or on shutdown, always stop the `DNS` service first, restore files and registry keys, and then start the service again.

Follow this exact restore sequence:

```
# Example restore sequence (adjust path to the actual backup folder)
$restoreFrom = "C:\Backup\DNS\2026-11-12_0800"

# 1. Stop the DNS service cleanly
Stop-Service -Name DNS

# 2. Restore zone files and registry keys
Copy-Item -Path "$restoreFrom\*.dns" -Destination "$env:SystemRoot\System32\dns\" -Force
```

```
reg.exe import "$restoreFrom\dns-parameters.reg"
reg.exe import "$restoreFrom\dns-zones.reg"

# 3. Start service and verify immediately
Start-Service -Name DNS
Get-Service -Name DNS | Select-Object Name, Status, StartType
Get-DnsServerZone | Format-Table ZoneName, ZoneType, IsPaused, IsShutdown
Resolve-DnsName -Name "web01.example.test" -Server 192.0.2.10 -DnsOnly -NoHostsFile
```

A backup is only trustworthy once you have tested this restore sequence in a lab or maintenance window.

Common Backup and Restore Pitfalls

- **Restoring `.dns` files without the registry:** On startup, `dns.exe` does not blindly load every `.dns` file in `System32\dns`—it only loads zones registered under `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\DNS Server\Zones`. Without the registry export, zones will not appear in `Get-DnsServerZone` on a rebuilt server.
- **Importing the registry with missing `.dns` files:** The zone appears in `Get-DnsServerZone` but fails to load (`IsShutdown : True` or an error in the `DNS Server` event log) because the backing file is missing or lacks read permissions.
- **Replacing files while the service is running:** When `dns.exe` later flushes memory to disk, it overwrites your restored files. Always run `Stop-Service -Name DNS` before restoring.

Automated Daily Backups via Task Scheduler

To ensure backups run reliably without manual intervention, save a backup script at `C:\Scripts\Backup-DnsServer.ps1` and schedule it daily at 02:00 as `SYSTEM` via PowerShell:

```
# Create backup script (including 30-day retention cleanup)
New-Item -Path "C:\Scripts" -ItemType Directory -Force | Out-Null
@'
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
$backupRoot = "C:\Backup\DNS\$(Get-Date -Format 'yyyy-MM-dd_HHmm')"
New-Item -Path $backupRoot -ItemType Directory -Force | Out-Null
Copy-Item -Path "$env:SystemRoot\System32\dns\*.dns" -Destination $backupRoot -Force
reg.exe export "HKLM\SYSTEM\CurrentControlSet\Services\DNS\Parameters" "$backupRoot\dns-
parameters.reg" /y | Out-Null
reg.exe export "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\DNS Server\Zones"
"$backupRoot\dns-zones.reg" /y | Out-Null
Get-DnsServer | Export-Clixml -Path "$backupRoot\dns-server-config.clixml"
Get-ChildItem -Path "C:\Backup\DNS" -Directory |
    Where-Object { $_.CreationTime -lt (Get-Date).AddDays(-30) } |
    Remove-Item -Recurse -Force
'@ | Set-Content -Path "C:\Scripts\Backup-DnsServer.ps1" -Encoding UTF8

# Register daily scheduled task at 02:00 as SYSTEM
$action    = New-ScheduledTaskAction -Execute "powershell.exe" -Argument "-NoProfile -
ExecutionPolicy Bypass -File C:\Scripts\Backup-DnsServer.ps1"
$trigger   = New-ScheduledTaskTrigger -Daily -At "02:00"
$principal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType ServiceAccount -
RunLevel Highest
Register-ScheduledTask -TaskName "Backup-WinCore01-DNS" -Action $action -Trigger $trigger
-Principal $principal -Force
```

Cache Maintenance, Event Logs, and Routine Monitoring

In steady-state operation, Windows DNS requires little manual tuning, but benefits from periodic checks:

- **Inspect cache settings:** Use `Get-DnsServerCache` to check `MaxTTL`, `MaxNegativeTtl`, `MaxKBSize`, and `EnablePollutionProtection`. Only run `Clear-DnsServerCache -Force` intentionally after a specific change—never on an hourly cron/task schedule.

```
# Inspect current cache parameters
Get-DnsServerCache | Format-List MaxTTL, MaxNegativeTtl, MaxKBSize,
EnablePollutionProtection
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Schematic example output (not executed live on the target system):

```
MaxTTL                    : 1.00:00:00
MaxNegativeTtl            : 00:15:00
MaxKBSize                 : 0
EnablePollutionProtection : True
```

To prevent stale `NXDOMAIN` responses from upstream forwarders from lingering too long in cache, adjust `MaxNegativeTtl` (and optionally cap cache memory) via `Set-DnsServerCache`:

```
# Cap negative caching at 5 minutes and optionally limit cache size (e.g., 51200 KB = 50
MB)
Set-DnsServerCache -MaxNegativeTtl 00:05:00 -MaxKBSize 51200 -PassThru
```

- **Static records and aging/scavenging:** All records added manually via PowerShell carry `Timestamp : 0` (static) and are never purged by DNS aging. In a workgroup without dynamic client registration (`DynamicUpdate : None`), keep automatic scavenging disabled and confirm zone aging is off:

```
# Confirm server-wide scavenging and per-zone aging are disabled
Get-DnsServerScavenging | Format-List ScavengingState, ScavengingInterval
Get-DnsServerZoneAging -Name "example.test" | Format-List ZoneName, AgingEnabled,
RefreshInterval, NoRefreshInterval
```

Schematic example output (not executed live on the target system):

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
ScavengingState    : False
ScavengingInterval : 00:00:00

ZoneName          : example.test
AgingEnabled      : False
RefreshInterval   : 7.00:00:00
NoRefreshInterval : 7.00:00:00
```

Leaving both `ScavengingState` and `AgingEnabled` set to `False` keeps the standalone server behavior simple and predictable.

- **Review event logs:** Periodically inspect `Get-WinEvent -LogName "DNS Server"` for errors and `Microsoft-Windows-DNSServer/Audit` for unexpected changes, as shown in Section 9.

Behavior During Windows Updates and Reboots

Cumulative security updates on Windows Server 2025 Core require periodic reboots. Because `192.0.2.10` cannot answer DNS queries while rebooting, patch during announced maintenance windows—and plan a second DNS server in production environments where downtime is unacceptable.

After every patch reboot, verify three items immediately over SSH:

1. Is the `DNS` service set to `StartType : Automatic` and `Status : Running`?
2. Are all zones loaded without pause (`Get-DnsServerZone` showing `IsPaused : False` and `IsShutdown : False`)?
3. Do both internal records (`web01.example.test`, `192.0.2.25`) and external lookups via forwarder (`learn.microsoft.com`) resolve?

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

```
# Before patching: trigger an immediate DNS backup
Start-ScheduledTask -TaskName "Backup-WinCore01-DNS"

# Compact post-reboot verification after Windows Update
(Get-CimInstance -ClassName Win32_OperatingSystem).LastBootUpTime
Get-Service -Name DNS | Select-Object Name, Status, StartType
Get-DnsServerZone | Where-Object { -not $_.IsAutoCreated } | Format-Table ZoneName,
IsPaused, IsShutdown
Get-WinEvent -LogName "DNS Server" -MaxEvents 5 | Select-Object TimeCreated, Id,
LevelDisplayName, Message
Resolve-DnsName -Name "web01.example.test" -Server 192.0.2.10 -DnsOnly -NoHostsFile
Resolve-DnsName -Name "192.0.2.25" -Type PTR -Server 192.0.2.10 -DnsOnly -NoHostsFile
```

Production Readiness Criteria

Before assigning `WIN-CORE01` as the live DNS server for network clients, confirm every condition below is met:

- **Networking and addressing:** Static IPv4 (`192.0.2.10/24`) and verified server-side DNS client configuration without self-resolution failure.
- **Service and bindings:** `DNS` service set to `Automatic` , listening on `53/UDP` and `53/TCP` on `192.0.2.10` (plus `127.0.0.1`).
- **Access control:** Port 53 (and SSH port 22) firewall rules restricted to known internal client and management subnets; no open resolver exposure.
- **Zone security:** `DynamicUpdate` set to `None` , `SecureSecondaries` set to `NoTransfer` (or explicitly scoped to named secondary servers).
- **Forwarders and resolution:** Forwarders tested; forward (`A` , `AAAA` , `CNAME`), reverse (`PTR`), and `NXDOMAIN` responses verified via `Resolve-DnsName` .
- **Backup and redundancy:** Current `.dns` + registry backup exists and has been restore-tested; a second DNS server is planned for environments requiring zero-downtime patching.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Controlled Client Cutover

Once all readiness criteria are met, roll out the new DNS server in three controlled stages rather than switching the entire subnet at once:

1. **Test on a single client first:** Point one administrative test client to `192.0.2.10` (`Set-DnsClientServerAddress`) and verify internal (`web01.example.test`, `192.0.2.25`) and external (`learn.microsoft.com`) resolution.
2. **Update DHCP Option 006 gradually:** Only after the test client proves stable, update DHCP Option `006` (*DNS Servers*) to `192.0.2.10` (or `192.0.2.10` and `192.0.2.11` when redundant).
3. **Update statically configured servers:** Migrate static hosts one by one and confirm via `ipconfig /all` or `Get-DnsClientServerAddress` that no stale unreachable DNS server remains as primary.

```
# On a Windows test client: switch DNS server, flush cache, and test live resolution
Set-DnsClientServerAddress -InterfaceAlias "Ethernet" -ServerAddresses ("192.0.2.10")
Clear-DnsClientCache

# Check ALL interfaces (including IPv6!) to see which DNS servers are active
Get-DnsClientServerAddress | Where-Object { $_.ServerAddresses.Count -gt 0 } |
    Format-Table InterfaceAlias, AddressFamily, ServerAddresses

# Query without -Server to test the client's actual OS configuration
Resolve-DnsName -Name "web01.example.test" -DnsOnly -NoHostsFile
Resolve-DnsName -Name "learn.microsoft.com" -DnsOnly -NoHostsFile
```

> □□ **Watch out for IPv6 DNS servers on dual-stack clients:** If a Windows client receives an external IPv6 DNS address via Router Advertisement (RDNSS) or DHCPv6 (such as the router's IPv6 address), Windows may prefer that IPv6 resolver and return an external `NXDOMAIN` for `web01.example.test` even though `192.0.2.10` is configured on IPv4.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

> Always check `Get-DnsClientServerAddress` for **both IPv4 and IPv6**.

Comparing Runtime Configuration Against a Baseline

Especially in multi-admin environments, a compact baseline check immediately reveals whether zones were added, `SecureSecondaries` was relaxed, or forwarders were altered:

```
# Output core runtime configuration as a compact verification report
[PSCustomObject]@{
    ServerName    = $env:COMPUTERNAME
    ServiceStatus = (Get-Service -Name DNS).Status
    Forwarders    = ((Get-DnsServerForwarder).IPAddress -join ', ')
    UseRootHint   = (Get-DnsServerForwarder).UseRootHint
    Recursion     = (Get-DnsServerRecursion).Enable
    Zones         = ((Get-DnsServerZone | Where-Object { -not $_.IsAutoCreated } |
        ForEach-Object { "$($_.ZoneName) ($($_.ZoneType)/$($_.SecureSecondaries))" }) -
join '; ')
} | Format-List
```

Schematic example output (not executed live on the target system):

```
ServerName    : WIN-CORE01
ServiceStatus : Running
Forwarders    : 192.0.2.1, 192.0.2.2
UseRootHint   : False
Recursion     : True
Zones         : 2.0.192.in-addr.arpa (Primary/NoTransfer); example.test
(Primary/NoTransfer)
```

For automated monitoring agents or scheduled health checks, wrap the core verification in a script returning exit code `0` (healthy) or `1` (failure):

```
# Automated DNS health check returning exit code 0 (OK) or 1 (error)
try {
    $svc = Get-Service -Name DNS -ErrorAction Stop
    if ($svc.Status -ne 'Running') { throw "DNS service is not running." }
    $null = Resolve-DnsName -Name "web01.example.test" -Type A -Server 192.0.2.10 -DnsOnly
-NoHostsFile -ErrorAction Stop
    $null = Resolve-DnsName -Name "192.0.2.25" -Type PTR -Server 192.0.2.10 -DnsOnly -
NoHostsFile -ErrorAction Stop
    Write-Output "OK: DNS service and zone resolution on WIN-CORE01 are healthy."
    exit 0
} catch {
    Write-Error "ERROR: $($_.Exception.Message)"
    exit 1
}
```

Command Reference (Cheatsheet)

The reference below groups all PowerShell commands used throughout this guide by task. Every command targets the `WIN-CORE01` workgroup baseline (`192.0.2.10` , zone `example.test`) and runs directly over SSH in an elevated PowerShell session.

DNS Server Role

Check role installation status Shows whether the DNS Server role and RSAT tools are installed on Windows Server 2025 Core

$ `Get-WindowsFeature -Name DNS, RSAT-DNS-Server`

Install DNS role with management tools Installs the DNS Server role and the DnsServer PowerShell module without AD dependencies

```
$ Install-WindowsFeature -Name DNS -IncludeManagementTools
```

Verify DnsServer module availability Checks whether the PowerShell module for DNS management is available

```
$ Get-Module -ListAvailable -Name DnsServer
```

Service Management

Check service status and startup type Verifies the DNS service is running and set to start automatically

```
$ Get-Service -Name DNS | Select-Object Name, Status, StartType
```

Cleanly restart DNS service Restarts the DNS service (also flushes the server DNS cache)

```
$ Restart-Service -Name DNS
```

Check open port 53 sockets Lists all active UDP and TCP listeners on port 53

```
$ Get-NetUDPEndpoint -LocalPort 53; Get-NetTCPConnection -LocalPort 53 -State Listen
```

Language-independent port 53 firewall check Displays all firewall rules targeting port 53 regardless of OS display language

```
$ Get-NetFirewallPortFilter | Where-Object { $_.LocalPort -eq 53 } | Get-NetFirewallRule | Select-Object DisplayName, Enabled, Profile, Action
```

Scope firewall rule to client subnet Allows inbound UDP port 53 strictly from 192.0.2.0/24 on the Private profile

```
$ New-NetFirewallRule -DisplayName "DNS-Server (UDP-In, Lab-Subnet)" -Direction Inbound -Protocol UDP -LocalPort 53 -RemoteAddress "192.0.2.0/24" -Profile Private -Action Allow
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

DNS Zones

List all DNS zones Shows zone name, type, pause state, and backing zone file

```
$ Get-DnsServerZone | Format-Table ZoneName, ZoneType, IsPaused, IsShutdown, ZoneFile
```

Create file-backed forward zone Creates the primary forward lookup zone example.test with dynamic updates disabled

```
$ Add-DnsServerPrimaryZone -Name "example.test" -ZoneFile "example.test.dns" -DynamicUpdate "None" -PassThru
```

Create file-backed reverse zone Creates the primary IPv4 reverse lookup zone for 192.0.2.0/24

```
$ Add-DnsServerPrimaryZone -NetworkID "192.0.2.0/24" -ZoneFile "2.0.192.in-addr.arpa.dns" -DynamicUpdate "None" -PassThru
```

Create secondary zone on a second host Creates a read-only copy of example.test pulling from primary 192.0.2.10

```
$ Add-DnsServerSecondaryZone -Name "example.test" -ZoneFile "example.test.dns" -MasterServers "192.0.2.10" -PassThru
```

Flush in-memory zone buffer to disk Forces immediate writeback of the RAM zone buffer into the .dns file

```
$ dnscmd.exe /ZoneWriteBack example.test
```

Disable zone transfers Blocks AXFR/IXFR transfers for a primary zone in standalone operation

```
$ Set-DnsServerPrimaryZone -Name "example.test" -SecureSecondaries "NoTransfer" -PassThru
```

Resource Records

List all records in a zone Displays all resource records including type, TTL, and record data

```
$ Get-DnsServerResourceRecord -ZoneName "example.test" | Format-Table HostName, RecordType, TimeToLive, RecordData
```

Create A record Creates an IPv4 host record for web01.example.test

```
$ Add-DnsServerResourceRecordA -ZoneName "example.test" -Name "web01" -IPv4Address "192.0.2.25" -TimeToLive 00:05:00 -PassThru
```

Create AAAA record Creates an IPv6 host record for web01.example.test

```
$ Add-DnsServerResourceRecordAAAA -ZoneName "example.test" -Name "web01" -IPv6Address "2001:db8::20" -TimeToLive 00:05:00 -PassThru
```

Create CNAME alias Points portal.example.test to the canonical hostname web01.example.test

```
$ Add-DnsServerResourceRecordCName -ZoneName "example.test" -Name "portal" -HostNameAlias "web01.example.test" -TimeToLive 00:05:00 -PassThru
```

Create PTR record in reverse zone Maps IP 192.0.2.25 to the canonical hostname web01.example.test.

```
$ Add-DnsServerResourceRecordPtr -ZoneName "2.0.192.in-addr.arpa" -Name "25" -PtrDomainName "web01.example.test." -TimeToLive 00:05:00 -PassThru
```

Create TXT record Stores a text record for internal documentation or verification

```
$ Add-DnsServerResourceRecord -ZoneName "example.test" -Name "_meta" -Txt -DescriptiveText "admindocs-lab-zone" -TimeToLive 01:00:00
```

Create SRV record Maps a service name and port to a target host

```
$ Add-DnsServerResourceRecord -ZoneName "example.test" -Name "_https._tcp" -Srv -DomainName "web01.example.test." -Port 443 -Priority 10 -Weight 100 -TimeToLive 01:00:00
```

Safely delete a resource record (dry-run) Simulates removing an A record before executing the change

```
$ Remove-DnsServerResourceRecord -ZoneName "example.test" -RRType "A" -Name "web01" -RecordData "192.0.2.25" -WhatIf
```

Forwarders and Recursion

Inspect forwarders and recursion Displays configured upstream resolvers and recursion state

```
$ Get-DnsServerForwarder; Get-DnsServerRecursion
```

Configure forwarder without root hints Forwards external queries to 192.0.2.1 and disables root-hints fallback

```
$ Set-DnsServerForwarder -IPAddress "192.0.2.1" -Timeout 3 -UseRootHint $false -PassThru
```

Create conditional forwarder zone Delegates queries for lab.example.test specifically to another internal DNS server

```
$ Add-DnsServerConditionalForwarderZone -Name "lab.example.test" -MasterServers 192.0.2.50 -ForwarderTimeout 3 -PassThru
```

Disable recursion explicitly Turns off recursive forwarding on authoritative-only DNS servers

```
$ Set-DnsServerRecursion -Enable $false -PassThru
```

DNS Queries

Authoritative forward lookup bypassing cache/hosts Queries web01.example.test directly over the network against 192.0.2.10

```
$ Resolve-DnsName -Name "web01.example.test" -Type A -Server 192.0.2.10 -DnsOnly -NoHostsFile
```

Test IPv4 reverse lookup Verifies the PTR record for 192.0.2.25 directly against WIN-CORE01

```
$ Resolve-DnsName -Name "192.0.2.25" -Type PTR -Server 192.0.2.10 -DnsOnly -NoHostsFile
```

Force TCP fallback test Verifies that port 53/TCP is reachable for large DNS responses

```
$ Resolve-DnsName -Name "web01.example.test" -Server 192.0.2.10 -DnsOnly -TcpOnly
```

Classic nslookup query Queries 192.0.2.10 on systems without the DnsClient PowerShell module

```
$ nslookup web01.example.test 192.0.2.10
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-dns-server-powershell-setup-and-management

Troubleshooting, Logs, and Maintenance

Network and port 53 check from client Tests IP reachability and TCP handshake to port 53 on WIN-CORE01

```
$ Test-NetConnection -ComputerName 192.0.2.10 -Port 53
```

Read DNS Server operational event log Shows the latest 20 events from the DNS Server service log

```
$ Get-WinEvent -LogName "DNS Server" -MaxEvents 20 | Select-Object TimeCreated, Id, LevelDisplayName, Message | Format-List
```

Inspect DNS Audit log (last 24 hours) Lists zone, forwarder, and record changes from the Audit channel

```
$ Get-WinEvent -FilterHashtable @{ LogName = 'Microsoft-Windows-DNSServer/Audit'; StartTime = (Get-Date).AddHours(-24) } -MaxEvents 20 | Select-Object TimeCreated, Id, Message
```

Check and flush DNS server cache Displays current cache parameters and flushes the server cache if needed

```
$ Get-DnsServerCache; Clear-DnsServerCache -Force
```

Run built-in DNS server self-test Verifies DNS server responsiveness including configured forwarders

```
$ Test-DnsServer -IPAddress "192.0.2.10" -Context Forwarder
```

Enable temporary packet debug logging Writes DNS queries and answers to a log file during active troubleshooting

```
$ Set-DnsServerDiagnostics -Queries $true -Answers $true -EnableLoggingToFile $true -LogFilePath "C:\Logs\dns-debug.log"
```

Export DNS zone registry configuration Backs up zone metadata from the Windows registry to a .reg file

```
$ reg.exe export "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\DNS Server\Zones" "C:\Backup\dns-zones.reg" /y
```

Further Resources

The following primary sources cover the DNS concepts, PowerShell modules, and Server Core baseline used in this guide:

Windows Server 2025 Core: Installation and Administration via SSH and PowerShell Foundational guide covering SConfig, static networking, OpenSSH, and the WIN-CORE01 baseline

DNS Overview in Windows Server (Microsoft Learn) Official architecture and feature overview for the Windows Server DNS role

Manage DNS Zones (Microsoft Learn) Documentation on primary, secondary, and stub zones and zone storage types

Install and Configure DNS Server (Microsoft Learn) Microsoft quickstart for installing the DNS Server role and baseline settings

DnsServer PowerShell Module (Microsoft Learn) Complete cmdlet reference for zones, resource records, forwarders, cache, and diagnostics

Set-DnsServerPrimaryZone (Microsoft Learn) Parameter reference for securing zone transfers (-SecureSecondaries) and dynamic updates

Resolve-DnsName (Microsoft Learn) Reference for targeted DNS queries in PowerShell including -DnsOnly, -NoHostsFile, and -TcpOnly