// Tutorial //

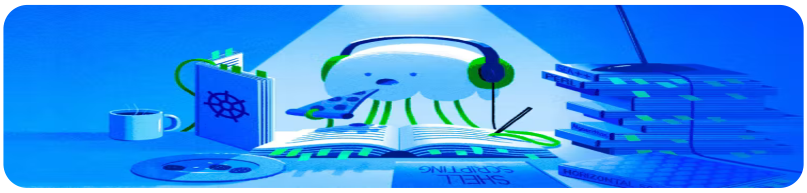
How To Configure BIND as a Private Network DNS Server on Ubuntu 18.04

Published on July 7, 2018 · Updated on March 23, 2022

DNS Networking Ubuntu 18.04

By Justin Ellingwood and Mitchell Anicas

English



Not using Ubuntu 18.04?
Choose a different version or distribution. Ubuntu 18.04

Introduction

An important part of managing server configuration and infrastructure includes maintaining an easy way to look up network interfaces and IP addresses by name, by setting up a proper Domain Name System (DNS). Using fully qualified domain names (FQDNs), instead of IP addresses, to specify network addresses eases the configuration of services and applications, and increases the maintainability of configuration files. Setting up your own DNS for your private network is a great way to improve the management of your servers.

In this tutorial, we will go over how to set up an internal DNS server, using the BIND name server software (BIND9) on Ubuntu 18.04, that can be used by your servers to resolve private hostnames and private IP addresses. This provides a central way to manage your internal hostnames and private IP addresses, which is indispensable when your environment expands to more than a few hosts.

Prerequisites

To complete this tutorial, you will need the following infrastructure. Create each server **in the same datacenter** with **private networking enabled**:

- A fresh Ubuntu 18.04 server to serve as the Primary DNS server, **ns1**
- (Recommended) A second Ubuntu 18.04 server to serve as a Secondary DNS server, **ns2**
- Additional servers in the same datacenter that will be using your DNS servers

On each of these servers, configure administrative access via a `sudo` user and a firewall by following our Ubuntu 18.04 initial server setup guide.

If you are unfamiliar with DNS concepts, it is recommended that you read at least the first three parts of our Introduction to Managing DNS.

Example Infrastructure and Goals

For the purposes of this article, we will assume the following:

- We have two servers which will be designated as our DNS name servers. We will refer to these as **ns1** and **ns2** in this guide.
- We have two additional client servers that will be using the DNS infrastructure we create. We will call these **host1** and **host2** in this guide. You can add as many as you'd like for your infrastructure.
- All of these servers exist in the same datacenter. We will assume that this is the **nyc3** datacenter.
- All of these servers have private networking enabled and are on the `10.128.0.0/16` subnet (**you will likely have to adjust this for your servers**).
- All servers are connected to a project that runs on `example.com`. Since our DNS system will be entirely internal and private, you do not have to purchase a domain name. However, using a domain you own may help avoid conflicts with publicly routable domains.

With these assumptions in mind, we will use a naming scheme based around the subdomain `nyc3.example.com` to refer to the example private subnet or zone referenced throughout this guide. Therefore, **host1**'s private Fully-Qualified Domain Name (FQDN) will be `host1.nyc3.example.com`. Refer to the following table the relevant details:

Host	Role	Private FQDN	Private IP Address
ns1	Primary DNS Server	`ns1.nyc3.example.com`	`10.128.10.11`
ns2	Secondary DNS Server	`ns2.nyc3.example.com`	`10.128.20.12`
host1	Generic Host 1	`host1.nyc3.example.com`	`10.128.100.101`
host2	Generic Host 2	`host2.nyc3.example.com`	`10.128.200.102`

Instant DNS Lookup Tool
Tool

POPULAR TOPICS
Ubuntu
Linux Basics
JavaScript
React
Python
Security
Apache
MySQL
Databases
Docker
Kubernetes
Ebooks
Browse all topic tags

ALL TUTORIALS →

QUESTIONS
Q&A
Ask a question
DigitalOcean Product Docs
DigitalOcean Support

EVENTS
Tech Talks
Hacktoberfest
Deploy

GET INVOLVED
Community Newsletter
Hollie's Hub for Good
Write for DOnations
Community tools and integrations
Hatch Startup program

CREATE YOUR FREE COMMUNITY ACCOUNT! →

Note: Your existing setup will be different, but the example names and IP addresses will be used to demonstrate how to configure a DNS server to provide a functioning internal DNS. You should be able to easily adapt this setup to your own environment by replacing the host names and private IP addresses with your own. It is not necessary to use the region name of the datacenter in your naming scheme, but we use it here to denote that these hosts belong to a particular datacenter's private network. If you utilize multiple datacenters, you can set up an internal DNS within each respective datacenter.

RELATED

DNS Tips and Tricks
Tutorial

How To Launch Your Site on a New Ubuntu 12.04 Server with LAMP, SFTP, and DNS
Tutorial

By the end of this tutorial, we will have a primary DNS server, **ns1**, and optionally a secondary DNS server, **ns2**, which will serve as a backup.

Let's get started by installing BIND our both our primary and secondary DNS servers, **ns1** and **ns2**.

Installing BIND on DNS Servers

Note: Text that is highlighted like this is important! It will often be used to denote something that needs to be replaced with your own settings or that it should be modified or added to a configuration file. For example, if you see something like `host1.nyc3.example.com`, replace it with the FQDN of your own server. Likewise, if you see `host1_private_IP`, replace it with the private IP address of your own server.

On both DNS servers, **ns1** and **ns2**, update the `apt` package cache by typing:

```
ns$ sudo apt-get update
```

Copy

Now install BIND:

```
ns$ sudo apt-get install bind9 bind9utils bind9-doc
```

Copy

Setting Bind to IPv4 Mode

Before continuing, let's set BIND to IPv4 mode since our private networking uses IPv4 exclusively. On both servers, edit the `bind9` default settings file using your preferred text editor. The following example uses `nano`:

```
ns$ sudo nano /etc/default/bind9
```

Copy

Add `-4` to the end of the `OPTIONS` parameter. It should look like the following:

/etc/default/bind9

```
. . .
OPTIONS="-u bind -4"
```

Save and close the file when you are finished. If you used `nano` to edit the file, you can do so by pressing `CTRL + X`, `Y`, then `ENTER`.

Restart BIND to implement the changes:

```
ns$ sudo systemctl restart bind9
```

Copy

Now that BIND is installed, let's configure the primary DNS server.

Configuring the Primary DNS Server

BIND's configuration consists of multiple files, which are included from the main configuration file, `named.conf`. These filenames begin with `named` because that is the name of the process that BIND runs (with `named` being short for "**name d**aemon", as in "domain name daemon"). We will start with configuring the `named.conf.options` file.

Configuring the Options File

On **ns1**, open the `named.conf.options` file for editing:

```
ns1$ sudo nano /etc/bind/named.conf.options
```

Copy

Above the existing `options` block, create a new ACL (access control list) block called `trusted`. This is where we will define a list of clients from which we will allow recursive DNS queries (i.e. your servers that are in the same datacenter as **ns1**). Using our example private IP addresses, we will add **ns1**, **ns2**, **host1**, and **host2** to our list of trusted clients:

/etc/bind/named.conf.options — 1 of 3

```
acl "trusted" {
        10.128.10.11;    # ns1 - can be set to localhost
        10.128.20.12;    # ns2
        10.128.100.101;  # host1
        10.128.200.102;  # host2
};

options {

        . . .
```

Now that we have our list of trusted DNS clients, we will want to edit the `options` block. Currently, the start of the block looks like the following:

/etc/bind/named.conf.options — 2 of 3

```
        . . .
};

options {
        directory "/var/cache/bind";
        . . .
}
```

Below the `directory` directive, add the highlighted configuration lines (and substitute in the appropriate **ns1** private IP address) so it looks something like this:

/etc/bind/named.conf.options — 3 of 3

```
        . . .

};

options {
        directory "/var/cache/bind";

        recursion yes;                 # enables resursive queries
        allow-recursion { trusted; };  # allows recursive queries from "trusted" clients
        listen-on { 10.128.10.11; };   # ns1 private IP address - listen on private network o
        allow-transfer { none; };      # disable zone transfers by default

        forwarders {
                8.8.8.8;
                8.8.4.4;
        };

        . . .
};
```

When you are finished, save and close the `named.conf.options` file. The above configuration specifies that only your own servers (the `trusted` ones) will be able to query your DNS server for outside domains.

Next, we will specify our DNS zones by configuring the `named.conf.local` file.

Configuring the Local File

On **ns1**, open the `named.conf.local` file for editing:

```
ns1$ sudo nano /etc/bind/named.conf.local
```

Copy

Aside from a few comments, the file should be empty. Here, we will specify our forward and reverse zones. *DNS zones* designate a specific scope for managing and defining DNS records. Since our example domains will all be within the `nyc3.example.com` subdomain, we will use that as our forward zone. Because our servers' private IP addresses are each in the `10.128.0.0/16` IP space, we will set up a reverse zone so that we can define reverse lookups within that range.

Add the forward zone with the following lines, substituting the zone name with your own and the **secondary DNS server's private IP address** in the `allow-transfer` directive:

/etc/bind/named.conf.local — 1 of 2

```
zone "nyc3.example.com" {
    type master;
    file "/etc/bind/zones/db.nyc3.example.com"; # zone file path
    allow-transfer { 10.128.20.12; };           # ns2 private IP address - secondary
};
```

Assuming that our private subnet is `10.128.0.0/16`, add the reverse zone by with the following lines (**note that our reverse zone name starts with `128.10` which is the octet reversal of `10.128`**):

/etc/bind/named.conf.local — 2 of 2

```
    . . .
};

zone "128.10.in-addr.arpa" {
    type master;
    file "/etc/bind/zones/db.10.128";  # 10.128.0.0/16 subnet
    allow-transfer { 10.128.20.12; };  # ns2 private IP address - secondary
};
```

If your servers span multiple private subnets but are in the same datacenter, be sure to specify an additional zone and zone file for each distinct subnet. When you are finished adding all of your desired zones, save and close the `named.conf.local` file.

Now that our zones are specified in BIND, we need to create the corresponding forward and reverse zone files.

Creating the Forward Zone File

The forward zone file is where we define DNS records for forward DNS lookups. That is, when the DNS receives a name query, `host1.nyc3.example.com` for example, it will look in the forward zone file to resolve **host1**'s corresponding private IP address.

Let's create the directory where our zone files will reside. According to our `named.conf.local` configuration, that location should be `/etc/bind/zones`:

```
ns1$ sudo mkdir /etc/bind/zones
```

Copy

We will base our forward zone file on the sample `db.local` zone file. Copy it to the proper location with the following commands:

```
ns1$ sudo cp /etc/bind/db.local /etc/bind/zones/db.nyc3.example.com
```

Copy

Now let's edit our forward zone file:

```
ns1$ sudo nano /etc/bind/zones/db.nyc3.example.com
```

Copy

Initially, it will look something like the following:

/etc/bind/zones/db.nyc3.example.com — original

```
$TTL    604800
@       IN      SOA     localhost. root.localhost. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      localhost.      ; delete this line
@       IN      A       127.0.0.1       ; delete this line
@       IN      AAAA    ::1             ; delete this line
```

First, you will want to edit the SOA record. Replace the first `localhost` with **ns1**'s FQDN, then replace `root.localhost` with `admin.nyc3.example.com`. Every time you edit a zone file, you need to increment the `Serial` value before you restart the `named` process. We will increment it to `3`. It should now look something like this:

/etc/bind/zones/db.nyc3.example.com — updated 1 of 3

```
@       IN      SOA     ns1.nyc3.example.com. admin.nyc3.example.com. (
                              3         ; Serial

                              . . .
```

Next, delete the three records at the end of the file (after the SOA record). If you're not sure which lines to delete, they are marked with comments reading `delete this line` above.

At the end of the file, add your name server records with the following lines (replace the names with your own). Note that the second column specifies that these are `NS` records:

/etc/bind/zones/db.nyc3.example.com — updated 2 of 3

```
. . .

; name servers - NS records
    IN      NS      ns1.nyc3.example.com.
    IN      NS      ns2.nyc3.example.com.
```

Now, add the A records for your hosts that belong in this zone. This includes any server whose name we want to end with `.nyc3.example.com` (substitute the names and private IP addresses). Using our example names and private IP addresses, we will add A records for **ns1**, **ns2**, **host1**, and **host2** like so:

/etc/bind/zones/db.nyc3.example.com — updated 3 of 3

```
. . .

; name servers - A records
ns1.nyc3.example.com.          IN      A       10.128.10.11
ns2.nyc3.example.com.          IN      A       10.128.20.12

; 10.128.0.0/16 - A records
host1.nyc3.example.com.        IN      A      10.128.100.101
host2.nyc3.example.com.        IN      A      10.128.200.102
```

Save and close the `db.nyc3.example.com` file.

Our final example forward zone file looks like the following:

/etc/bind/zones/db.nyc3.example.com — updated

```
$TTL    604800
@       IN      SOA     ns1.nyc3.example.com. admin.nyc3.example.com. (
                  3     ; Serial
             604800     ; Refresh
              86400     ; Retry
            2419200     ; Expire
             604800 )   ; Negative Cache TTL
;
; name servers - NS records
     IN      NS      ns1.nyc3.example.com.
     IN      NS      ns2.nyc3.example.com.

; name servers - A records
ns1.nyc3.example.com.          IN      A       10.128.10.11
ns2.nyc3.example.com.          IN      A       10.128.20.12

; 10.128.0.0/16 - A records
host1.nyc3.example.com.        IN      A      10.128.100.101
host2.nyc3.example.com.        IN      A      10.128.200.102
```

Now let's move onto the reverse zone file(s).

Creating the Reverse Zone File(s)

Reverse zone files are where we define DNS PTR records for reverse DNS lookups. That is, when the DNS receives a query by IP address, `10.128.100.101` for example, it will look in the reverse zone file(s) to resolve the corresponding FQDN, `host1.nyc3.example.com` in this case.

On **ns1**, for each reverse zone specified in the `named.conf.local` file, create a reverse zone file. We will base our reverse zone file(s) on the sample `db.127` zone file. Copy it to the proper location with the following commands (substituting the destination filename so it matches your reverse zone definition):

```
ns1$ sudo cp /etc/bind/db.127 /etc/bind/zones/db.10.128
```

Edit the reverse zone file that corresponds to the reverse zone(s) defined in `named.conf.local`:

```
ns1$ sudo nano /etc/bind/zones/db.10.128
```

Initially, it will look something like the following:

/etc/bind/zones/db.10.128 — original

```
$TTL    604800
@       IN      SOA     localhost. root.localhost. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      localhost.      ; delete this line
1.0.0   IN      PTR     localhost.      ; delete this line
```

In the same manner as the forward zone file, you will want to edit the SOA record and increment the **serial** value. It should look something like this:

/etc/bind/zones/db.10.128 — updated 1 of 3

```
@       IN      SOA     ns1.nyc3.example.com. admin.nyc3.example.com. (
                              3         ; Serial

                              . . .
```

Now delete the two records at the end of the file (after the SOA record). If you're not sure which lines to delete, they are marked with a `delete this line` comment above.

At the end of the file, add your name server records with the following lines (replace the names with your own). Note that the second column specifies that these are `NS` records:

/etc/bind/zones/db.10.128 — updated 2 of 3

```
. . .

; name servers - NS records
      IN      NS      ns1.nyc3.example.com.
      IN      NS      ns2.nyc3.example.com.
```

Then add `PTR` records for all of your servers whose IP addresses are on the subnet of the zone file that you are editing. In our example, this includes all of our hosts because they are all on the `10.128.0.0/16` subnet. Note that the first column consists of the last two octets of your servers' private IP addresses in **reversed order**. Be sure to substitute names and private IP addresses to match your servers:

/etc/bind/zones/db.10.128 — updated 3 of 3

```
. . .

; PTR Records
11.10   IN      PTR     ns1.nyc3.example.com.    ; 10.128.10.11
12.20   IN      PTR     ns2.nyc3.example.com.    ; 10.128.20.12
101.100 IN      PTR     host1.nyc3.example.com.  ; 10.128.100.101
102.200 IN      PTR     host2.nyc3.example.com.  ; 10.128.200.102
```

Save and close the reverse zone file (repeat this section if you need to add more reverse zone files).

Our final example reverse zone file looks like the following:

/etc/bind/zones/db.10.128 — updated

```
$TTL    604800
@       IN      SOA     nyc3.example.com. admin.nyc3.example.com. (
                              3         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
; name servers
      IN      NS      ns1.nyc3.example.com.
      IN      NS      ns2.nyc3.example.com.

; PTR Records
11.10   IN      PTR     ns1.nyc3.example.com.    ; 10.128.10.11
12.20   IN      PTR     ns2.nyc3.example.com.    ; 10.128.20.12
101.100 IN      PTR     host1.nyc3.example.com.  ; 10.128.100.101
102.200 IN      PTR     host2.nyc3.example.com.  ; 10.128.200.102
```

We're done editing our files, so next we can check our files for errors.

Checking the BIND Configuration Syntax

Run the following command to check the syntax of the `named.conf*` files:

```
ns1$ sudo named-checkconf
```

If your named configuration files have no syntax errors, you will return to your shell prompt and see no error messages. If there are problems with your configuration files, review the error message and the `Configure Primary DNS Server` section, then try `named-checkconf` again.

The `named-checkzone` command can be used to check the correctness of your zone files. Its first argument specifies a zone name, and the second argument specifies the corresponding zone file, which are both defined in `named.conf.local`.

For example, to check the `nyc3.example.com` forward zone configuration, run the following command (change the names to match your forward zone and file):

```
$ sudo named-checkzone nyc3.example.com /etc/bind/zones/db.nyc3.example.com
```

And to check the `128.10.in-addr.arpa` reverse zone configuration, run the following command (change the numbers to match your reverse zone and file):

```
$ sudo named-checkzone 128.10.in-addr.arpa /etc/bind/zones/db.10.128
```

When all of your configuration and zone files have no errors in them, you should be ready to restart the BIND service.

Restarting BIND

Restart BIND:

```
ns1$ sudo systemctl restart bind9
```

If you have the UFW firewall configured, open up access to BIND by typing:

```
ns1$ sudo ufw allow Bind9
```

Your primary DNS server is now set up and ready to respond to DNS queries. Let's move on to creating the secondary DNS server.

Configuring the Secondary DNS Server

In most environments, it is a good idea to set up a secondary DNS server that will respond to requests if the primary becomes unavailable. Luckily, configuring the secondary DNS server is much less complicated.

On **ns2**, edit the `named.conf.options` file:

```
ns2$ sudo nano /etc/bind/named.conf.options
```

At the top of the file, add the ACL with the private IP addresses of all of your trusted servers:

/etc/bind/named.conf.options — updated 1 of 2 (secondary)

```
acl "trusted" {
        10.128.10.11;   # ns1
        10.128.20.12;   # ns2 - can be set to localhost
        10.128.100.101;  # host1
        10.128.200.102;  # host2
};

options {

        . . .
```

Below the `directory` directive, add the following lines:

/etc/bind/named.conf.options — updated 2 of 2 (secondary)

```
        recursion yes;
        allow-recursion { trusted; };
        listen-on { 10.128.20.12; };      # ns2 private IP address
        allow-transfer { none; };          # disable zone transfers by default

        forwarders {
                8.8.8.8;
                8.8.4.4;
        };
```

Save and close the `named.conf.options` file. This file should look exactly like **ns1**'s `named.conf.options` file except it should be configured to listen on **ns2**'s private IP address.

Now edit the `named.conf.local` file:

```
ns2$ sudo nano /etc/bind/named.conf.local
```

Define slave zones that correspond to the master zones on the primary DNS server. Note that the type is `slave`, the file does not contain a path, and there is a `masters` directive which should be set to the primary DNS server's private IP address. If you defined multiple reverse zones in the primary DNS server, make sure to add them all here:

/etc/bind/named.conf.local — updated (secondary)

```
zone "nyc3.example.com" {
    type slave;
    file "db.nyc3.example.com";
    masters { 10.128.10.11; };  # ns1 private IP
};

zone "128.10.in-addr.arpa" {
    type slave;
    file "db.10.128";
    masters { 10.128.10.11; };  # ns1 private IP
};
```

Now save and close the `named.conf.local` file.

Run the following command to check the validity of your configuration files:

```
ns2$ sudo named-checkconf
```

If this command doesn't return any errors, restart BIND:

```
ns2$ sudo systemctl restart bind9
```

Then allow DNS connections to the server by altering the UFW firewall rules:

```
ns2$ sudo ufw allow Bind9
```

With that, you now have primary and secondary DNS servers for private network name and IP address resolution. Now you must configure your client servers to use your private DNS servers.

Configuring DNS Clients

Before all of your servers in the `trusted` ACL can query your DNS servers, you must configure each of them to use **ns1** and **ns2** as name servers. This process varies depending on OS, but for most Linux distributions it involves adding your name servers to the `/etc/resolv.conf` file.

Ubuntu 18.04 Clients

On Ubuntu 18.04, networking is configured with Netplan, an abstraction that allows you to write standardized network configuration and apply it to incompatible backend networking software. To configure DNS, we need to write a Netplan configuration file.

First, find the device associated with your private network by querying the private subnet with the `ip address` command:

```
$ ip address show to 10.128.0.0/16
```

```
Output
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qle
    inet 10.128.100.101/16 brd 10.128.255.255 scope global eth1
           valid_lft forever preferred_lft forever
```

In this example, the private interface is `eth1`.

Next, create a new file in `/etc/netplan` called `00-private-nameservers.yaml`:

```
$ sudo nano /etc/netplan/00-private-nameservers.yaml
```

Inside, paste the following contents. You will need to modify the interface of the private network, the addresses of your **ns1** and **ns2** DNS servers, and the DNS zone:

Note: Netplan uses the YAML data serialization format for its configuration files. Because YAML uses indentation and whitespace to define its data structure, make sure that your definition uses consistent indentation to avoid errors.

/etc/netplan 00-private-nameservers.yaml

```
network:
    version: 2
    ethernets:
        eth1:                                    # Private network interface
            nameservers:
                addresses:
                - 10.128.10.11                # Private IP for ns1
                - 10.132.20.12                # Private IP for ns2
                search: [ nyc3.example.com ]    # DNS zone
```

Save and close the file when you are finished.

Next, tell Netplan to attempt to use the new configuration file by using `netplan try`. If there are problems that cause a loss of networking, Netplan will automatically roll back the changes after a timeout:

```
$ sudo netplan try
```

```
Output
Warning: Stopping systemd-networkd.service, but it can still be activated by:
  systemd-networkd.socket
Do you want to keep these settings?


Press ENTER before the timeout to accept the new configuration


Changes will revert in 120 seconds
```

If the countdown is updating correctly at the bottom, the new configuration is at least functional enough to not break your SSH connection. Press **ENTER** to accept the new configuration.

Now, check that the system's DNS resolver to determine if your DNS configuration has been applied:

```
$ sudo systemd-resolve --status
```

Copy

Scroll down until you see the section for your private network interface. You should see the private IP addresses for your DNS servers listed first, followed by some fallback values. Your domain should should be in the `DNS Domain`:

```
Output
. . .
Link 3 (eth1)
      Current Scopes: DNS
       LLMNR setting: yes
MulticastDNS setting: no
      DNSSEC setting: no
    DNSSEC supported: no
         DNS Servers: 10.128.10.11
                      10.128.20.12
                      67.207.67.2
                      67.207.67.3
          DNS Domain: nyc3.example.com
. . .
```

Your client should now be configured to use your internal DNS servers.

Ubuntu 16.04 and Debian Clients

On Ubuntu 16.04 and Debian Linux servers, you can edit the `/etc/network/interfaces` file:

```
$ sudo nano /etc/network/interfaces
```

Copy

Inside, find the `dns-nameservers` line, and prepend your own name servers in front of the list that is currently there. Below that line, add a `dns-search` option pointed to the base domain of your infrastructure. In our case, this would be `nyc3.example.com`:

/etc/network/interfaces

```
    . . .

    dns-nameservers 10.128.10.11 10.128.20.12 8.8.8.8
    dns-search nyc3.example.com

    . . .
```

Save and close the file when you are finished.

Now, restart your networking services, applying the new changes with the following commands. Make sure you replace `eth0` with the name of your networking interface:

```
$ sudo ifdown --force eth0 && sudo ip addr flush dev eth0 && sudo ifup --force eth0
```

Copy

This should restart your network without dropping your current connection. If it worked correctly, you should see something like this:

```
Output
RTNETLINK answers: No such process
Waiting for DAD... Done
```

Double check that your settings were applied by typing:

```
$ cat /etc/resolv.conf
```

Copy

You should see your name servers in the `/etc/resolv.conf` file, as well as your search domain:

```
Output
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.128.10.11
nameserver 10.128.20.12
nameserver 8.8.8.8
search nyc3.example.com
```

Your client is now configured to use your DNS servers.

CentOS Clients

On CentOS, RedHat, and Fedora Linux, edit the `/etc/sysconfig/network-scripts/ifcfg-eth0` file. You may have to substitute `eth0` with the name of your primary network interface:

```
$ sudo nano /etc/sysconfig/network-scripts/ifcfg-eth0
```
Copy

Search for the `DNS1` and `DNS2` options and set them to the private IP addresses of your primary and secondary name servers. Add a `DOMAIN` parameter followed by your infrastructure's base domain. In this guide, that would be `nyc3.example.com`:

/etc/sysconfig/network-scripts/ifcfg-eth0

```
. . .
DNS1=10.128.10.11
DNS2=10.128.20.12
DOMAIN='nyc3.example.com'
. . .
```

Save and close the file when you are finished.

Now, restart the networking service by typing:

```
$ sudo systemctl restart network
```
Copy

The command may hang for a few seconds, but should return you to the prompt shortly.

Check that your changes were applied by typing:

```
$ cat /etc/resolv.conf
```
Copy

You should see your name servers and search domain in the list:

/etc/resolv.conf

```
nameserver 10.128.10.11
nameserver 10.128.20.12
search nyc3.example.com
```

Your client should now be able to connect to and use your DNS servers.

Testing Clients

Use `nslookup` to test if your clients can query your name servers. You should be able to do this on all of the clients that you have configured and are in the `trusted` ACL.

For CentOS clients, you may need to install the utility with:

```
$ sudo yum install bind-utils
```
Copy

We can start by performing a forward lookup.

Forward Lookup

To perform a forward lookup to retrieve the IP address of `host1.nyc3.example.com`, run the following command:

```
$ nslookup host1
```
Copy

Querying `host1` expands to `host1.nyc3.example.com` because the `search` option is set to your private subdomain, and DNS queries will attempt to look on that subdomain before looking for the host elsewhere. The output of the previous command would look like the following:

```
Output
Server:     127.0.0.53
Address:    127.0.0.53#53

Non-authoritative answer:
Name:   host1.nyc3.example.com
Address: 10.128.100.101
```

Next, we can check reverse lookups.

Reverse Lookup

To test the reverse lookup, query the DNS server with **host1**'s private IP address:

```
$ nslookup 10.128.100.101
```
Copy

You should see output like the following:

```
Output
11.10.128.10.in-addr.arpa   name = host1.nyc3.example.com.

Authoritative answers can be found from:
```

If all of the names and IP addresses resolve to the correct values, that means that your zone files are configured properly. If you receive unexpected values, be sure to review the zone files on your primary DNS server (e.g. `db.nyc3.example.com` and `db.10.128`).

As a final step, we will go over how you can maintain your zone records.

Maintaining DNS Records

Now that you have a working internal DNS, you need to maintain your DNS records so they accurately reflect your server environment.

Adding a Host to DNS

Whenever you add a host to your environment (in the same datacenter), you will want to add it to DNS. Here is a list of steps that you need to take:

Primary Name Server

- Forward zone file: Add an `A` record for the new host, increment the value of `Serial`
- Reverse zone file: Add a `PTR` record for the new host, increment the value of `Serial`
- Add your new host's private IP address to the `trusted` ACL (`named.conf.options`)

Test your configuration files:

```
$ sudo named-checkconf
$ sudo named-checkzone nyc3.example.com db.nyc3.example.com
$ sudo named-checkzone 128.10.in-addr.arpa /etc/bind/zones/db.10.128
```

Copy

Then reload BIND:

```
$ sudo systemctl reload bind9
```

Copy

Your primary server should be configured for the new host now.

Secondary Name Server

- Add your new host's private IP address to the `trusted` ACL (`named.conf.options`)

Check the configuration syntax:

```
$ sudo named-checkconf
```

Copy

Then reload BIND:

```
$ sudo systemctl reload bind9
```

Copy

Your secondary server will now accept connections from the new host.

Configure New Host to Use Your DNS

- Configure `/etc/resolv.conf` to use your DNS servers
- Test using `nslookup`

Removing Host from DNS

If you remove a host from your environment or want to just take it out of DNS, just remove all the things that were added when you added the server to DNS (i.e. the reverse of the steps above).

Conclusion

Now you may refer to your servers' private network interfaces by name, rather than by IP address. This makes configuration of services and applications easier because you no longer have to remember the private IP addresses, and the files will be easier to read and understand. Also, now you can change your configurations to point to a new server in a single place, your primary DNS server, instead of having to edit a variety of distributed configuration files, which eases maintenance.

Once you have your internal DNS set up, and your configuration files are using private FQDNs to specify network connections, it is **critical** that your DNS servers are properly maintained. If they both become unavailable, your services and applications that rely on them will cease to function properly. This is why it is recommended to set up your DNS with at least one secondary server, and to maintain working backups of all of them.

About the authors

Justin Ellingwood Author
Developer and author at DigitalOcean.

Mitchell Anicas Author
Developer and author at DigitalOcean.

Still looking for an answer?

Ask a question   Search for more help

Was this helpful? Yes No

Comments

10 Comments

Leave a comment...

Login to Comment

aarona2davis • December 21, 2018
Excellent tutorial.

Regarding the section:

“Checking the BIND Configuration Syntax”

I’ve found when running the following command I receive these errors “zone zone.name/IN: loading from master file db.zone.name failed: file not found & zone zone.name/IN: not loaded due to errors.”

sudo named-checkzone nyc3.example.com db.nyc3.example.com

However if I append the path name, as you did in the second example for the reverse zone I receive no errors.

sudo named-checkzone nyc3.example.com /etc/bind/zones/db.nyc3.example.com

Should the tutorial be updated to reflect my changes? So far DNS is working as expected for me.

Thanks!

Reply

aarona2davis • December 21, 2018
Excellent tutorial.

Regarding the section:

“Checking the BIND Configuration Syntax”

I’ve found when running the following command I receive these errors “zone zone.name/IN: loading from master file db.zone.name failed: file not found & zone zone.name/IN: not loaded due to errors.”

sudo named-checkzone nyc3.example.com db.nyc3.example.com

However if I append the path name, as you did in the second example for the reverse zone I receive no errors.

sudo named-checkzone nyc3.example.com /etc/bind/zones/db.nyc3.example.com

Should the tutorial be updated to reflect my changes? So far DNS is working as expected for me.

Thanks!

Reply

Xenon032 • December 9, 2021

Hi, what about if I would reach a local apache website with a local domain? How to implement that with bind9? Thank you.

Reply

A.P. deBROUWER • October 3, 2020

i got inspired by this tutorial how to set up an internal DNS server

so i did implement this and made https://github.com/noud/infra-dns a working internal DNS server infrastructure on APT Linux.

Reply

pialislam2019 • September 19, 2020

I have face some error zone betfire247.com/IN: loading from master file db.betfire247.com failed: file not found zone betfire247.com/IN: not loaded due to errors.

Reply

rohitv • September 16, 2020

> > > /etc/default/bind9 OPTIONS=“-u bind -4”

In Ubuntu 20.04 - this file is /etc/default/named. Or better check which environment file is being used in systemd service.

cat /etc/systemd/system/bind9.service

[Service] EnvironmentFile=-/etc/default/named

Reply

gutierreznorwill • November 23, 2019

Hi great tutorial, is the same steps to create a Reverse DNS for a ISP or an Authoritative-Only DNS?

Thanks

Reply

Salim Sarımurat • May 27, 2019

Great tutorial, thank you!

After all the configurations and testing with `dig` tool, I encountered errors about **DNSSEC** for some domains and fixed them by adding the following lines to `/etc/bind/named.conf.options` file.

```
dnssec-enable yes;
dnssec-validation yes;
```

Reply

echanobe • December 19, 2018

Thank you.

For the ACL, can I put it in named.conf instead in named.conf.option?

Reply

Dario Fumagalli • July 20, 2018

Very great tutorial! I had some issues on Ubuntu 18.04 because the previous tutorial did not cover it. The new portion covering Netplan really saved my day!

Reply

Load More Comments

GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.

HOLLIE'S HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn More