CoreDNS is a DNS server that is designed to be flexible, extensible, and scalable. It is written in Go and is based on the popular Caddy web server. CoreDNS is designed to replace the traditional DNS servers like BIND and NSD, which can be difficult to configure and manage.

CoreDNS is modular, which means that it can be extended with plugins to add functionality. It can be used as a standalone server, or it can be integrated with other systems like Kubernetes, Docker, and etcd. CoreDNS is often used in cloud-native environments to provide DNS resolution for containerized applications.

One of the key features of CoreDNS is its flexibility. It supports a wide range of DNS protocols, including DNS-over-HTTPS (DoH) and DNS-over-TLS (DoT), which help to improve security and privacy. It also supports various middleware plugins that can be used to modify DNS responses on the fly, such as rewriting responses, adding or removing records, and rate limiting.

CoreDNS is open source software and is licensed under the Apache License 2.0. It has a large and active community of contributors and users, who continue to add new features and plugins to the project.

This scenario uses an interactive environment to explain how to configure and start a CoreDNS server. CoreDNS is a DNS service discovery for the cloud and aims to be a fast and flexible server.

In this scenario, CoreDNS will be populated based on a DNS zone file. The zone defines all the DNS entries for a particular domain.

Set Up CoreDNS
CoreDNS is a standalone, self-contained, binary. Built using Golang, the binary is available on OSX and Linux.

Download
Start downloading the binary using the following command:
curl -OL https://github.com/coredns/coredns/releases/download/v1.9.3/coredns_1.9.3_linux_amd64.tgz

The Zone File
A DNS zone file is a text file that describes the DNS configuration for a domain. The zone file defines a mapping between subdomains and IP addresses. This mapping can be used for defining subdomains such as www or Mail Exchanger (MX) records to which mail should be delivered.

The zone file can also contain additional settings to control aspects such as DNS caching.

Take a look at the contents of an example zone file for the domain example.com using cat db.example.com.

```
demo    IN  CNAME  www
$ cat db.example.com
$TTL    86400
$ORIGIN example.com.
@  1D  IN  SOA ns1.example.com. hostmaster.example.com. (
                            2002022401 ; serial
                            3H ; refresh
                            15 ; retry
                            1w ; expire
                            3h ; nxdomain ttl
                           )
       IN  NS     ns1.example.com. ; in the domain
       IN  NS     ns2.external-domain.org. ; external to domain
@      IN  MX  10 mail.example.com. ; external mail provider
ns1    IN  A      192.168.0.1
www    IN  A      192.168.0.2
admin  IN  A      192.168.0.3
demo   IN  CNAME  www
$
```

The contents of the file "db.example.com" appear to be a DNS zone file for the domain "example.com". Here is a breakdown of the different entries in the file:

$TTL 86400: This sets the Time-to-Live (TTL) value for the zone to 86400 seconds (1 day).
$ORIGIN example.com.: This sets the origin for the zone to "example.com.", which means that any unqualified names in the file (like "www" and "admin") will be interpreted as belonging to the "example.com." domain.

@ 1D IN SOA ns1.example.com. hostmaster.example.com. (...): This is the Start of Authority (SOA) record for the zone, which specifies the primary DNS server for the zone ("ns1.example.com."), the email address of the person responsible for the zone ("hostmaster.example.com."), and various other parameters like the serial number, refresh interval, and TTL for negative responses.
IN NS ns1.example.com.: This specifies that the DNS server "ns1.example.com." is authoritative for the zone.
IN NS ns2.external-domain.org.: This specifies a secondary DNS server that is external to the "example.com." domain.
@ IN MX 10 mail.example.com.: This specifies the mail exchanger (MX) record for the domain, which specifies where email messages should be delivered. In this case, it points to an external mail provider at "mail.example.com." with a priority of 10.
ns1 IN A 192.168.0.1: This specifies the IP address (A record) for the DNS server "ns1.example.com.".
www IN A 192.168.0.2: This specifies the IP address (A record) for the host "www.example.com.".
admin IN A 192.168.0.3: This specifies the IP address (A record) for the host "admin.example.com.".
demo IN CNAME www: This creates a Canonical Name (CNAME) record for the host "demo.example.com.", which points to the host "www.example.com.". This allows users to access the same content using different names.

Time-to-Live (TTL) is a value used in the Domain Name System (DNS) to determine how long a particular DNS record should be cached by a resolver or a caching DNS server before it expires and needs to be refreshed. TTL is expressed in seconds and is set by the owner of the DNS record.

When a DNS resolver or caching server receives a DNS response containing a record with a TTL value, it caches the record for the duration specified by the TTL. During this time, subsequent DNS queries for the same record can be answered from the cache, without the need to query the authoritative DNS server again. This helps to improve the performance of DNS queries and reduce the load on DNS servers.

Once the TTL value expires, the cached record is considered stale, and the resolver or caching server must query the authoritative DNS server again to obtain the latest version of the record. If the authoritative DNS server indicates that the record has changed since the last time it was queried, the resolver or caching server updates its cache with the new record and resets the TTL value.

TTL values can be set at various levels in the DNS hierarchy, including for the zone as a whole, individual resource records, and even individual DNS packets. The value of TTL can have a significant impact on the performance, reliability, and security of DNS, and it is important for DNS administrators to choose appropriate TTL values for their DNS records based on their specific needs and constraints.

An authoritative DNS server is a DNS server that is responsible for providing the definitive answer (or authoritative answer) to a DNS query for a particular domain or subdomain. An authoritative DNS server contains the original and official DNS records for the domain or subdomain it is authoritative for, and it is the final source of information for resolving DNS queries for that domain or subdomain.

When a DNS resolver receives a DNS query for a domain or subdomain, it typically sends the query to a recursive DNS server, which is responsible for resolving the query by querying various DNS servers in the DNS hierarchy. If the recursive DNS server cannot find the answer to the query in its cache or in any of the DNS servers it queries, it sends a query to one of the authoritative DNS servers for the domain or subdomain in question.

The authoritative DNS server receives the query, looks up the requested information in its own DNS records, and provides the correct answer to the recursive DNS server. The recursive DNS server then caches the answer and returns it to the original requester, which could be a web browser, an email client, or any other program that needs to resolve a domain name to an IP address.

Having an authoritative DNS server is important for ensuring the accuracy, consistency, and security of DNS records for a domain or subdomain. It is typically managed by the owner or administrator of the domain or subdomain and can be either self-hosted or provided by a third-party DNS service.

Zone File Structure

The first two lines define the domain for the zone and the (Time-To-Live) TTL for a cache of DNS requests. The next block defines the mapping between subdomains and IP addresses.

In this example, the domain has two name servers specified, ns1.example.com and ns2.external-domain.com. Name servers determine which servers can respond with the correct zone information for the particular domain. Domains should list at least two domains for redundancy.

The zone lists one Mail Exchanger (MX) record for mail, mail.example.com. An MX record specifies a mail server responsible for accepting email messages on behalf of a recipient's domain.

A Records are the most basic type of DNS record and are used to point a domain or subdomain to an IP address. This zone file defines three, ns1 pointing to the IP 192.168.0.1, www to 192.168.0.2 and admin to 192.168.0.3.

Finally, it defines one Canonical Name (CNAME) record called demo pointing to the www subdomain. A CNAME is an alias to another record.

Configuration
A structure text file defines the CoreDNS configuration. CoreDNS has a series of middleware layers that control its behavior and how to handle requests.

The configuration file also defines which zone Files to load and how to respond. CoreDNS supports multiple backend data sources for zone configuration. Other labs will explain the additional backend data sources.

CoreDNS Configuration
By default, a file called Corefile defines the configuration. View the configuration for this lab using cat Corefile.

```
$ cat Corefile
.:2053 {
  forward . 8.8.8.8:53
  file db.example.com example.com
  errors stdout
  log stdout
}
$
```

The contents of this file suggest that it is a configuration file for the CoreDNS DNS server. Here is a breakdown of the configuration:

The first line specifies the IP address and port number that CoreDNS will listen on for incoming DNS queries. In this case, it is listening on port 2053 of all available network interfaces on the system.

The next line begins the configuration block for handling incoming DNS queries. The dot (.) specifies that CoreDNS should handle queries for any domain name.

The "forward" directive tells CoreDNS to forward any queries that it cannot answer locally to the DNS server at IP address 8.8.8.8, which is Google's public DNS server.

The "file" directive specifies that CoreDNS should use the DNS records in the file "db.example.com" to answer queries for the domain name "example.com".

The "errors" directive specifies that error messages should be written to standard output (stdout).

The "log" directive specifies that query and response logs should be written to standard output (stdout).

Overall, this configuration is telling CoreDNS to listen on port 2053 for DNS queries, forward queries it cannot answer locally to Google's public DNS server, use the DNS records in "db.example.com" to answer queries for the "example.com" domain, and write logs and error messages to standard output.

The configuration has a few fundamental properties. First it defines that the DNS server should listen to requests on port 2053. It also defines which files to load; in this case, example.com.

All other requests are forwarded (proxied) to 8.8.8.8, Google's free and open DNS server. By using Google's DNS, you can ensure requests for public domains will be returned with the correct IP address.

Start the Task
Start the service as a background process using nohup:

nohup is a Unix command that stands for "no hangup". It is used to run a command or script and prevent it from being terminated when the user logs out or the terminal is closed.

Normally, when a command or script is executed from a terminal, it is associated with that terminal, and when the terminal is closed, the command or script is terminated along with it. However, when a command or script is executed with nohup, it is not associated with the terminal and is not affected by the terminal's status.

To use nohup, you simply prefix the command or script with nohup, like this:

nohup command or script &
The & at the end runs the command or script in the background, so that you can continue to use the terminal for other tasks.

When the command or script is executed with nohup, any output that would normally be sent to the terminal is redirected to a file called nohup.out in the current directory. This file contains the standard output and standard error streams of the command or script.

The nohup command is useful for running long-running tasks, such as server processes, backups, or data transfers, that need to continue running even after the user logs out or the terminal is closed.

This is a Unix command that uses nohup to run a program called "coredns" in the background with its standard output and standard error streams redirected to /dev/null, effectively suppressing any output from the program.

nohup ./coredns -quiet >/dev/null 2>&1 &
Here is a breakdown of the command:

nohup: Prefixes the command with nohup, which prevents the program from being terminated when the user logs out or the terminal is closed.
./coredns: Specifies the name of the program to run. This assumes that the program is located in the current directory and has executable permissions.
-quiet: Passes the -quiet flag to the coredns program, which tells it to run in quiet mode and suppress most log output.
>/dev/null: Redirects standard output to /dev/null, which discards all output from the program.
2>&1: Redirects standard error to the same place as standard output, which is /dev/null in this case.
&: Runs the command in the background, allowing the user to continue using the terminal for other tasks.
Overall, this command is launching the coredns program in the background, suppressing all output from the program, and allowing the user to continue using the terminal for other tasks. The nohup command ensures that the program continues running even if the user logs out or the terminal is closed.

In production, it's recommended that you use an init system such as SystemD:

Run a Query
The CoreDNS server is now running on port 2053 and configured for our example.com domain.

Query the DNS using the dig command. Querying for www.example.com will cause CoreDNS to look up the relevant IP address in the zone definition:

```
$ dig @localhost -p 2053 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @localhost -p 2053 www.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 45775
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 3b37c8bac3cec3b7 (echoed)
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        86400   IN      A       192.168.0.2

;; AUTHORITY SECTION:
example.com.            86400   IN      NS      ns1.example.com.
example.com.            86400   IN      NS      ns2.external-domain.org.

;; Query time: 0 msec
;; SERVER: 127.0.0.1#2053(127.0.0.1)
;; WHEN: Sun Apr 16 08:53:11 UTC 2023
;; MSG SIZE  rcvd: 175
```

For this subdomain, the DNS server returns IP address 192.168.0.2 as defined in our zone file.

Querying for the CNAME will return the hops as a single round-trip lookup:

dig @localhost -p 2053 demo.example.com

For the demo.example.com CNAME, the server first returns www.example.com and resolves it to the IP 192.168.0.2.

```
$ dig @localhost -p 2053 demo.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @localhost -p 2053 demo.example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43556
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: f36066775b5952f3 (echoed)
;; QUESTION SECTION:
;demo.example.com.              IN      A

;; ANSWER SECTION:
demo.example.com.       86400   IN      CNAME   www.example.com.
www.example.com.        86400   IN      A       192.168.0.2

;; AUTHORITY SECTION:
example.com.            86400   IN      NS      ns1.example.com.
example.com.            86400   IN      NS      ns2.external-domain.org.

;; Query time: 0 msec
;; SERVER: 127.0.0.1#2053(127.0.0.1)
;; WHEN: Sun Apr 16 08:53:55 UTC 2023
;; MSG SIZE  rcvd: 221
```

Applications are now able to use the DNS server to map well-known names to which server is capable of handling the requests.

If we query for an external hostname, then this will be proxied to Google's DNS server and return the correct IP addresses:

dig @localhost -p 2053 microsoft.com

```
$ dig @localhost -p 2053 microsoft.com

; <<>> DiG 9.16.1-Ubuntu <<>> @localhost -p 2053 microsoft.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58195
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;microsoft.com.                 IN      A

;; ANSWER SECTION:
microsoft.com.          1322    IN      A       20.112.52.29
microsoft.com.          1322    IN      A       20.81.111.85
microsoft.com.          1322    IN      A       20.84.181.62
microsoft.com.          1322    IN      A       20.103.85.33
microsoft.com.          1322    IN      A       20.53.203.50

;; Query time: 8 msec
;; SERVER: 127.0.0.1#2053(127.0.0.1)
;; WHEN: Sun Apr 16 08:54:22 UTC 2023
;; MSG SIZE  rcvd: 187
```

The dig command is a DNS tool used to query DNS servers for information about domain names, IP addresses, and other DNS-related information. It is a popular command-line tool on Unix-based operating systems, including Linux and macOS.

Here are some common uses of the dig command:

Querying a DNS server for a domain name's IP address: dig example.com
Querying a DNS server for a domain name's MX record (mail exchange record): dig example.com MX
Querying a specific DNS server for a domain name's NS record (name server record): dig example.com NS @8.8.8.8 (in this example, the @8.8.8.8 specifies the DNS server to query)
Querying a DNS server for a domain name's TXT record (text record): dig example.com TXT
Checking if a domain name's DNS is configured correctly: dig +trace example.com

dig can also be used with various options and flags to control the output format and behavior of the command. For example:

-t: Specifies the type of DNS record to query for (A, MX, NS, TXT, etc.)
-x: Reverse lookup mode, where you provide an IP address instead of a domain name to lookup the associated domain name
+short: Returns a more concise output format with only the IP address or domain name
+noall: Disables all output except for the final answer
+time: Specifies the maximum time to wait for a response from the DNS server, in seconds
Overall, dig is a powerful tool for troubleshooting DNS issues and gathering information about DNS configurations.