DNS Interface

One of the primary query interfaces for Consul is DNS. The DNS interface allows applications to make use of service discovery without any high-touch integration with Consul.

For example, instead of making HTTP API requests to Consul, a host can use the DNS server directly via name lookups like `redis.service.us-east-1.consul`. This query automatically translates to a lookup of nodes that provide the `redis` service, are located in the `us-east-1` datacenter, and have no failing health checks. It's that simple!

There are a number of configuration options that are important for the DNS interface, specifically `client_addr`, `ports.dns`, `recursors`, `domain`, and `dns_config`. By default, Consul will listen on 127.0.0.1:8600 for DNS queries in the `consul.` domain, without support for further DNS recursion. Please consult the documentation on configuration options, specifically the configuration items linked above, for more details.

There are a few ways to use the DNS interface. One option is to use a custom DNS resolver library and point it at Consul. Another option is to set Consul as the DNS server for a node and provide a `recursors` configuration so that non-Consul queries can also be resolved. The last method is to forward all queries for the "consul." domain to a Consul agent from the existing DNS server.

You can experiment with Consul's DNS server on the command line using tools such as `dig`:

```
$ dig @127.0.0.1 -p 8600 redis.service.dc1.consul. ANY
```

Note: In DNS, all queries are case-insensitive. A lookup of `PostgreSQL.node.dc1.consul` will find all nodes named `postgresql`.

Node Lookups

To resolve names, Consul relies on a very specific format for queries. There are fundamentally two types of queries: node lookups and service lookups. A node lookup, a simple query for the address of a named node, looks like this:

```
<node>.node[.datacenter].<domain>
```

For example, if we have a `foo` node with default settings, we could look for `foo.node.dc1.consul.` The datacenter is an optional part of the FQDN: if not provided, it defaults to the datacenter of the agent. If we know `foo` is running in the same datacenter as our local agent, we can instead use `foo.node.consul.` This convention allows for terse syntax where appropriate while supporting queries of nodes in remote datacenters as necessary.

For a node lookup, the only records returned are A records containing the IP address of the node.

```
$ dig @127.0.0.1 -p 8600 foo.node.consul ANY

; <<>> DiG 9.8.3-P1 <<>> @127.0.0.1 -p 8600 foo.node.consul ANY
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24355
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;foo.node.consul.       IN  ANY
```

```
;; ANSWER SECTION:
foo.node.consul.    0    IN   A    10.1.10.12

;; AUTHORITY SECTION:
consul.         0    IN   SOA ns.consul. postmaster.consul. 1392836399 3
```

Service Lookups

A service lookup is used to query for service providers. Service queries support two lookup methods: standard and strict RFC 2782.

Standard Lookup

The format of a standard service lookup is:

```
[tag.]<service>.service[.datacenter].<domain>
```

The `tag` is optional, and, as with node lookups, the `datacenter` is as well. If no tag is provided, no filtering is done on tag. If no datacenter is provided, the datacenter of this Consul agent is assumed.

If we want to find any redis service providers in our local datacenter, we could query `redis.service.consul`. If we want to find the PostgreSQL primary in a particular datacenter, we could query `primary.postgresql.service.dc2.consul`.

The DNS query system makes use of health check information to prevent routing to unhealthy nodes. When a service query is made, any services failing their health check or failing a node system check will be omitted from the results. To allow for simple load balancing, the set of nodes returned is also randomized each time. These mechanisms make it easy to use DNS along with application-level retries as the foundation for an auto-healing service oriented architecture.

For standard services queries, both A and SRV records are supported. SRV records provide the port that a service is registered on, enabling clients to avoid relying on well-known ports. SRV records are only served if the client specifically requests them, like so:

```
$ dig @127.0.0.1 -p 8600 consul.service.consul SRV

; <<>> DiG 9.8.3-P1 <<>> @127.0.0.1 -p 8600 consul.service.consul ANY
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50483
;; flags: qr aa rd; QUERY: 1, ANSWER: 3, AUTHORITY: 1, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;consul.service.consul.     IN  SRV

;; ANSWER SECTION:
consul.service.consul.  0   IN  SRV 1 1 8300 foobar.node.dc1.consul.

;; ADDITIONAL SECTION:
foobar.node.dc1.consul. 0   IN  A   10.1.10.12
```

RFC 2782 Lookup

The format for RFC 2782 SRV lookups is:

```
_<service>._<protocol>.service[.datacenter][.domain]
```

Per RFC 2782, SRV queries should use underscores, _, as a prefix to the `service` and `protocol` values in a query to prevent DNS collisions. The `protocol` value can be any of the tags for a service. If the service has no tags, `tcp` should be used. If `tcp` is specified as the protocol, the query will not perform any tag filtering.

Other than the query format and default `tcp` protocol/tag value, the behavior of the RFC style lookup is the same as the standard style of lookup.

If you registered the service `rabbitmq` on port 5672 and tagged it with `amqp`, you could make an RFC 2782 query for its SRV record as `_rabbitmq._amqp.service.consul`:

```
$ dig @127.0.0.1 -p 8600 _rabbitmq._amqp.service.consul SRV

; <<>> DiG 9.8.3-P1 <<>> @127.0.0.1 -p 8600 _rabbitmq._amqp.service.co
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52838
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;_rabbitmq._amqp.service.consul.    IN  SRV

;; ANSWER SECTION:
_rabbitmq._amqp.service.consul. 0   IN  SRV 1 1 5672 rabbitmq.node1.dc

;; ADDITIONAL SECTION:
rabbitmq.node1.dc1.consul.  0   IN  A   10.1.11.20
```

Again, note that the SRV record returns the port of the service as well as its IP.

Prepared Query Lookups

The format of a prepared query lookup is:

```
<query or name>.query[.datacenter].<domain>
```

The `datacenter` is optional, and if not provided, the datacenter of this Consul agent is assumed.

The `query or name` is the ID or given name of an existing Prepared Query. These behave like standard service queries but provide a much richer set of features, such as filtering by multiple tags and automatically failing over to look for services in remote datacenters if no healthy nodes are available in the local datacenter. Consul 0.6.4 and later also added support for prepared query templates which can match names using a prefix match, allowing one template to apply to potentially many services.

To allow for simple load balancing, the set of nodes returned is randomized each time. Both A and SRV records are supported. SRV records provide the port that a service is registered on, enabling clients to avoid relying on well-known ports. SRV records are only served if the client specifically requests them.

UDP Based DNS Queries

When the DNS query is performed using UDP, Consul will truncate the results without setting the truncate bit. This is to prevent a redundant lookup over TCP that generates additional load. If the lookup is done over TCP, the results are not truncated.

Caching

By default, all DNS results served by Consul set a 0 TTL value. This disables caching of DNS results. However, there are many situations in which caching is desirable for performance and scalability. This is discussed more in the guide for DNS Caching.

WAN Address Translation

By default, Consul DNS queries will return a node's local address, even when being queried from a remote datacenter. If you need to use a different address to reach a node from outside its datacenter, you can configure this behavior using the `advertise-wan` and `translate_wan_addrs` configuration options.