Consul Provider

Consul (https://www.consul.io) is a tool for service discovery, configuration and orchestration. The Consul provider exposes resources used to interact with a Consul cluster. Configuration of the provider is optional, as it provides defaults for all arguments.

Use the navigation to the left to read about the available resources.

NOTE: The Consul provider should not be confused with the Consul remote state backend (/docs/backends/types/consul.html), which is one of many backends that can be used to store Terraform state. The Consul provider is instead used to manage resources within Consul itself, such as adding external services or working with the key/value store.

Example Usage

```
# Configure the Consul provider
provider "consul" {
  address    = "demo.consul.io:80"
  datacenter = "nyc1"
}

# Access a key in Consul
data "consul_keys" "app" {
  key {
    name    = "ami"
    path    = "service/app/launch_ami"
    default = "ami-1234"
  }
}

# Use our variable from Consul
resource "aws_instance" "app" {
  ami = "${consul_keys.app.var.ami}"
}
```

Argument Reference

The following arguments are supported:

- `address` - (Optional) The HTTP(S) API address of the agent to use. Defaults to "127.0.0.1:8500".
- `scheme` - (Optional) The URL scheme of the agent to use ("http" or "https"). Defaults to "http".
- `http_auth` - (Optional) HTTP Basic Authentication credentials to be used when communicating with Consul, in the format of either `user` or `user:pass`. This may also be specified using the `CONSUL_HTTP_AUTH` environment variable.
- `datacenter` - (Optional) The datacenter to use. Defaults to that of the agent.
- `token` - (Optional) The ACL token to use by default when making requests to the agent. Can also be specified with

`CONSUL_HTTP_TOKEN` or `CONSUL_TOKEN` as an environment variable.

- `ca_file` - (Optional) A path to a PEM-encoded certificate authority used to verify the remote agent's certificate.
- `cert_file` - (Optional) A path to a PEM-encoded certificate provided to the remote agent; requires use of `key_file`.
- `key_file` - (Optional) A path to a PEM-encoded private key, required if `cert_file` is specified.
- `ca_path` - (Optional) A path to a directory of PEM-encoded certificate authority files to use to check the authenticity of client and server connections. Can also be specified with the `CONSUL_CAPATH` environment variable.
- `insecure_https` - (Optional) Boolean value to disable SSL certificate verification; setting this value to true is not recommended for production use. Only use this with scheme set to "https".

Environment Variables

All environment variables prefixed with `CONSUL_HTTP` listed in the Consul environment variables (https://www.consul.io/docs/commands/index.html#environment-variables) documentation are supported by the Terraform provider.

consul_acl_auth_method

The `consul_acl_auth_method` data source returns the information related to a Consul Auth Method (https://www.consul.io/docs/acl/acl-auth-methods.html).

Example Usage

```
data "consul_acl_auth_method" "test" {
  name = "minikube"
}

output "consul_acl_auth_method" {
  value = data.consul_acl_auth_method.test.config
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the ACL Auth Method.

Attributes Reference

The following attributes are exported:

- `description` - The description of the ACL Auth Method.
- `type` - The type of the ACL Auth Method.
- `config` - The configuration options of the ACL Auth Method.

consul_acl_policy

The `consul_acl_policy` data source returns the information related to a Consul ACL Policy (https://www.consul.io/docs/acl/acl-system.html#acl-policies).

Example Usage

```
data "consul_acl_policy" "agent" {
  name  = "agent"
}

output "consul_acl_policy" {
  value = data.consul_acl_policy.agent.rules
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the ACL Policy.

Attributes Reference

The following attributes are exported:

- `description` - The description of the ACL Policy.
- `rules` - The rules associated with the ACL Policy.
- `datacenters` - The datacenters associated with the ACL Policy.

consul_acl_role

The `consul_acl_role` data source returns the information related to a Consul ACL Role (https://www.consul.io/api/acl/roles.html).

Example Usage

```
data "consul_acl_role" "test" {
  name = "example-role"
}

output "consul_acl_role" {
  value = data.consul_acl_role.test.id
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the ACL Role.

Attributes Reference

The following attributes are exported:

- `description` - The description of the ACL Role.
- `policies` - The list of policies associated with the ACL Role. Each entry has an `id` and a `name` attribute.
- `service_identities` - The list of service identities associated with the ACL Role. Each entry has a `service_name` attribute and a list of `datacenters`.

consul_acl_token

The `consul_acl_token` data source returns the information related to the `consul_acl_token` resource with the exception of its secret ID.

If you want to get the secret ID associated with a token, use the `consul_acl_token_secret_id` data source (/docs/providers/consul/d/acl_token_secret_id.html).

Example Usage

```
data "consul_acl_token" "test" {
  accessor_id = "00000000-0000-0000-0000-000000000002"
}

output "consul_acl_policies" {
  value = "${data.consul_acl_token.test.policies}"
}
```

Argument Reference

The following arguments are supported:

- `accessor_id` - (Required) The accessor ID of the ACL token.

Attributes Reference

The following attributes are exported:

- `description` - The description of the ACL token.
- `policies` - A list of policies associated with the ACL token. Each entry has an `id` and a `name` attribute.
- `local` - Whether the ACL token is local to the datacenter it was created within.

consul_acl_token_secret_id

Warning: When using this is resource, the ACL Token secret ID will be written to the Terraform state. It is strongly recommended to use the `gpg_key` attribute and to make sure the remote state has strong access controls before using this resource.

The `consul_acl_token_secret` data source returns the secret ID associated to the accessor ID. This can be useful to make systems that cannot use an auth method to interface with Consul.

If you want to get other attributes of the Consul ACL token, please use the `consul_acl_token` data source (/docs/providers/consul/d/acl_token.html).

Example Usage

```
resource "consul_acl_policy" "test" {
    name = "test"
    rules = "node \"\" { policy = \"read\" }"
    datacenters = [ "dc1" ]
}

resource "consul_acl_token" "test" {
    description = "test"
    policies = ["${consul_acl_policy.test.name}"]
    local = true
}

data "consul_acl_token_secret_id" "read" {
    accessor_id = "${consul_acl_token.test.id}"
    gpg_key     = "keybase:my_username"
}

output "consul_acl_token_secret_id" {
  value = "${data.consul_acl_token.read.encrypted_secret_id}"
}
```

Argument Reference

The following arguments are supported:

- `accessor_id` - (Required) The accessor ID of the ACL token.
- `gpg_key` - (Optional) Either a base-64 encoded PGP public key, or a keybase username in the form `keybase:some_person_that_exists`. **If you do not set this argument, the token secret ID will be written as plain text in the Terraform state.**

Attributes Reference

The following attributes are exported:

- `secret_id` - The secret ID of the ACL token if `gpg_key` has not been set.
- `encrypted_secret_id` - The encrypted secret ID of the ACL token if `gpg_key` has been set. You can decrypt the secret by using the command line, for example with: `terraform output encrypted_secret | base64 --decode | keybase pgp decrypt`.

consul_agent_config

Note: The `consul_agent_config` resource differs from `consul_agent_self` (/docs/providers/consul/d/agent_self.html), providing less information but utilizing stable APIs. `consul_agent_self` will be deprecated in a future release.

The `consul_agent_config` data source returns configuration data (https://www.consul.io/api/agent.html#read-configuration) from the agent specified in the `provider`.

Example Usage

```
data "consul_agent_config" "remote_agent" {}

output "consul_version" {
  value = "${data.consul_agent_config.remote_agent.version}"
}
```

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the agent is running in
- `node_id` - The ID of the node the agent is running on
- `node_name` - The name of the node the agent is running on
- `server` - Boolean if the agent is a server or not
- `revision` - The first 9 characters of the VCS revision of the build of Consul that is running
- `version` - The version of the build of Consul that is running

consul_agent_self

Warning: The `consul_agent_self` resource has been deprecated and will be removed from a future release of the provider. Read the upgrade instructions (/docs/providers/consul/upgrading.html#deprecation-of-consul_agent_self) for more information.

The `consul_agent_self` data source returns configuration and status data (https://www.consul.io/docs/agent/http/agent.html#agent_self) from the agent specified in the `provider`.

Example Usage

```
data "consul_agent_self" "read-dc1-agent" {
  query_options {
    # Optional parameter: implicitly uses the current datacenter of the agent
    datacenter = "dc1"
  }
}

# Set the description to a whitespace delimited list of the services
resource "example_resource" "app" {
  description = "Consul datacenter ${data.consul_agent_self.read-dc1-agent.datacenter}"

  # ...
}
```

Attributes Reference

The following attributes are exported:

- `acl_datacenter` (https://www.consul.io/docs/agent/options.html#acl_datacenter)
- `acl_default_policy` (https://www.consul.io/docs/agent/options.html#acl_default_policy)
- `acl_disabled_ttl`
- `acl_down_policy` (https://www.consul.io/docs/agent/options.html#acl_down_policy)
- `acl_enforce_0_8_semantics` (https://www.consul.io/docs/agent/options.html#acl_enforce_version_8)
- `acl_ttl` (https://www.consul.io/docs/agent/options.html#acl_ttl)
- `addresses` (https://www.consul.io/docs/agent/options.html#addresses)
- `advertise_addr` (https://www.consul.io/docs/agent/options.html#_advertise)
- `advertise_addr_wan` (https://www.consul.io/docs/agent/options.html#_advertise-wan)
- `advertise_addrs` (https://www.consul.io/docs/agent/options.html#advertise_addrs)
- `atlas_join` (https://www.consul.io/docs/agent/options.html#_atlas_join)

- `bind_addr` (https://www.consul.io/docs/agent/options.html#_bind)
- `bootstrap_expect` (https://www.consul.io/docs/agent/options.html#_bootstrap_expect)
- `bootstrap_mode` (https://www.consul.io/docs/agent/options.html#_bootstrap)
- `check_deregister_interval_min`
- `check_reap_interval`
- `check_update_interval` (https://www.consul.io/docs/agent/options.html#check_update_interval)
- `client_addr` (https://www.consul.io/docs/agent/options.html#_client)
- `dns` - A map of DNS configuration attributes. See below for details on the contents of the `dns` attribute.
- `dns_recursors` (https://www.consul.io/docs/agent/options.html#recursors) - A list of all DNS recursors.
- `data_dir` (https://www.consul.io/docs/agent/options.html#_data_dir)
- `datacenter` (https://www.consul.io/docs/agent/options.html#_datacenter)
- `dev_mode` (https://www.consul.io/docs/agent/options.html#_dev)
- `domain` (https://www.consul.io/docs/agent/options.html#_domain)
- `enable_anonymous_signature` (https://www.consul.io/docs/agent/options.html#disable_anonymous_signature)
- `enable_coordinates`
- `enable_debug` (https://www.consul.io/docs/agent/options.html#enable_debug)
- `enable_remote_exec` (https://www.consul.io/docs/agent/options.html#disable_remote_exec)
- `enable_syslog` (https://www.consul.io/docs/agent/options.html#_syslog)
- `enable_ui` (https://www.consul.io/docs/agent/options.html#_ui)
- `enable_update_check` (https://www.consul.io/docs/agent/options.html#disable_update_check)
- `id` (https://www.consul.io/docs/agent/options.html#_node_id)
- `leave_on_int` (https://www.consul.io/docs/agent/options.html#skip_leave_on_interrupt)
- `leave_on_term` (https://www.consul.io/docs/agent/options.html#leave_on_terminate)
- `log_level` (https://www.consul.io/docs/agent/options.html#_log_level)
- `name` (https://www.consul.io/docs/agent/options.html#_node)
- `performance` (https://www.consul.io/docs/agent/options.html#performance)
- `pid_file` (https://www.consul.io/docs/agent/options.html#_pid_file)
- `ports` (https://www.consul.io/docs/agent/options.html#ports)
- `protocol_version` (https://www.consul.io/docs/agent/options.html#_protocol)
- `reconnect_timeout_lan` (https://www.consul.io/docs/agent/options.html#reconnect_timeout)
- `reconnect_timeout_wan` (https://www.consul.io/docs/agent/options.html#reconnect_timeout_wan)

- `rejoin_after_leave` (https://www.consul.io/docs/agent/options.html#_rejoin)
- `retry_join` (https://www.consul.io/docs/agent/options.html#retry_join)
- `retry_join_ec2` (https://www.consul.io/docs/agent/options.html#retry_join_ec2) - A map of EC2 retry attributes. See below for details on the available information.
- `retry_join_gce` (https://www.consul.io/docs/agent/options.html#retry_join_gce) - A map of GCE retry attributes. See below for details on the available information.
- `retry_join_wan` (https://www.consul.io/docs/agent/options.html#_retry_join_wan)
- `retry_max_attempts` (https://www.consul.io/docs/agent/options.html#_retry_max)
- `retry_max_attempts_wan` (https://www.consul.io/docs/agent/options.html#_retry_max_wan)
- `serf_lan_bind_addr` (https://www.consul.io/docs/agent/options.html#_serf_lan_bind)
- `serf_wan_bind_addr` (https://www.consul.io/docs/agent/options.html#_serf_wan_bind)
- `server_mode` (https://www.consul.io/docs/agent/options.html#_server)
- `server_name` (https://www.consul.io/docs/agent/options.html#server_name)
- `session_ttl_min` (https://www.consul.io/docs/agent/options.html#session_ttl_min)
- `start_join` (https://www.consul.io/docs/agent/options.html#start_join)
- `start_join_wan` (https://www.consul.io/docs/agent/options.html#start_join_wan)
- `syslog_facility` (https://www.consul.io/docs/agent/options.html#syslog_facility)
- `tls_ca_file` (https://www.consul.io/docs/agent/options.html#ca_file)
- `tls_cert_file` (https://www.consul.io/docs/agent/options.html#cert_file)
- `tls_key_file` (https://www.consul.io/docs/agent/options.html#key_file)
- `tls_min_version` (https://www.consul.io/docs/agent/options.html#tls_min_version)
- `tls_verify_incoming` (https://www.consul.io/docs/agent/options.html#verify_incoming)
- `tls_verify_outgoing` (https://www.consul.io/docs/agent/options.html#verify_outgoing)
- `tls_verify_server_hostname` (https://www.consul.io/docs/agent/options.html#verify_server_hostname)
- `tagged_addresses` (https://www.consul.io/docs/agent/options.html#translate_wan_addrs)
- `telemetry` (https://www.consul.io/docs/agent/options.html#telemetry) - A map of telemetry configuration.
- `translate_wan_addrs` (https://www.consul.io/docs/agent/options.html#translate_wan_addrs)
- `ui_dir` (https://www.consul.io/docs/agent/options.html#ui_dir)
- `unix_sockets` (https://www.consul.io/docs/agent/options.html#unix_sockets)
- `version` - The version of the Consul agent.
- `version_prerelease`
- `version_revision`

DNS Attributes

- `allow_stale` (https://www.consul.io/docs/agent/options.html#allow_stale)
- `enable_compression` (https://www.consul.io/docs/agent/options.html#disable_compression)
- `enable_truncate` (https://www.consul.io/docs/agent/options.html#enable_truncate)
- `max_stale` (https://www.consul.io/docs/agent/options.html#max_stale)
- `node_ttl` (https://www.consul.io/docs/agent/options.html#node_ttl)
- `only_passing` (https://www.consul.io/docs/agent/options.html#only_passing)
- `recursor_timeout` (https://www.consul.io/docs/agent/options.html#recursor_timeout)
- `service_ttl` (https://www.consul.io/docs/agent/options.html#service_ttl)
- `udp_answer_limit` (https://www.consul.io/docs/agent/options.html#udp_answer_limit)

Retry Join EC2 Attributes

- `region` (https://www.consul.io/docs/agent/options.html#region)
- `tag_key` (https://www.consul.io/docs/agent/options.html#tag_key)
- `tag_value` (https://www.consul.io/docs/agent/options.html#tag_value)

Retry Join GCE Attributes

- `credentials_file` (https://www.consul.io/docs/agent/options.html#credentials_file)
- `project_name` (https://www.consul.io/docs/agent/options.html#project_name)
- `tag_value` (https://www.consul.io/docs/agent/options.html#tag_value)
- `zone_pattern` (https://www.consul.io/docs/agent/options.html#zone_pattern)

Telemetry Attributes

- `circonus_api_app` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_api_app)
- `circonus_api_token` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_api_token)
- `circonus_api_url` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_api_url)
- `circonus_broker_id` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_broker_id)
- `circonus_check_id` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_check_id)
- `circonus_check_tags` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_check_tags)
- `circonus_display_name` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_check_display_name)
- `circonus_force_metric_activation` (https://www.consul.io/docs/agent/options.html#telemetry-

circonus_check_force_metric_activation)

- `circonus_instance_id` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_check_instance_id)
- `circonus_search_tag` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_check_search_tag)
- `circonus_select_tag` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_broker_select_tag)
- `circonus_submission_interval` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_submission_interval)
- `circonus_submission_url` (https://www.consul.io/docs/agent/options.html#telemetry-circonus_submission_url)
- `dogstatsd_addr` (https://www.consul.io/docs/agent/options.html#telemetry-dogstatsd_addr)
- `dogstatsd_tags` (https://www.consul.io/docs/agent/options.html#telemetry-dogstatsd_tags)
- `enable_hostname` (https://www.consul.io/docs/agent/options.html#telemetry-disable_hostname)
- `statsd_addr` (https://www.consul.io/docs/agent/options.html#telemetry-statsd_address)
- `statsite_addr` (https://www.consul.io/docs/agent/options.html#telemetry-statsite_address)
- `statsite_prefix` (https://www.consul.io/docs/agent/options.html#telemetry-statsite_prefix)

consul_autopilot_health

The `consul_autopilot_health` data source returns autopilot health information (https://www.consul.io/api/operator/autopilot.html#read-health) about the current Consul cluster.

Example Usage

```
data "consul_autopilot_health" "read" {}

output "health" {
  value = "${data.consul_autopilot_health.read.healthy}"
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.

Attributes Reference

The following attributes are exported:

- `healthy` - Whether all the servers in the cluster are currently healthy
- `failure_tolerance` - The number of redundant healthy servers that could fail without causing an outage
- `servers` - A list of server health information. See below for details on the available information.

Server health information

- `id` - The Raft ID of the server
- `name` - The node name of the server
- `address` - The address of the server
- `serf_status` - The status of the SerfHealth check of the server
- `version` - The Consul version of the server
- `leader` - Whether the server is currently leader
- `last_contact` - The time elapsed since the server's last contact with the leader
- `last_term` - The server's last known Raft leader term

- `last_index` - The index of the server's last committed Raft log entry
- `healthy` - Whether the server is healthy according to the current Autopilot configuration
- `voter` - Whether the server is a voting member of the Raft cluster
- `stable_since` - The time this server has been in its current `Healthy` state

consul_key_prefix

Allows Terraform to read values from a "namespace" of Consul keys that share a common name prefix.

Example Usage

```
data "consul_key_prefix" "app" {
  datacenter = "nyc1"
  token      = "abcd"

  # Prefix to add to prepend to all of the subkey names below.
  path_prefix = "myapp/config/"

  # Read the ami subkey
  subkey {
    name    = "ami"
    path    = "app/launch_ami"
    default = "ami-1234"
  }
}

# Start our instance with the dynamic ami value
resource "aws_instance" "app" {
  ami = "${data.consul_key_prefix.app.var.ami}"

  # ...
}
```

```
data "consul_key_prefix" "web" {
  datacenter = "nyc1"
  token      = "efgh"

  # Prefix to add to prepend to all of the subkey names below.
  path_prefix = "myapp/config/"
}

# Start our instance with the dynamic ami value
resource "aws_instance" "web" {
  ami = "${data.consul_key_prefix.web.subkeys["app/launch_ami"]}"

  # ...
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.

- `token` - (Optional) The ACL token to use. This overrides the token that the agent provides by default.
- `path_prefix` - (Required) Specifies the common prefix shared by all keys that will be read by this data source instance. In most cases, this will end with a slash to read a "folder" of subkeys.
- `subkey` - (Optional) Specifies a subkey in Consul to be read. Supported values documented below. Multiple blocks supported.

The `subkey` block supports the following:

- `name` - (Required) This is the name of the key. This value of the key is exposed as `var.<name>` . This is not the path of the subkey in Consul.
- `path` - (Required) This is the subkey path in Consul (which will be appended to the given `path_prefix`) to construct the full key that will be used to read the value.
- `default` - (Optional) This is the default value to set for `var.<name>` if the key does not exist in Consul. Defaults to an empty string.

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the keys are being read from.
- `path_prefix` - the common prefix shared by all keys being read.
- `var.<name>` - For each name given, the corresponding attribute has the value of the key.
- `subkeys` - A map of the subkeys and values is set if no `subkey` block is provided.

consul_keys

The `consul_keys` resource reads values from the Consul key/value store. This is a powerful way dynamically set values in templates.

Example Usage

```
data "consul_keys" "app" {
  datacenter = "nyc1"
  token      = "abcd"

  # Read the launch AMI from Consul
  key {
    name    = "ami"
    path    = "service/app/launch_ami"
    default = "ami-1234"
  }
}

# Start our instance with the dynamic ami value
resource "aws_instance" "app" {
  ami = "${data.consul_keys.app.var.ami}"

  # ...
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.
- `token` - (Optional) The ACL token to use. This overrides the token that the agent provides by default.
- `key` - (Required) Specifies a key in Consul to be read. Supported values documented below. Multiple blocks supported.

The `key` block supports the following:

- `name` - (Required) This is the name of the key. This value of the key is exposed as `var.<name>` . This is not the path of the key in Consul.
- `path` - (Required) This is the path in Consul that should be read or written to.
- `default` - (Optional) This is the default value to set for `var.<name>` if the key does not exist in Consul. Defaults to an empty string.

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the keys are being read from.
- `var.<name>` - For each name given, the corresponding attribute has the value of the key.

consul_nodes

The `consul_nodes` data source returns a list of Consul nodes that have been registered with the Consul cluster in a given datacenter. By specifying a different datacenter in the `query_options` it is possible to retrieve a list of nodes from a different WAN-attached Consul datacenter.

Example Usage

```
data "consul_nodes" "read-dc1-nodes" {
  query_options {
    # Optional parameter: implicitly uses the current datacenter of the agent
    datacenter = "dc1"
  }
}

# Set the description to a whitespace delimited list of the node names
resource "example_resource" "app" {
  description = "${join(" ", formatlist("%s", data.consul_nodes.node_names))}"

  # ...
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The Consul datacenter to query. Defaults to the same value found in `query_options` parameter specified below, or if that is empty, the `datacenter` value found in the Consul agent that this provider is configured to talk to then the datacenter in the provider setup.
- `query_options` - (Optional) See below.

The `query_options` block supports the following:

- `allow_stale` - (Optional) When `true` , the default, allow responses from Consul servers that are followers.
- `require_consistent` - (Optional) When `true` force the client to perform a read on at least quorum servers and verify the result is the same. Defaults to `false` .
- `token` - (Optional) Specify the Consul ACL token to use when performing the request. This defaults to the same API token configured by the `consul` provider but may be overriden if necessary.
- `wait_index` - (Optional) Index number used to enable blocking quereis.
- `wait_time` - (Optional) Max time the client should wait for a blocking query to return.

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the keys are being read from to.
- `node_ids` - A list of the Consul node IDs.
- `node_names` - A list of the Consul node names.
- `nodes` - A list of nodes and details about each Consul agent. The list of per-node attributes is detailed below.

The following is a list of the per-node attributes contained within the `nodes` map:

- `id` - The Node ID of the Consul agent.
- `meta` (https://www.consul.io/docs/agent/http/catalog.html#Meta) - Node meta data tag information, if any.
- `name` (https://www.consul.io/docs/agent/http/catalog.html#Node) - The name of the Consul node.
- `address` (https://www.consul.io/docs/agent/http/catalog.html#Address) - The IP address the node is advertising to the Consul cluster.
- `tagged_addresses` (https://www.consul.io/docs/agent/http/catalog.html#TaggedAddresses) - List of explicit LAN and WAN IP addresses for the agent.

consul_service_health

`consul_service_health` can be used to get the list of the instances that are currently healthy, according to their associated health-checks. The result includes the list of service instances, the node associated to each instance and its health-checks.

This resource is likely to change as frequently as the health-checks are being updated, you should expect different results in a frequent basis.

Example Usage

```
provider "consul" {}

data "consul_service_health" "vault" {
  service = "vault"
  passing = true
}

provider "vault" {
  address = "https://${data.consul_service_health.vault.results.0.service.0.address}:${data.consul_servic
e_health.vault.results.0.service.0.port}"
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The Consul datacenter to query.
- `name` - (Required) The service name to select.
- `near` - (Optional) Specifies a node name to sort the node list in ascending order based on the estimated round trip time from that node.
- `tag` - (Optional) A single tag that can be used to filter the list to return based on a single matching tag.
- `node_meta` - (Optional) Filter the results to nodes with the specified key/value pairs.
- `passing` - (Optional) Whether to return only nodes with all checks in the passing state. Defaults to `true`.

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the keys are being read from to.
- `name` - The name of the service.
- `near` - The node to which the result must be sorted to.

- `tag` - The name of the tag used to filter the list.
- `node_meta` - The list of metadata to filter the nodes.
- `passing` - Whether to return only nodes with all checks in the passing state.
- `results` - A list of entries and details about each endpoint advertising a service. Each element in the list has three attributes: `node`, `service` and `checks`. The list of the attributes of each one is detailed below.

The following is a list of the per-entry `node` attributes:

- `id` - The Node ID of the Consul node advertising the service.
- `name` - The name of the node.
- `address` - The address of the node.
- `datacenter` - The datacenter in which the node is running.
- `tagged_addresses` (https://www.consul.io/docs/agent/http/catalog.html#TaggedAddresses) - List of explicit LAN and WAN IP addresses for the agent.
- `meta` - Node meta data tag information, if any.

The following is a list of the per-entry `service` attributes:

- `id` - The ID of the service.
- `name` - The name of the service.
- `tags` - The list of tags associated with this instance.
- `address` - The address of this instance.
- `port` - The port of this instance.
- `meta` - Service metadata tag information, if any.

`checks` is a list of the health-checks associated to the entry with the following attributes:

- `id` - The ID of this health-check.
- `node` - The name of the node associated with this health-check.
- `name` - The name of this health-check.
- `status` - The status of this health-check.
- `notes` - A human readable description of the current state of the health-check.
- `output` - The output of the health-check.
- `service_id` - The ID of the service associated to this health-check.
- `service_name` - The name of the service associated with this health-check.
- `service_tags` - The list of tags associated with this health-check.

consul_service

`consul_service` provides details about a specific Consul service in a given datacenter. The results include a list of nodes advertising the specified service, the node's IP address, port number, node ID, etc. By specifying a different datacenter in the `query_options` it is possible to retrieve a list of services from a different WAN-attached Consul datacenter.

This data source is different from the `consul_services` (plural) data source, which provides a summary of the current Consul services.

Example Usage

```
data "consul_service" "read-consul-dc1" {
    name = "consul"
    # Optional parameter: implicitly uses the current datacenter of the agent
    datacenter = "dc1"
}

# Set the description to a whitespace delimited list of the node names
resource "example_resource" "app" {
  description = "${join(" ", data.consul_service.nodes)}"

  # ...
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The Consul datacenter to query. Defaults to the same value found in `query_options` parameter specified below, or if that is empty, the `datacenter` value found in the Consul agent that this provider is configured to talk to.
- `name` - (Required) The service name to select.
- `query_options` - (Optional) See below.
- `tag` - (Optional) A single tag that can be used to filter the list of nodes to return based on a single matching tag..

The `query_options` block supports the following:

- `allow_stale` - (Optional) When `true` , the default, allow responses from Consul servers that are followers.
- `require_consistent` - (Optional) When `true` force the client to perform a read on at least quorum servers and verify the result is the same. Defaults to `false` .
- `token` - (Optional) Specify the Consul ACL token to use when performing the request. This defaults to the same API token configured by the `consul` provider but may be overriden if necessary.
- `wait_index` - (Optional) Index number used to enable blocking quereis.
- `wait_time` - (Optional) Max time the client should wait for a blocking query to return.

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the keys are being read from to.
- `name` - The name of the service
- `tag` - The name of the tag used to filter the list of nodes in `service`.
- `service` - A list of nodes and details about each endpoint advertising a service. Each element in the list is a map of attributes that correspond to each individual node. The list of per-node attributes is detailed below.

The following is a list of the per-node `service` attributes:

- `create_index` (https://www.consul.io/docs/agent/http/catalog.html#CreateIndex) - The index entry at which point this entry was added to the catalog.
- `modify_index` (https://www.consul.io/docs/agent/http/catalog.html#ModifyIndex) - The index entry at which point this entry was modified in the catalog.
- `node_address` (https://www.consul.io/docs/agent/http/catalog.html#Address) - The address of the Consul node advertising the service.
- `node_id` - The Node ID of the Consul agent advertising the service.
- `node_meta` (https://www.consul.io/docs/agent/http/catalog.html#Meta) - Node meta data tag information, if any.
- `node_name` (https://www.consul.io/docs/agent/http/catalog.html#Node) - The name of the Consul node.
- `address` (https://www.consul.io/docs/agent/http/catalog.html#ServiceAddress) - The IP address of the service. If the `ServiceAddress` in the Consul catalog is empty, this value is automatically populated with the `node_address` (the `Address` in the Consul Catalog).
- `enable_tag_override` (https://www.consul.io/docs/agent/http/catalog.html#ServiceEnableTagOverride) - Whether service tags can be overridden on this service.
- `id` (https://www.consul.io/docs/agent/http/catalog.html#ServiceID) - A unique service instance identifier.
- `name` (https://www.consul.io/docs/agent/http/catalog.html#ServiceName) - The name of the service.
- `port` (https://www.consul.io/docs/agent/http/catalog.html#ServicePort) - Port number of the service.
- `tagged_addresses` (https://www.consul.io/docs/agent/http/catalog.html#TaggedAddresses) - List of explicit LAN and WAN IP addresses for the agent.
- `tags` (https://www.consul.io/docs/agent/http/catalog.html#ServiceTags) - List of tags for the service.
- `meta` (https://www.consul.io/docs/agent/http/catalog.html#Meta) - Service meta data tag information, if any.

consul_services

The `consul_services` data source returns a list of Consul services that have been registered with the Consul cluster in a given datacenter. By specifying a different datacenter in the `query_options` it is possible to retrieve a list of services from a different WAN-attached Consul datacenter.

This data source is different from the `consul_service` (singular) data source, which provides a detailed response about a specific Consul service.

Example Usage

```
data "consul_services" "read-dc1" {
  query_options {
    # Optional parameter: implicitly uses the current datacenter of the agent
    datacenter = "dc1"
  }
}

# Set the description to a whitespace delimited list of the services
resource "example_resource" "app" {
  description = "${join(" ", data.consul_services.names)}"

  # ...
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The Consul datacenter to query. Defaults to the same value found in `query_options` parameter specified below, or if that is empty, the `datacenter` value found in the Consul agent that this provider is configured to talk to.
- `query_options` - (Optional) See below.

The `query_options` block supports the following:

- `allow_stale` - (Optional) When `true` , the default, allow responses from Consul servers that are followers.
- `require_consistent` - (Optional) When `true` force the client to perform a read on at least quorum servers and verify the result is the same. Defaults to `false` .
- `token` - (Optional) Specify the Consul ACL token to use when performing the request. This defaults to the same API token configured by the `consul` provider but may be overriden if necessary.
- `wait_index` - (Optional) Index number used to enable blocking quereis.
- `wait_time` - (Optional) Max time the client should wait for a blocking query to return.

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the keys are being read from to.
- `names` - A list of the Consul services found. This will always contain the list of services found.
- `services.<service>` - For each name given, the corresponding attribute is a Terraform map of services and their tags. The value is an alphanumerically sorted, whitespace delimited set of tags associated with the service.
- `tags` - A map of the tags found for each service. If more than one service shares the same tag, unique service names will be joined by whitespace (this is the inverse of `services` and can be used to lookup the services that match a single tag).

consul_acl_auth_method

Starting with Consul 1.5.0, the consul_acl_auth_method resource can be used to managed Consul ACL auth methods.

Example Usage

```
resource "consul_acl_auth_method" "minikube" {
    name        = "minikube"
    type        = "kubernetes"
    description = "dev minikube cluster"

    config = {
        Host = "https://192.0.2.42:8443"
        CACert = "-----BEGIN CERTIFICATE-----\n...-----END CERTIFICATE-----\n"
        ServiceAccountJWT = "eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9..."
    }
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the ACL auth method.
- `type` - (Required) The type of the ACL auth method.
- `description` - (Optional) A free form human readable description of the auth method.
- `config` - (Required) The raw configuration for this ACL auth method.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the the auth method.
- `name` - The name of the ACL auth method.
- `type` - The type of the ACL auth method.
- `description` - A free form human readable description of the auth method.
- `config` - The raw configuration for this ACL auth method.

consul_acl_binding_rule

Starting with Consul 1.5.0, the consul_acl_binding_rule resource can be used to managed Consul ACL binding rules.

Example Usage

```
resource "consul_acl_auth_method" "minikube" {
    name        = "minikube"
    type        = "kubernetes"
    description = "dev minikube cluster"

    config = {
        Host = "https://192.0.2.42:8443"
        CACert = "-----BEGIN CERTIFICATE-----\n...-----END CERTIFICATE-----\n"
        ServiceAccountJWT = "eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9..."
    }
}

resource "consul_acl_binding_rule" "test" {
    auth_method = "${consul_acl_auth_method.minikube.name}"
    description = "foobar"
    selector    = "serviceaccount.namespace==default"
    bind_type   = "service"
    bind_name   = "minikube"
}
```

Argument Reference

The following arguments are supported:

- `auth_method` - (Required) The name of the ACL auth method this rule apply.
- `description` - (Optional) A free form human readable description of the binding rule.
- `selector` - (Optional) The expression used to math this rule against valid identities returned from an auth method validation.
- `bind_type` - (Required) Specifies the way the binding rule affects a token created at login.
- `bind_name` - (Required) The name to bind to a token at login-time.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the the binding rule.
- `auth_method` - The name of the ACL auth method this rule apply.

- `description` - A free form human readable description of the binding rule.
- `selector` - The expression used to math this rule against valid identities returned from an auth method validation.
- `bind_type` - Specifies the way the binding rule affects a token created at login.
- `bind_name` - The name to bind to a token at login-time.

consul_acl_policy

Starting with Consul 1.4.0, the consul_acl_policy can be used to managed Consul ACL policies.

Example Usage

```
resource "consul_acl_policy" "test" {
  name        = "my_policy"
  datacenters = ["dc1"]
  rules       = <<-RULE
    node_prefix "" {
      policy = "read"
    }
    RULE
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the policy.
- `description` - (Optional) The description of the policy.
- `rules` - (Required) The rules of the policy.
- `datacenters` - (Optional) The datacenters of the policy.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the policy.
- `name` - The name of the policy.
- `description` - The description of the policy.
- `rules` - The rules of the policy.
- `datacenters` - The datacenters of the policy.

Import

`consul_acl_policy` can be imported:

```
$ terraform import consul_acl_policy.my-policy 1c90ef03-a6dd-6a8c-ac49-042ad3752896
```

consul_acl_role

Starting with Consul 1.5.0, the consul_acl_role can be used to managed Consul ACL roles.

Example Usage

```
resource "consul_acl_policy" "read-policy" {
    name = "read-policy"
    rules = "node \"\" { policy = \"read\" }"
    datacenters = [ "dc1" ]
}

resource "consul_acl_role" "test" {
    name = "foo"
    description = "bar"

    policies = [
        "${consul_acl_policy.read-policy.id}"
    ]

    service_identities {
        service_name = "foo"
    }
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the ACL role.
- `description` - (Optional) A free form human readable description of the role.
- `policies` - (Optional) The list of policies that should be applied to the role.
- `service_identities` - (Optional) The list of service identities that should be applied to the role.

The `service_identities` supports:

- `service_name` - (Required) The name of the service.
- `datacenters` - (Optional) The datacenters the effective policy is valid within.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the role.
- `name` - The name of the ACL role.

- `description` - A free form human readable description of the role.
- `policies` - The list of policies that should be applied to the role.
- `service_identities` - The list of service identities that should be applied to the role.

consul_acl_token

The `consul_acl_token` resource writes an ACL token into Consul.

Example Usage

Basic usage

```
resource "consul_acl_policy" "agent" {
  name  = "agent"
  rules = <<-RULE
    node_prefix "" {
      policy = "read"
    }
    RULE
}

resource "consul_acl_token" "test" {
  description = "my test token"
  policies = ["${consul_acl_policy.agent.name}"]
  local = true
}
```

Set explictly the `accessor_id`

```
resource "random_uuid" "test" { }

resource "consul_acl_token" "test_predefined_id" {
  accessor_id = ${random_uuid.test_uuid.result}
  description = "my test uuid token"
  policies = ["${consul_acl_policy.agent.name}"]
  local = true
}
```

Argument Reference

The following arguments are supported:

- `accessor_id` - (Optional) The uuid of the token. If omitted, Consul will generate a random uuid.
- `description` - (Optional) The description of the token.
- `policies` - (Optional) The list of policies attached to the token.
- `local` - (Optional) The flag to set the token local to the current datacenter.

Attributes Reference

The following attributes are exported:

- `id` - The token accessor ID.
- `accessor_id` - The token accessor ID.
- `description` - The description of the token.
- `policies` - The list of policies attached to the token.
- `local` - The flag to set the token local to the current datacenter.

Import

`consul_acl_token` can be imported. This is especially useful to manage the anonymous and the master token with Terraform:

```
$ terraform import consul_acl_token.anonymous 00000000-0000-0000-0000-000000000002
$ terraform import consul_acl_token.master-token 624d94ca-bc5c-f960-4e83-0a609cf588be
```

consul_acl_token_policy_attachment

The `consul_acl_token_attachment` resource links a Consul Token and an ACL policy. The link is implemented through an update to the Consul ACL token.

NOTE: This resource is only useful to attach policies to an ACL token that has been created outside the current Terraform configuration, like the anonymous or the master token. If the token you need to attach a policy to has been created in the current Terraform configuration and will only be used in it, you should use the `policies` attribute of `consul_acl_token` (/docs/providers/consul/r/acl_token.html).

Example Usage

Attach a policy to the anonymous token

```
resource "consul_acl_policy" "agent" {
  name  = "agent"
  rules = <<-RULE
    node_prefix "" {
      policy = "read"
    }
    RULE
}

resource "consul_acl_token_policy_attachment" "attachment" {
    token_id = "00000000-0000-0000-0000-000000000002"
    policy   = "${consul_acl_policy.agent.name}"
}
```

Attach a policy to a token creted in another Terraform configuration

In `first_configuration/main.tf`

```
resource "consul_acl_token" "test" {
  accessor_id = "9b20de68-3ea2-4b70-b4f1-506afad062a4"
  description = "my test token"
  local = true

  lifecycle {
    ignore_changes = ["policies"]
  }
}
```

In `second_configuration/main.tf`

```
resource "consul_acl_policy" "agent" {
  name  = "agent"
  rules = <<-RULE
    node_prefix "" {
      policy = "read"
    }
    RULE
}

resource "consul_acl_token_policy_attachment" "attachment" {
    token_id = "9b20de68-3ea2-4b70-b4f1-506afad062a4"
    policy   = "${consul_acl_policy.agent.name}"
}
```

NOTE: consul_acl_token would attempt to enforce an empty set of policies, because its policies attribute is empty. For this reason it is necessary to add the lifecycle clause to prevent Terraform from attempting to empty the set of policies associated to the token.

Argument Reference

The following arguments are supported:

- `token_id` - (Required) The id of the token.
- `policy` - (Required) The name of the policy attached to the token.

Attributes Reference

The following attributes are exported:

- `id` - The attachment ID.
- `token_id` - The id of the token.
- `policy` - The name of the policy attached to the token.

Import

`consul_acl_token_policy_attachment` can be imported. This is especially useful to manage the policies attached to the anonymous and the master tokens with Terraform:

```
$ terraform import consul_acl_token_policy_attachment.anonymous 00000000-0000-0000-0000-000000000002:poli
cy_name
$ terraform import consul_acl_token_policy_attachment.master-token 624d94ca-bc5c-f960-4e83-0a609cf588be:p
olicy_name
```

consul_agent_service

The `consul_agent_service` resource has been deprecated in version 2.0.0 of the provider and will be removed in a future release. Please read the upgrade guide (/docs/providers/consul/upgrading.html#deprecation-of-consul_agent_service) for more information.

Provides access to the agent service data in Consul. This can be used to define a service associated with a particular agent. Currently, defining health checks for an agent service is not supported.

Example Usage

```
resource "consul_agent_service" "app" {
  address = "www.google.com"
  name    = "google"
  port    = 80
  tags    = ["tag0", "tag1"]
}
```

Argument Reference

The following arguments are supported:

- `address` - (Optional) The address of the service. Defaults to the address of the agent.
- `name` - (Required) The name of the service.
- `port` - (Optional) The port of the service.
- `tags` - (Optional) A list of values that are opaque to Consul, but can be used to distinguish between services or nodes.

Attributes Reference

The following attributes are exported:

- `address` - The address of the service.
- `id` - The ID of the service, defaults to the value of `name`.
- `name` - The name of the service.
- `port` - The port of the service.
- `tags` - The tags of the service.

consul_autopilot_config

Provides access to the Autopilot Configuration (https://www.consul.io/docs/guides/autopilot.html) of Consul to automatically manage Consul servers.

It includes to automatically cleanup dead servers, monitor the status of the Raft cluster and stable server introduction.

Example Usage

```
resource "consul_autopilot_config" "config" {
    cleanup_dead_servers      =  false
    last_contact_threshold    =  "1s"
    max_trailing_logs         =  500
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.
- `cleanup_dead_servers` - (Optional) Whether to remove failing servers when a replacement comes online. Defaults to true.
- `last_contact_threshold` - (Optional) The time after which a server is considered as unhealthy and will be removed. Defaults to `"200ms"`.
- `max_trailing_logs` - (Optional) The maximum number of Raft log entries a server can trail the leader. Defaults to 250.
- `server_stabilization_time` - (Optional) The period to wait for a server to be healthy and stable before being promoted to a full, voting member. Defaults to `"10s"`.
- `redundancy_zone_tag` - (Optional) The redundancy zone (https://www.consul.io/docs/guides/autopilot.html#redundancy-zones) tag to use. Consul will try to keep one voting server by zone to take advantage of isolated failure domains. Defaults to an empty string.
- `disable_upgrade_migration` - (Optional) Whether to disable upgrade migrations (https://www.consul.io/docs/guides/autopilot.html#redundancy-zones). Defaults to false.
- `upgrade_version_tag` - (Optional) The tag to override the version information used during a migration. Defaults to an empty string.

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter used.
- `cleanup_dead_servers` - Whether to remove failing servers.
- `last_contact_threshold` - The time after which a server is considered as unhealthy and will be removed.
- `max_trailing_logs` - The maximum number of Raft log entries a server can trail the leader.
- `server_stabilization_time` - The period to wait for a server to be healthy and stable before being promoted to a full, voting member.
- `redundancy_zone_tag` - The redundancy zone (https://www.consul.io/docs/guides/autopilot.html#redundancy-zones) tag used. Consul will try to keep one voting server by zone to take advantage of isolated failure domains.
- `disable_upgrade_migration` - Whether to disable upgrade migrations (https://www.consul.io/docs/guides/autopilot.html#redundancy-zones).
- `upgrade_version_tag` - The tag to override the version information used during a migration.

consul_catalog_entry

> The `consul_catalog_entry` resource has been deprecated in version 2.0.0 of the provider and will be removed in a future release. Please read the upgrade guide (/docs/providers/consul/upgrading.html#deprecation-of-consul_catalog_entry) for more information.

Registers a node or service with the Consul Catalog (https://www.consul.io/docs/agent/http/catalog.html#catalog_register). Currently, defining health checks is not supported.

Example Usage

```
resource "consul_catalog_entry" "app" {
  address = "192.168.10.10"
  node    = "foobar"

  service = {
    address = "127.0.0.1"
    id      = "redis1"
    name    = "redis"
    port    = 8000
    tags    = ["master", "v1"]
  }
}
```

Argument Reference

The following arguments are supported:

- `address` - (Required) The address of the node being added to, or referenced in the catalog.
- `node` - (Required) The name of the node being added to, or referenced in the catalog.
- `service` - (Optional) A service to optionally associated with the node. Supported values are documented below.
- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.
- `token` - (Optional) ACL token.

The `service` block supports the following:

- `address` - (Optional) The address of the service. Defaults to the node address.
- `id` - (Optional) The ID of the service. Defaults to the `name`.
- `name` - (Required) The name of the service
- `port` - (Optional) The port of the service.
- `tags` - (Optional) A list of values that are opaque to Consul, but can be used to distinguish between services or nodes.

Attributes Reference

The following attributes are exported:

- `address` - The address of the service.
- `node` - The ID of the service, defaults to the value of `name`.

consul_config_entry

The Configuration Entry (https://www.consul.io/docs/agent/config_entries.html) resource can be used to provide cluster-wide defaults for various aspects of Consul.

Example Usage

```
resource "consul_config_entry" "proxy_defaults" {
  kind = "proxy-defaults"
  # Note that only "global" is currently supported for proxy-defaults and that
  # Consul will override this attribute if you set it to anything else.
  name = "global"

  config_json = jsonencode({
    Config = {
      local_connect_timeout_ms = 1000
      handshake_timeout_ms     = 10000
    }
  })
}

resource "consul_config_entry" "web" {
  name = "web"
  kind = "service-defaults"

  config_json = jsonencode({
    Protocol    = "http"
  })
}

resource "consul_config_entry" "admin" {
  name = "admin"
  kind = "service-defaults"

  config_json = jsonencode({
    Protocol    = "http"
  })
}

resource "consul_config_entry" "service_resolver" {
  kind = "service-resolver"
  name = consul_config_entry.web.name

  config_json = jsonencode({
    DefaultSubset = "v1"

    Subsets = {
      "v1" = {
        Filter = "Service.Meta.version == v1"
      }
      "v2" = {
        Filter = "Service.Meta.version == v2"
      }
    }
  })
```

```
  })

  depends_on = [
    consul_config_entry.web,
    consul_config_entry.admin
  ]
}

resource "consul_config_entry" "service_splitter" {
  kind = "service-splitter"
  name = consul_config_entry.web.name

  config_json = jsonencode({
    Splits = [
      {
        Weight        = 90
        ServiceSubset = "v1"
      },
      {
        Weight        = 10
        ServiceSubset = "v2"
      },
    ]
  })

  depends_on = [consul_config_entry.service_resolver]
}

resource "consul_config_entry" "service_router" {
  kind = "service-router"
  name = "web"

  config_json = jsonencode({
    Routes = [
      {
        Match = {
          HTTP = {
            PathPrefix = "/admin"
          }
        }

        Destination = {
          Service = "admin"
        }
      },
      # NOTE: a default catch-all will send unmatched traffic to "web"
    ]
  })

}
```

Argument Reference

The following arguments are supported:

- `kind` - (Required) The kind of configuration entry to register.

- `name` - (Required) The name of the configuration entry being registred.
- `config_json` - (Optional) An arbitrary map of configuration values.

Attributes Reference

The following attributes are exported:

- `id` - The id of the configuration entry.
- `kind` - The kind of the configuration entry.
- `name` - The name of the configuration entry.
- `config_json` - A map of configuration values.

consul_intention

Intentions (https://www.consul.io/docs/connect/intentions.html) are used to define rules for which services may connect to one another when using Consul Connect (https://www.consul.io/docs/connect/index.html).

It is appropriate to either reference existing services or specify non-existent services that will be created in the future when creating intentions. This resource can be used in conjunction with the `consul_service` datasource when referencing services registered on nodes that have a running Consul agent.

Example Usage

Create a simplest intention with static service names:

```
resource "consul_intention" "database" {
  source_name      = "api"
  destination_name = "db"
  action           = "allow"
}
```

Referencing a known service via a datasource:

```
resource "consul_intention" "database" {
  source_name      = "api"
  destination_name = "${consul_service.pg.name}"
  action           = "allow"
}

data "consul_service" "pg" {
  name = "postgresql"
}
```

Argument Reference

The following arguments are supported:

- `source_name` - (Required, string) The name of the source service for the intention. This service does not have to exist.
- `destination_name` - (Required, string) The name of the destination service for the intention. This service does not have to exist.
- `action` - (Required, string) The intention action. Must be one of `allow` or `deny`.
- `meta` - (Optional, map) Key/value pairs that are opaque to Consul and are associated with the intention.
- `description` - (Optional, string) Optional description that can be used by Consul tooling, but is not used internally.
- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the intention.
- `source_name` - The source for the intention.
- `destination_name` - The destination for the intention.
- `description` - A description of the intention.
- `meta` - Key/value pairs associated with the intention.

consul_key_prefix

Allows Terraform to manage a "namespace" of Consul keys that share a common name prefix.

Like `consul_keys`, this resource can write values into the Consul key/value store, but *unlike* `consul_keys` this resource can detect and remove extra keys that have been added some other way, thus ensuring that rogue data added outside of Terraform will be removed on the next run.

This resource is thus useful in the case where Terraform is exclusively managing a set of related keys.

To avoid accidentally clobbering matching data that existed in Consul before a `consul_key_prefix` resource was created, creation of a key prefix instance will fail if any matching keys are already present in the key/value store. If any conflicting data is present, you must first delete it manually or explicitly import the prefix.

Warning After this resource is instantiated, Terraform takes control over *all* keys with the given path prefix, and will remove any matching keys that are not present in the configuration. It will also delete *all* keys under the given prefix when a `consul_key_prefix` resource is destroyed, even if those keys were created outside of Terraform.

Example Usage

```
resource "consul_key_prefix" "myapp_config" {
  datacenter = "nyc1"
  token      = "abcd"

  # Prefix to add to prepend to all of the subkey names below.
  path_prefix = "myapp/config/"

  subkeys = {
    "elb_cname"         = "${aws_elb.app.dns_name}"
    "s3_bucket_name"    = "${aws_s3_bucket.app.bucket}"
    "database/hostname" = "${aws_db_instance.app.address}"
    "database/port"     = "${aws_db_instance.app.port}"
    "database/username" = "${aws_db_instance.app.username}"
    "database/name"     = "${aws_db_instance.app.name}"
  }

  subkey {
    path  = "database/password"
    value = "${aws_db_instance.app.password}"
    flags = 2
  }

}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.
- `token` - (Optional) The ACL token to use. This overrides the token that the agent provides by default.
- `path_prefix` - (Required) Specifies the common prefix shared by all keys that will be managed by this resource instance. In most cases this will end with a slash, to manage a "folder" of keys.
- `subkeys` - (Optional) A mapping from subkey name (which will be appended to the given `path_prefix`) to the value that should be stored at that key. Use slashes, as shown in the above example, to create "sub-folders" under the given path prefix.
- `subkey` - (Optional) A subkey to add. Supported values documented below. Multiple blocks supported.

The `subkey` block supports the following:

- `path` - (Required) This is the path (which will be appended to the given `path_prefix`) in Consul that should be written to.
- `value` - (Required) The value to write to the given path.
- `flags` - (Optional) An unsigned integer value (https://www.consul.io/api/kv.html#flags-1) to attach to the key (defaults to 0).

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the keys are being read/written to.

Import

`consul_key_prefix` can be imported. This is useful when the path already and you know all keys in path must be removed.

```
$ terraform import consul_key_prefix.myapp_config myapp/config/
```

consul_keys

The `consul_keys` resource writes sets of individual values into Consul. This is a powerful way to expose infrastructure details to clients.

This resource manages individual keys, and thus it can create, update and delete the keys explicitly given. However, it is not able to detect and remove additional keys that have been added by non-Terraform means. To manage *all* keys sharing a common prefix, and thus have Terraform remove errant keys not present in the configuration, consider using the `consul_key_prefix` resource instead.

Example Usage

```
resource "consul_keys" "app" {
  datacenter = "nyc1"
  token      = "abcd"

  # Set the CNAME of our load balancer as a key
  key {
    path  = "service/app/elb_address"
    value = "${aws_elb.app.dns_name}"
  }
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.
- `token` - (Optional) The ACL token to use. This overrides the token that the agent provides by default.
- `key` - (Required) Specifies a key in Consul to be written. Supported values documented below.

The `key` block supports the following:

- `path` - (Required) This is the path in Consul that should be written to.
- `value` - (Required) The value to write to the given path.
- `flags` - (Optional) An unsigned integer value (https://www.consul.io/api/kv.html#flags-1) to attach to the key (defaults to 0).
- `delete` - (Optional) If true, then the key will be deleted when either its configuration block is removed from the configuration or the entire resource is destroyed. Otherwise, it will be left in Consul. Defaults to false.

Deprecated `key` arguments

Prior to Terraform 0.7, this resource was used both to read *and* write the Consul key/value store. The read functionality has moved to the `consul_keys` *data source*, whose documentation can be found via the navigation.

The pre-0.7 interface for reading keys is still supported for backward compatibility, but will be removed in a future version of Terraform.

Attributes Reference

The following attributes are exported:

- `datacenter` - The datacenter the keys are being written to.

consul_node

Provides access to Node data in Consul. This can be used to define a node. Currently, defining health checks is not supported.

Example Usage

```
resource "consul_node" "foobar" {
  address = "192.168.10.10"
  name    = "foobar"
}
```

Argument Reference

The following arguments are supported:

- `address` - (Required) The address of the node being added to, or referenced in the catalog.
- `name` - (Required) The name of the node being added to, or referenced in the catalog.
- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.
- `meta` - (Optional, map) Key/value pairs that are associated with the node.

Attributes Reference

The following attributes are exported:

- `address` - The address of the service.
- `name` - The name of the service.
- `meta` - (Optional, map) Key/value pairs that are associated with the node.

consul_prepared_query

Allows Terraform to manage a Consul prepared query.

Managing prepared queries is done using Consul's REST API. This resource is useful to provide a consistent and declarative way of managing prepared queries in your Consul cluster using Terraform.

Example Usage

```
# Creates a prepared query myquery.query.consul that finds the nearest
# healthy myapp.service.consul instance that has the active tag and not
# the standby tag.
resource "consul_prepared_query" "myapp-query" {
  name         = "myquery"
  datacenter   = "us-central1"
  token        = "abcd"
  stored_token = "wxyz"
  only_passing = true
  near         = "_agent"

  service = "myapp"
  tags    = ["active", "!standby"]

  failover {
    nearest_n   = 3
    datacenters = ["us-west1", "us-east-2", "asia-east1"]
  }

  dns {
    ttl = "30s"
  }
}

# Creates a Prepared Query Template that matches *-near-self.query.consul
# and finds the nearest service that matches the glob character (e.g.
# foo-near-self.query.consul will find the nearest healthy foo.service.consul).
resource "consul_prepared_query" "service-near-self" {
  datacenter   = "nyc1"
  token        = "abcd"
  stored_token = "wxyz"
  name         = ""
  only_passing = true
  connect      = true
  near         = "_agent"

  template {
    type   = "name_prefix_match"
    regexp = "^(.*)-near-self$"
  }

  service = "$${match(1)}"

  failover {
    nearest_n   = 3
    datacenters = ["dc2", "dc3", "dc4"]
  }

  dns {
    ttl = "5m"
  }
}
```

Argument Reference

The following arguments are supported:

- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.
- `token` - (Optional) The ACL token to use when saving the prepared query. This overrides the token that the agent provides by default.
- `stored_token` - (Optional) The ACL token to store with the prepared query. This token will be used by default whenever the query is executed.
- `name` - (Required) The name of the prepared query. Used to identify the prepared query during requests. Can be specified as an empty string to configure the query as a catch-all.
- `service` - (Required) The name of the service to query.
- `session` - (Optional) The name of the Consul session to tie this query's lifetime to. This is an advanced parameter that should not be used without a complete understanding of Consul sessions and the implications of their use (it is recommended to leave this blank in nearly all cases). If this parameter is omitted the query will not expire.
- `tags` - (Optional) The list of required and/or disallowed tags. If a tag is in this list it must be present. If the tag is preceded with a "!" then it is disallowed.
- `only_passing` - (Optional) When `true`, the prepared query will only return nodes with passing health checks in the result.
- `connect` - (Optional) When `true` the prepared query will return connect proxy services for a queried service. Conditions such as `tags` in the prepared query will be matched against the proxy service. Defaults to false.
- `near` - (Optional) Allows specifying the name of a node to sort results near using Consul's distance sorting and network coordinates. The magic `_agent` value can be used to always sort nearest the node servicing the request.
- `failover` - (Optional) Options for controlling behavior when no healthy nodes are available in the local DC.
  - `nearest_n` - (Optional) Return results from this many datacenters, sorted in ascending order of estimated RTT.
  - `datacenters` - (Optional) Remote datacenters to return results from.
- `dns` - (Optional) Settings for controlling the DNS response details.
  - `ttl` - (Optional) The TTL to send when returning DNS results.
- `template` - (Optional) Query templating options. This is used to make a single prepared query respond to many different requests.
  - `type` - (Required) The type of template matching to perform. Currently only `name_prefix_match` is supported.
  - `regexp` - (Required) The regular expression to match with. When using `name_prefix_match`, this regex is applied against the query name.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the prepared query, generated by Consul.

Import

`consul_prepared_query` can be imported with the query's ID in the Consul HTTP API.

```
$ terraform import consul_prepared_query.my_service 71ecfb82-717a-4258-b4b6-2fb75144d856
```

consul_service

A high-level resource for creating a Service in Consul in the Consul catalog. This is appropriate for registering external services (https://www.consul.io/docs/guides/external.html) and can be used to create services addressable by Consul that cannot be registered with a local agent (https://www.consul.io/docs/agent/basics.html).

If the Consul agent is running on the node where this service is registered, it is not recommended to use this resource.

Example Usage

Creating a new node with the service:

```
resource "consul_service" "google" {
  name    = "google"
  node    = "${consul_node.compute.name}"
  port    = 80
  tags    = ["tag0"]
}

resource "consul_node" "compute" {
  name    = "compute-google"
  address = "www.google.com"
}
```

Utilizing an existing known node:

```
resource "consul_service" "google" {
  name    = "google"
  node    = "google"
  port    = 443
}
```

Register an health-check:

```
resource "consul_service" "redis" {
  name = "redis"
  node = "redis"
  port = 6379

  check {
    check_id                          = "service:redis1"
    name                              = "Redis health check"
    status                            = "passing"
    http                              = "https://www.hashicorptest.com"
    tls_skip_verify                   = false
    method                            = "PUT"
    interval                          = "5s"
    timeout                           = "1s"
    deregister_critical_service_after = "30s"

    header {
      name  = "foo"
      value = ["test"]
    }

    header {
      name  = "bar"
      value = ["test"]
    }
  }
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required, string) The name of the service.
- `node` - (Required, string) The name of the node the to register the service on.
- `address` - (Optional, string) The address of the service. Defaults to the address of the node.
- `service_id` (Optional, string) - If the service ID is not provided, it will be defaulted to the value of the `name` attribute.
- `port` - (Optional, int) The port of the service.
- `checks` - (Optional, list of checks) Health-checks to register to monitor the service. The list of attributes for each health-check is detailled below.
- `tags` - (Optional, set of strings) A list of values that are opaque to Consul, but can be used to distinguish between services or nodes.
- `datacenter` - (Optional) The datacenter to use. This overrides the agent's default datacenter and the datacenter in the provider setup.
- `meta` - (Optional) A map of arbitrary KV metadata linked to the service instance.

The following attributes are available for each health-check:

- `check_id` - (Optional, string) An ID, *unique per agent*. Will default to *name* if not set.
- `name` - (Required) The name of the health-check.
- `notes` - (Optional, string) An opaque field meant to hold human readable text.
- `status` - (Optional, string) The initial health-check status.
- `tcp` - (Optional, string) The TCP address and port to connect to for a TCP check.
- `http` - (Optional, string) The HTTP endpoint to call for an HTTP check.
- `header` - (Optional, set of headers) The headers to send for an HTTP check. The attributes of each header is given below.
- `tls_skip_verify` - (Optional, boolean) Whether to deactivate certificate verification for HTTP health-checks. Defaults to `false`.
- `method` - (Optional, string) The method to use for HTTP health-checks. Defaults to `GET`.
- `interval` - (Required, string) The interval to wait between each health-check invocation.
- `timeout` - (Required, string) The timeout value for HTTP checks.
- `deregister_critical_service_after` - (Optional, string) The time after which the service is automatically deregistered when in the `critical` state. Defaults to `30s`.

Each `header` must have the following attributes: * `name` - (Required, string) The name of the header. * `value` - (Required, list of strings) The header's list of values.

Attributes Reference

The following attributes are exported:

- `service_id` - The ID of the service.
- `address` - The address of the service.
- `node` - The node the service is registered on.
- `name` - The name of the service.
- `port` - The port of the service.
- `tags` - The tags of the service.
- `checks` - The list of health-checks associated with the service.
- `datacenter` - The datacenter of the service.
- `meta` - A map of arbitrary KV metadata linked to the service instance.

Upgrading the Consul Terraform Provider

This page includes details on our compatibility promise and guidelines to follow when upgrading between versions of the provider. Whenever possible, we recommend verifying upgrades in isolated test environments.

Upgrading to 2.4.0

Changes to consul_service

The `external` attribute introduced in 2.3.0 has been deprecated and does not update the associated Consul Node meta information anymore. The same functionality can be done by setting the meta attribute on the `consul_node` resource:

```
# This was working in 2.3.0
resource "consul_node" "compute" {
    name    = "compute-example"
    address = "www.hashicorptest.com"
}

resource "consul_service" "example1" {
    name = "example"
    node = "${consul_node.compute.name}"
    port = 80

    external = true
}

# This is working in 2.4.0
resource "consul_node" "compute" {
    name    = "compute-example"
    address = "www.hashicorptest.com"

  meta = {
    "external-node" = "true"
    "external-probe" = "true"
  }
}

resource "consul_service" "example1" {
    name = "example"
    node = "${consul_node.compute.name}"
    port = 80
}
```

Upgrading to 2.0.0

There were several major deprecation notices introduced in 2.0.0. This reviews the details of each and provides migration instructions to the appropriate resources.

Deprecation of consul_agent_self

The `consul_agent_self` data source will be removed in the next major version of the provider. As a result, we recommend moving to the new `consul_agent_config` (/docs/providers/consul/d/agent_config.html) data source.

The `consul_agent_config` resource returns far less attributes, so as a result it may not provide all necessary functionality. Consul does still provide this data via API but promises no compatibility across versions (https://www.consul.io/docs/upgrade-specific.html#config-section-of-agent-self-endpoint-has-changed), therefore it is being removed from this provider.

Deprecation of consul_agent_service

The `consul_agent_service` resource will be removed in the next major version of the provider. As a result, we recommend moving to the `consul_service` (/docs/providers/consul/r/service.html) resource.

This resource has been updated to use the correct catalog APIs in place of service registration APIs. The `consul_agent_service` resource previously also used the service registration API designed for registration against an agent running on a local node. Because Terraform is intended to be run externally to the cluster, and for other internal reasons, this API was the incorrect one to use.

View migration instructions here (/docs/providers/consul/upgrading.html#migrating-to-consul_service-or-consul_node-resources).

Migrating to consul_service or consul_node resources

Migration to the `consul_service` resources are possible in two ways. Both require the configuration to be modified.

From `consul_agent_service` to `consul_service`:

1. Rename `consul_agent_service` resources to `consul_service` in the Terraform configuration files.
2. Add the `node` attribute where the service is currently registered, retrievable by querying the catalog (https://www.consul.io/api/catalog.html#list-nodes-for-service) or using the UI. This new attribute is required.
3. For a small number of resources, the first class `state rm` (https://www.terraform.io/docs/commands/state/rm.html) and `import` (https://www.terraform.io/docs/import/usage.html) commands can be used to first remove the old resource from the state, and then import it under the new resource name.
4. For a large number of resources, edit the state file directly to rename every resource at the same time (replace all instances of `consul_agent_service` with `consul_service`). This requires understanding the consequences and guidelines for editing state files (https://www.terraform.io/docs/backends/state.html#manual-state-pull-push), so please read those.

After following these steps, `terraform plan` should show no changes.

From `consul_catalog_entry` to `consul_service` or `consul_node`:

1. Copy the attributes from the `service {}` or `node {}` blocks into new `consul_service` or `consul_node` resources in the Terraform configuration files.
2. For a small number of resources, the first class `state rm` (https://www.terraform.io/docs/commands/state/rm.html) and `import` (https://www.terraform.io/docs/import/usage.html) commands can be used to first remove the old

resource from the state, and then import it under the new resource name. The `node` attribute will need to be added to services.

Modifications to consul_service

The `consul_service` resource has been modified to use catalog APIs in place of service registration APIs for creating services in the Consul catalog. This should be a functionally compatible change, and create or read services as prior. It now replaces `consul_catalog_entry` (the `service {}` block) and `consul_agent_service`. The `node` attribute is now required and the node must exist.

Deprecation of consul_catalog_entry

The `consul_catalog_entry` resource will be removed in the next major version of the provider. As a result, we recommend moving to the `consul_service` (/docs/providers/consul/r/service.html) or `consul_node` (/docs/providers/consul/r/node.html) resources.

These resources have been updated (or created) to use the correct catalog APIs as with `consul_catalog_entry`, but provide a first-class resource name.

View migration instructions here (/docs/providers/consul/upgrading.html#migrating-to-consul_service-or-consul_node-resources).

Renaming of Catalog Data Sources

`consul_catalog_nodes`, `consul_catalog_services`, and `consul_catalog_service` have been renamed to `consul_nodes`, `consul_services`, and `consul_service` respectively. The prior naming will continue to work, but in the long term it may be deprecated and removed. This is to present a more consistent and intuitive naming convention for the resources.