**https://www.consul.io/docs/**

**Service segmentation made easy**

**服务拆分**

Secure service-to-service communication with automatic TLS encryption and identity-based authorization

使用TLS加密和基于认证方式的授权来加密服务间通讯

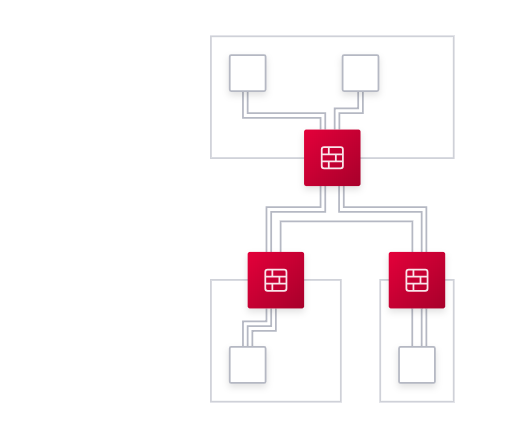
### The Challenge

Securing service-to-service communication with firewalls doesn’t scale in dynamic settings.

有防火墙的情况下保密服务间通讯没法覆盖动态设置

East-west firewalls use IP-based rules to secure ingress and egress traffic. But in a dynamic world where services move across machines and machines are frequently created and destroyed, this perimeter-based approach is difficult to scale as it results in complex network topologies and a sprawl of short-lived firewall rules

横跨东西方的防火墙使用基于IP的规则来保证出口和入口通讯流量。但是动态的世界中，服务会在不同主机之间频繁创建和销毁，这个基于边界的限定方法就难以完全覆盖了，原因就是复杂的网络拓扑结构和杂乱无序而又短命的防火墙规则



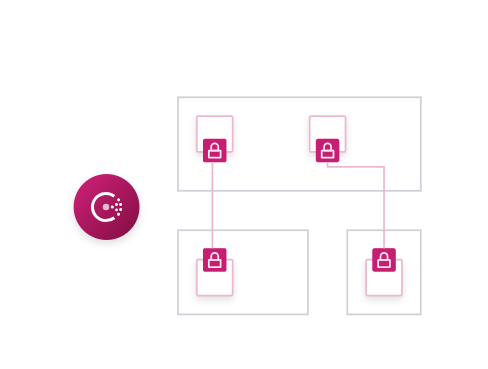
### The Solution

Service segmentation for dynamic service authorization.

为动态服务授权进行服务分割

Service segmentation is a new approach to secure the service itself rather than relying on the network. Consul uses service policies to codify which services are allowed to communicate. These policies scale across datacenters and large fleets without IP-based rules or networking middleware.

服务分割是保护服务本身而不是依赖于网络的新方案，Consul使用服务策略来编制服务通讯规则。这些策略无需基于ip的规则和网络中间件就能跨数据中心和集群的进行覆盖



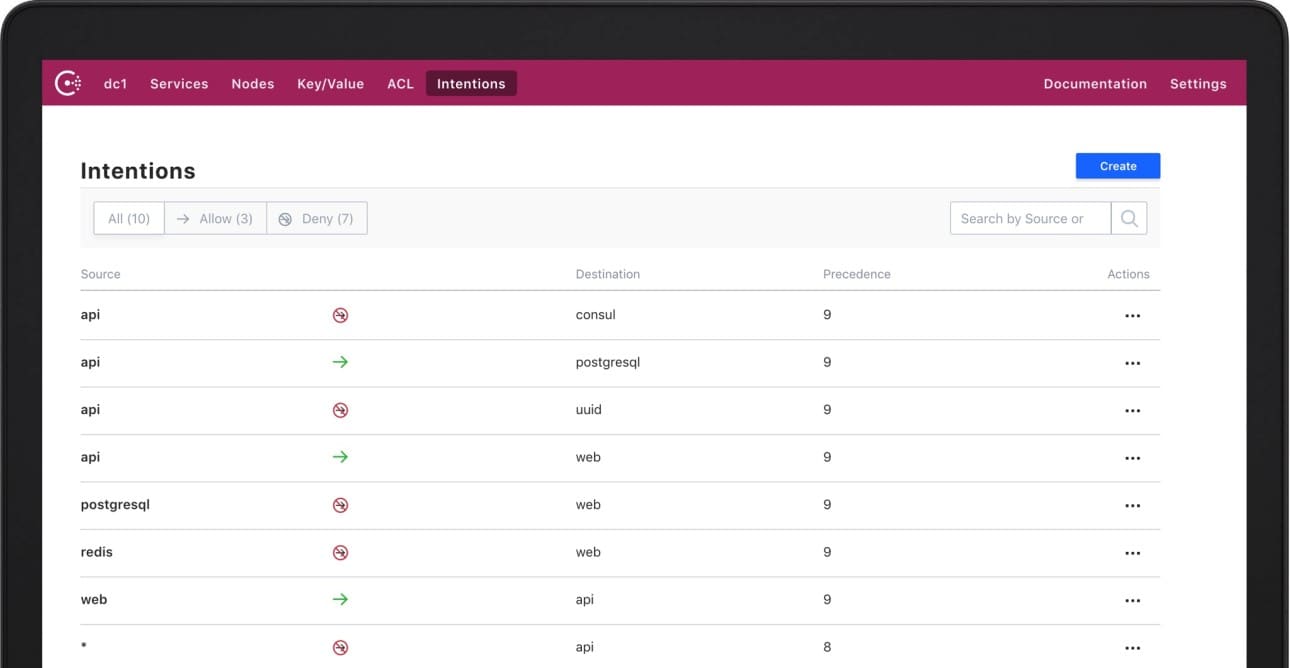
## Features

### Service Access Graph

服务访问权限图

Define and enforce service to service communication with a simple Intentions configuration. Service based rules, instead of IP-based rules, make it easy to manage dynamic infrastructure with frequently changing machines and service locations.

[Learn more](https://www.consul.io/docs/connect/intentions.html)

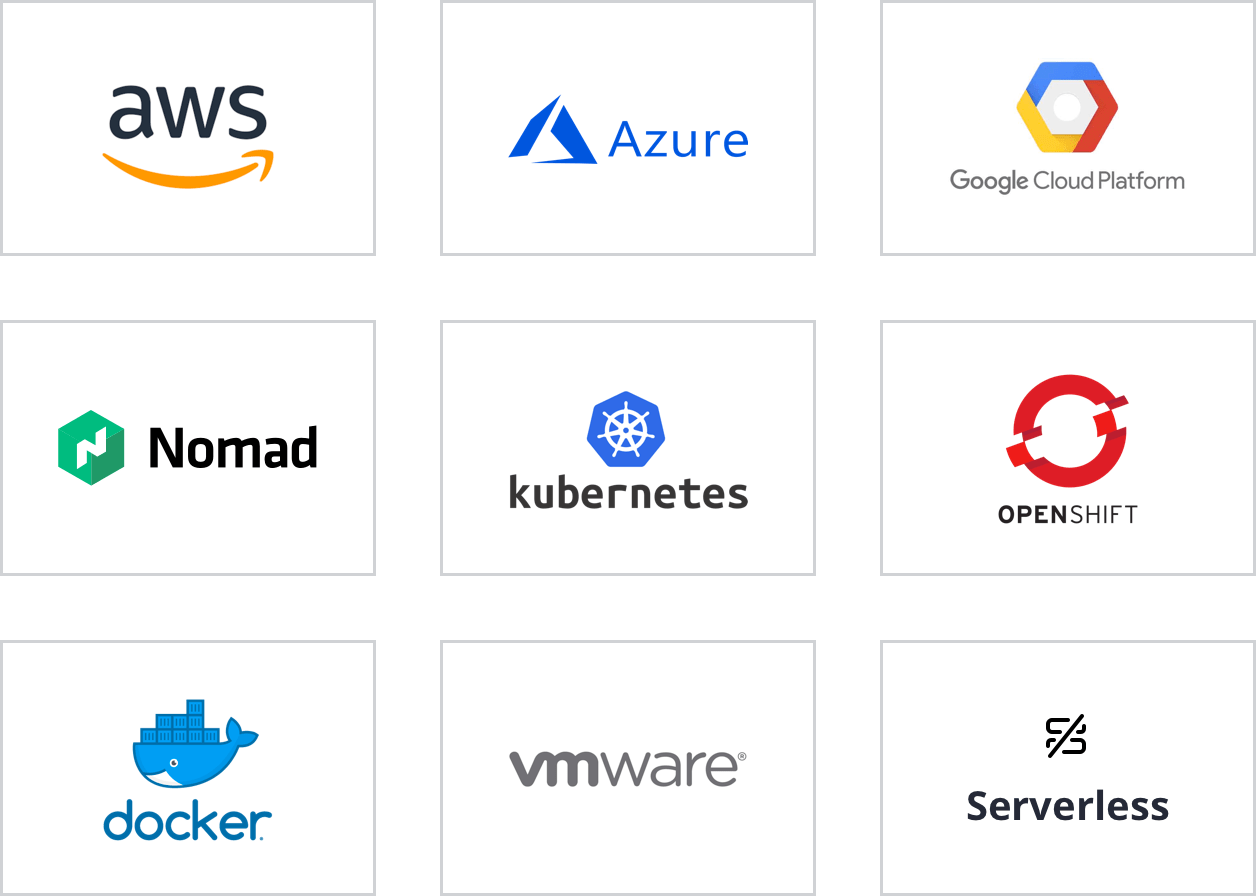


### Secure services across any runtime platform

跨运行环境保护服务

Secure communication between legacy and modern workloads. Sidecar proxies allow applications to be integrated without code changes and Layer 4 support provides nearly universal protocol compatibility.

[Learn more](https://www.consul.io/docs/connect/proxies.html)

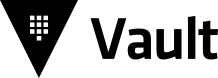


### Certificate-Based Service Identity

基于证书的服务认证

TLS certificates are used to identify services and secure communications. Certificates use the SPIFFE format for interoperability with other platforms. Consul can be a certificate authority to simplify deployment, or integrate with external signing authorities like Vault.

[Learn more](https://www.consul.io/docs/connect/ca.html)



### Encrypted communication

加密通讯

All traffic between services is encrypted and authenticated with mutual TLS. Using TLS provides a strong guarantee of the identity of services communicating, and ensures all data in transit is encrypted.

[Learn more](https://www.consul.io/docs/connect/security.html)

$ consul connect proxy -service web \

-service-addr 127.0.0.1:8000

-listen 10.0.1.109:7200

==> Consul Connect proxy starting...

Configuration mode: Flags

Service: web

Public listener: 10.0.1.109:7200 => 127.0.0.1:8000

...

$ tshark -V \

-Y "ssl.handshake.certificate" \

-O "ssl" \

-f "dst port 7200"

Frame 39: 899 bytes on wire (7192 bits), 899 bytes captured (7192 bits) on interface 0

Internet Protocol Version 4, Src: 10.0.1.110, Dst: 10.0.1.109

Transmission Control Protocol, Src Port: 61918, Dst Port: 7200, Seq: 136, Ack: 916, Len: 843

Secure Sockets Layer

TLSv1.2 Record Layer: Handshake Protocol: Certificate

Version: TLS 1.2 (0x0303)

Handshake Protocol: Certificate

RDNSequence item: 1 item (id-at-commonName=Consul CA 7)

RelativeDistinguishedName item (id-at-commonName=Consul CA 7)

Id: 2.5.4.3 (id-at-commonName)

DirectoryString: printableString (1)

printableString: Consul CA 7

# Service configuration made easy

# 服务配置更容易

Feature rich key/value store to easily configure services、

提供富键值对存储简化服务配置

### The Challenge

Runtime configuration management loses performance at scale.

大规模背景下运行时服务配置管理损失性能

Services have many runtime configurations, such as feature flags or maintenance modes, that need to be propagated in real time. Distributing these updates using configuration management or by re-deploying services can take minutes to hours. During these rollout periods, infrastructure can be out of sync and service configurations could be incorrect.

### The Solution

Real-time runtime configuration for distributed applications.

Consul can update service configurations across thousands of services in a globally distributed fleet in real-time. Configuration is stored in a hierarchical key/value store, and efficient edge triggers push changes out to applications quickly.

## Features

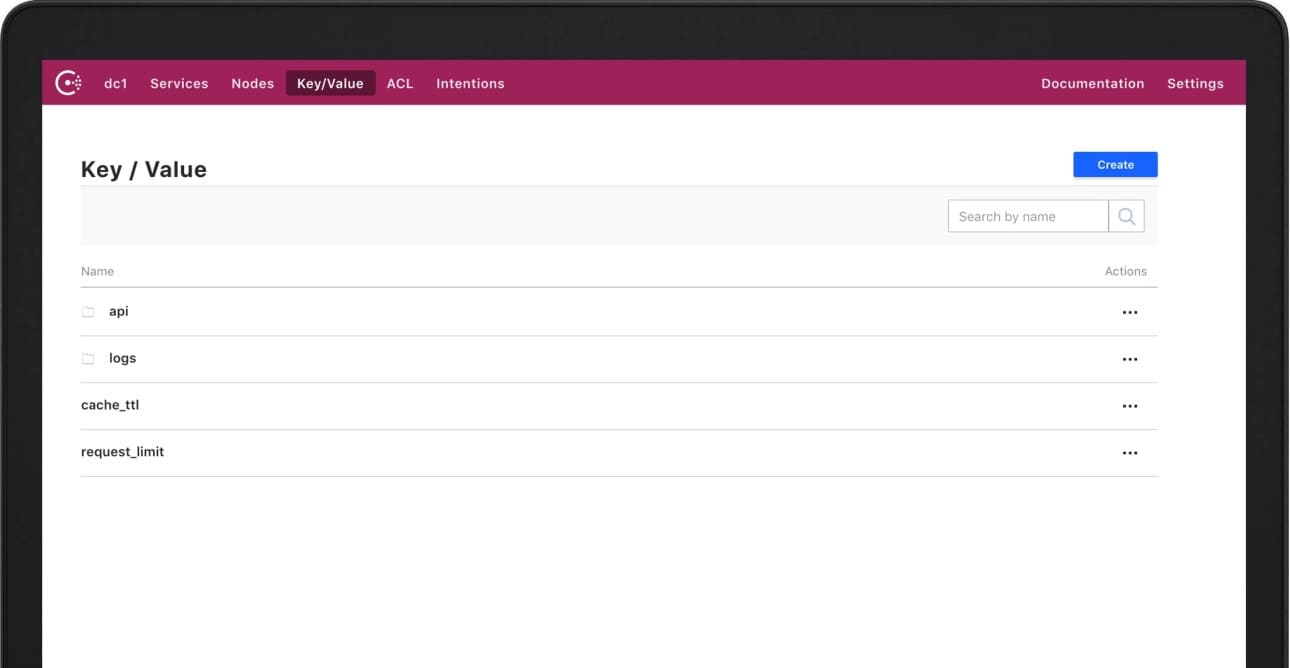
### Key/Value Store

键值对存储

Feature rich key/value store for dynamic service configuration data. Use it for feature flagging, maintenance modes, and more.

为动态服务配置数据提供富键值对存储，

[Learn more](https://learn.hashicorp.com/consul/getting-started/kv)



### Transaction Support

事务支持

The key/value store supports both read and write transactions. This allows multiple keys to be updated or read as an atomic transaction. Changes to service configuration can be done atomically to minimize churn and avoid inconsistencies.

[Learn more](https://www.consul.io/api/kv.html)

$ curl http://localhost:8500/v1/txn \

--request PUT \

--data \

'[

{

"KV": {

"Verb": "set",

"Key": "lock",

"Value": "MQ=="

}

},

{

"KV": {

"Verb": "cas",

"Index": 10,

"Key": "configuration",

"Value": "c29tZS1jb25maWc="

}

}

]'

### Blocking Queries / Edge-Triggered Requests

The Consul API supports blocking queries, allowing edge triggered updates. Clients use this to get notified immediately of any changes. Tools like consul-template allow configuration files to be rendered in real-time to third-party sources when any configuration changes are made.

[Learn more](https://www.consul.io/api/index.html#blocking-queries)

$ curl http://localhost:8500/v1/kv/web/config/rate\_limit?wait=1m&index=229

[

{

"LockIndex": 0,

"Key": "web/config/rate\_limit",

"Flags": 0,

"Value": "NjAw",

"CreateIndex": 229,

"ModifyIndex": 234

}

]

### Watches

监控

Watches use blocking queries to monitor for any configuration or health status updates and invoke user specified scripts to handle changes. This makes it easy to build reactive infrastructure.

[Learn more](https://www.consul.io/docs/agent/watches.html)

$ consul watch \

-type=key \

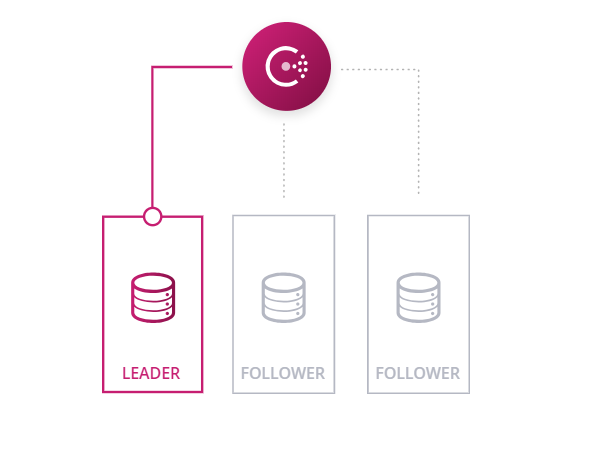
-key=web/config/rate\_limit \

/usr/local/bin/record-rate-limit.sh

### Distributed Locks and Semaphores

分布式锁和信号

The key/value store supports distributed locks and semaphores. This makes it easier for applications to perform leader election or manage access to shared resources.



# Service discovery made easy

Service registry, integrated health checks, and DNS and HTTP interfaces enable any service to discover and be discovered by other services

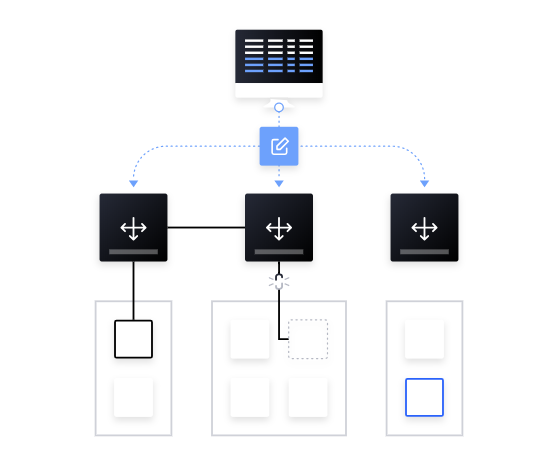
让服务发现更容易，服务注册，集成健康检查，通过DNS和http接口使得任何服务都能被其他服务发现

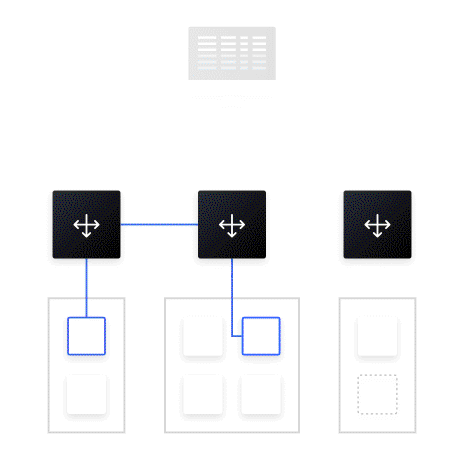
### The Challenge

Service load balancers aren't efficient in a dynamic world.

在动态世界中负载均衡效率低

Load balancers are often used to front a service tier and provide a static IP. These load balancers add cost, increase latency, introduce single points of failure, and must be updated as services scale up/down.





### The Solution

Service discovery for dynamic infrastructure.

动态架构下的服务发现

Instead of load balancers, connectivity in dynamic infrastructure is best solved with service discovery. Service discovery uses a registry to keep a real-time list of services, their location, and their health. Services query the registry to discover the location of upstream services and then connect directly. This allows services to scale up/down and gracefully handle failure without a load balancer intermediary.

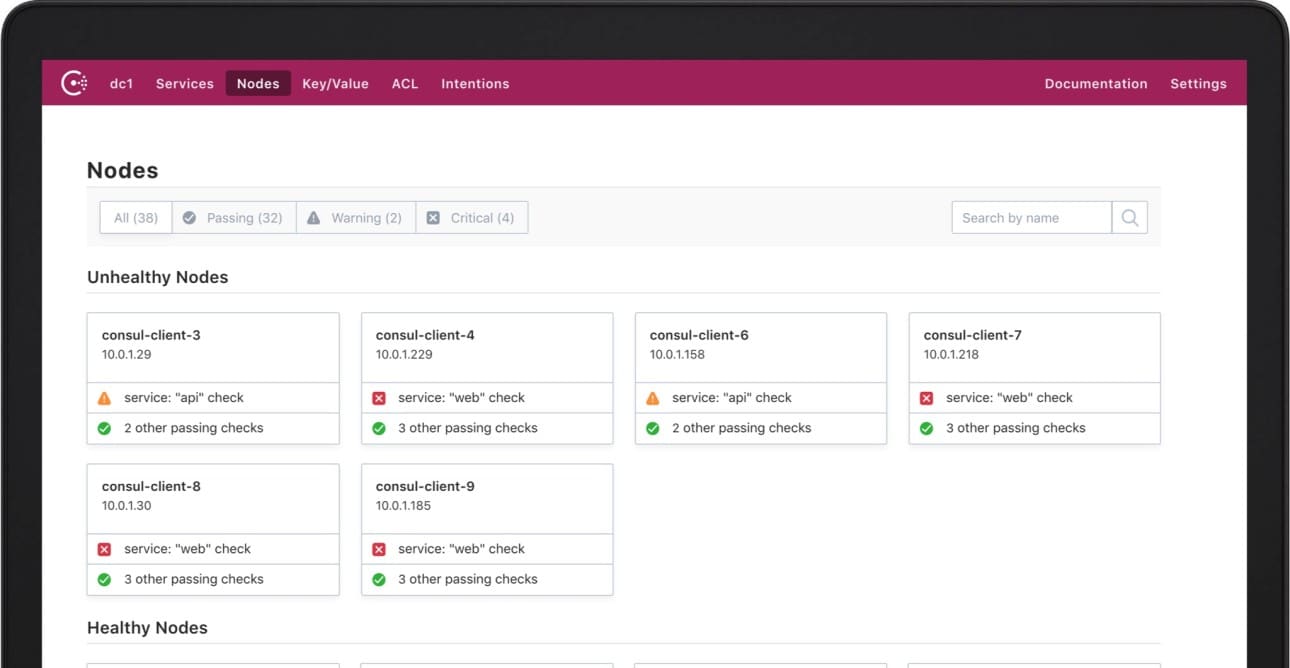
## Features

### Service Registry

服务注册

Consul provides a registry of all the running nodes and services, along with their current health status. This allows operators to understand the environment, and applications and automation tools to interact with dynamic infrastructure using an HTTP API.

[Learn more](https://learn.hashicorp.com/consul/getting-started/services)



### DNS Query Interface

Consul enables service discovery using a built-in DNS server. This allows existing applications to easily integrate, as almost all applications support using DNS to resolve IP addresses. Using DNS instead of a static IP address allows services to scale up/down and route around failures easily.

[Learn more](https://learn.hashicorp.com/consul/getting-started/services#querying-services)

$ dig web-frontend.service.consul. ANY

; <<>> DiG 9.8.3-P1 <<>> web-frontend.service.consul. ANY

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29981

;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:

;web-frontend.service.consul. IN ANY

;; ANSWER SECTION:

web-frontend.service.consul. 0 IN A 10.0.3.83

web-frontend.service.consul. 0 IN A 10.0.1.109

### HTTP API with Edge Triggers

通过边缘触发基于HTTP协议的api

Consul provides an HTTP API to query the service registry for nodes, services, and health check information. The API also supports blocking queries, or long-polling for any changes. This allows automation tools to react to （services being registered ）or （health status changes to change configurations ）or（ traffic routing ）in real time.

Consul提供了基于http的api来查询服务注册的节点，服务和健康检查信息。

Api还支持独占查询、长连接查询更改。这个功能为自动化工具对如下情况实时的作出反应提供了技术基础：服务注册、更改配置导致的健康状态变化、流量路由

[Learn more](https://learn.hashicorp.com/consul/getting-started/services#http-api)

$ curl http://localhost:8500/v1/health/service/web?index=11&wait=30s

{

...

"Node": "10-0-1-109",

"CheckID": "service:web",

"Name": "Service 'web' check",

"Status": "critical",

"ServiceID": "web",

"ServiceName": "web",

"CreateIndex": 10,

"ModifyIndex": 20

...

}

### Multi Datacenter

多数据中心

Consul supports multiple datacenters out of the box with no complicated configuration. Look up services in other datacenters or keep the request local. Advanced features like Prepared Queries enable automatic failover to other datacenters.

Consul不需要过多的配置开箱就能支持多数据中心。去别的数据中心查找服务或者将请求保留在本地。还有高级功能，比如预准备查询启用自动故障转移到其他部署的数据中心

[Learn more](https://www.consul.io/docs/guides/datacenters.html)

$ curl http://localhost:8500/v1/catalog/datacenters

[

"dc1",

"dc2"

]

$ curl http://localhost:8500/v1/catalog/nodes?dc=dc2

[

{

"ID": "7081dcdf-fdc0-0432-f2e8-a357d36084e1",

"Node": "10-0-1-109",

"Address": "10.0.1.109",

"Datacenter": "dc2",

"TaggedAddresses": {

"lan": "10.0.1.109",

"wan": "10.0.1.109"

},

"CreateIndex": 112,

"ModifyIndex": 125

},

...

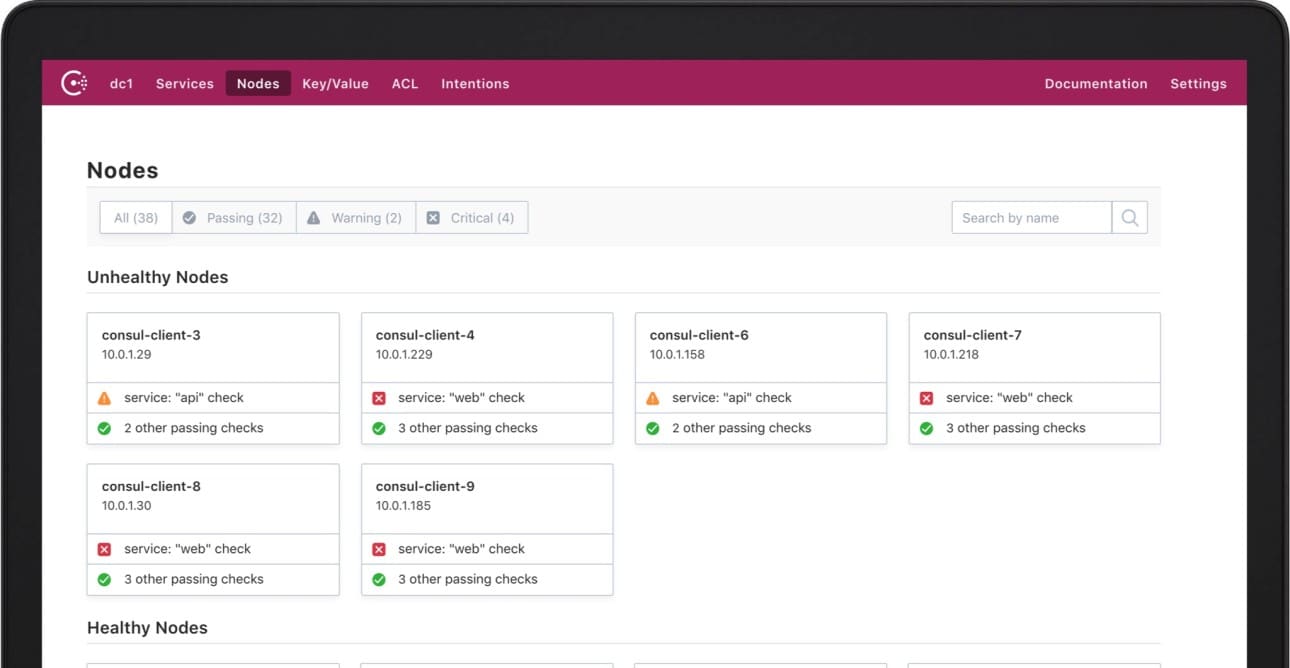
### Health Checks

健康检查

Pairing service discovery with health checking prevents routing requests to unhealthy hosts and enables services to easily provide circuit breakers.

使用健康检查进行服务发现配对能阻止请求被路由到失效主机，同时使服务很容易就能启用断路熔断器

[Learn more](https://learn.hashicorp.com/consul/getting-started/checks)



# Install Consul

Installing Consul is simple. There are two approaches to installing Consul:

1. Using a [precompiled binary](https://www.consul.io/docs/install/index.html#precompiled-binaries)
2. Installing [from source](https://www.consul.io/docs/install/index.html#compiling-from-source)

Downloading a precompiled binary is easiest, and we provide downloads over TLS along with SHA256 sums to verify the binary. We also distribute a PGP signature with the SHA256 sums that can be verified.

## [»](https://www.consul.io/docs/install/index.html" \l "precompiled-binaries)Precompiled Binaries

To install the precompiled binary, [download](https://www.consul.io/downloads.html) the appropriate package for your system. Consul is currently packaged as a zip file. We do not have any near term plans to provide system packages.

Once the zip is downloaded, unzip it into any directory. The consul binary inside is all that is necessary to run Consul (or consul.exe for Windows). Any additional files, if any, aren't required to run Consul.

Copy the binary to anywhere on your system. If you intend to access it from the command-line, make sure to place it somewhere on your PATH.

## [»](https://www.consul.io/docs/install/index.html" \l "compiling-from-source)Compiling from Source

To compile from source, you will need [Go](https://golang.org/) installed and configured properly (including a GOPATH environment variable set), as well as a copy of [git](https://www.git-scm.com/) in your PATH.

1. Clone the Consul repository from GitHub into your GOPATH:
2. $ mkdir -p $GOPATH/src/github.com/hashicorp **&&** cd **!**$
3. $ git clone https://github.com/hashicorp/consul.git
4. $ cd consul
5. Bootstrap the project. This will download and compile libraries and tools needed to compile Consul:
6. $ make tools
7. Build Consul for your current system and put the binary in ./bin/ (relative to the git checkout). The make dev target is just a shortcut that builds consul for only your local build environment (no cross-compiled targets).
8. $ make dev

## [»](https://www.consul.io/docs/install/index.html" \l "verifying-the-installation)Verifying the Installation

To verify Consul is properly installed, run consul -v on your system. You should see help output. If you are executing it from the command line, make sure it is on your PATH or you may get an error about Consul not being found.

$ consul -v

# Required Ports

Before running Consul, you should ensure the following bind ports are accessible.

| **Use** | **Default Ports** |
| --- | --- |
| DNS: The DNS server | 8600 |
| HTTP: The HTTP API | 8500 |
| HTTPS: The HTTPs API | disabled (8501)\* |
| gRPC: The gRPC API | disabled (8502)\* |
| LAN Serf: The Serf LAN port. | 8301 |
| Wan Serf: The Serf WAN port | 8302 |
| server: Server RPC address | 8300 |
| Sidecar Proxy Min: Inclusive min port number to use for automatically assigned sidecar service registrations. | 21000 |
| Sidecar Proxy Max: Inclusive max port number to use for automatically assigned sidecar service registrations. | 21255 |

\*For HTTPS and gRPC the ports specified in the table are recommendations.

Note, the default ports can be changed in the [agent configuration](https://www.consul.io/docs/agent/options.html#ports).

# Bootstrapping a Datacenter

An agent can run in either client or server mode. Server nodes are responsible for running the [consensus protocol](https://www.consul.io/docs/internals/consensus.html) and storing the cluster state. The client nodes are mostly stateless and rely heavily on the server nodes.

Before a Consul cluster can begin to service requests, a server node must be elected leader. Bootstrapping is the process of joining these initial server nodes into a cluster. Read the [architecture documentation](https://www.consul.io/docs/internals/architecture.html) to learn more about the internals of Consul.

It is recommended to have three or five total servers per datacenter. A single server deployment is highly discouraged as data loss is inevitable in a failure scenario. Please refer to the [deployment table](https://www.consul.io/docs/internals/consensus.html#deployment-table) for more detail.

**Note**: In versions of Consul prior to 0.4, bootstrapping was a manual process. For details on using the -bootstrapflag directly, see the [manual bootstrapping guide](https://www.consul.io/docs/install/manual-bootstrap.html). Manual bootstrapping with -bootstrap is not recommended in newer versions of Consul (0.5 and newer) as it is more error-prone. Instead you should use automatic bootstrapping with [-bootstrap-expect](https://www.consul.io/docs/agent/options.html#_bootstrap_expect).

## [»](https://www.consul.io/docs/install/bootstrapping.html" \l "bootstrapping-the-servers)Bootstrapping the Servers

The recommended way to bootstrap the servers is to use the [-bootstrap-expect](https://www.consul.io/docs/agent/options.html#_bootstrap_expect) configuration option. This option informs Consul of the expected number of server nodes and automatically bootstraps when that many servers are available. To prevent inconsistencies and split-brain (clusters where multiple servers consider themselves leader) situations, you should either specify the same value for [-bootstrap-expect](https://www.consul.io/docs/agent/options.html#_bootstrap_expect) or specify no value at all on all the servers. Only servers that specify a value will attempt to bootstrap the cluster.

Suppose we are starting a three server cluster. We can start Node A, Node B, and Node C with each providing the -bootstrap-expect 3 flag. Once the nodes are started, you should see a warning message in the service output.

[WARN] raft: EnableSingleNode disabled, and no known peers. Aborting election.

The warning indicates that the nodes are expecting 2 peers but none are known yet. Below you will learn how to connect the servers so that one can be elected leader.

## [»](https://www.consul.io/docs/install/bootstrapping.html" \l "creating-the-cluster)Creating the Cluster

You can trigger leader election by joining the servers together, to create a cluster. You can either configure the nodes to join automatically or manually.

### [»](https://www.consul.io/docs/install/bootstrapping.html" \l "automatically-join-the-servers)Automatically Join the Servers

There are multiple options for joining the servers. Choose the method which best suits your environment and specific use case.

* Specify a list of servers with [-join](https://www.consul.io/docs/agent/options.html#_join) and [start\_join](https://www.consul.io/docs/agent/options.html#start_join) options.
* Specify a list of servers with [-retry-join](https://www.consul.io/docs/agent/options.html#_retry_join) option.
* Use automatic joining by tag for supported cloud environments with the [-retry-join](https://www.consul.io/docs/agent/options.html#_retry_join) option.

All three methods can be set in the agent configuration file or the command line flag.

### [»](https://www.consul.io/docs/install/bootstrapping.html" \l "manually-join-the-servers)Manually Join the Servers

To manually create a cluster, you should connect to one of the servers and run the consul join command.

$ consul join <Node A Address> <Node B Address> <Node C Address>

Successfully joined cluster by contacting 3 nodes.

Since a join operation is symmetric, it does not matter which node initiates it. Once the join is successful, one of the nodes will output something like:

[INFO] consul: adding server foo (Addr: 127.0.0.2:8300) (DC: dc1)

[INFO] consul: adding server bar (Addr: 127.0.0.1:8300) (DC: dc1)

[INFO] consul: Attempting bootstrap with nodes: [127.0.0.3:8300 127.0.0.2:8300 127.0.0.1:8300]

...

[INFO] consul: cluster leadership acquired

### [»](https://www.consul.io/docs/install/bootstrapping.html" \l "verifying-the-cluster-and-connect-the-clients)Verifying the Cluster and Connect the Clients

As a sanity check, the [consul info](https://www.consul.io/docs/commands/info.html) command is a useful tool. It can be used to verify the raft.num\_peers and to view the latest log index under raft.last\_log\_index. When running [consul info](https://www.consul.io/docs/commands/info.html) on the followers, you should see raft.last\_log\_index converge to the same value once the leader begins replication. That value represents the last log entry that has been stored on disk.

Now that the servers are all started and replicating to each other, you can join the clients with the same join method you used for the servers. Clients are much easier as they can join against any existing node. All nodes participate in a gossip protocol to perform basic discovery, so once joined to any member of the cluster, new clients will automatically find the servers and register themselves.

**Note:** It is not strictly necessary to start the server nodes before the clients; however, most operations will fail until the servers are available.

# Upgrading Consul

Consul is meant to be a long-running agent on any nodes participating in a Consul cluster. These nodes consistently communicate with each other. As such, protocol level compatibility and ease of upgrades is an important thing to keep in mind when using Consul.

This page documents how to upgrade Consul when a new version is released.

## [»](https://www.consul.io/docs/upgrading.html" \l "standard-upgrades)Standard Upgrades

For upgrades we strive to ensure backwards compatibility. To support this, nodes gossip their protocol version and builds. This enables clients and servers to intelligently enable new features when available, or to gracefully fallback to a backward compatible mode of operation otherwise.

For most upgrades, the process is simple. Assuming the current version of Consul is A, and version B is released.

1. Check the [version's upgrade notes](https://www.consul.io/docs/upgrade-specific.html) to ensure there are no compatibility issues that will affect your workload. If there are plan accordingly before continuing.
2. On each server, install version B of Consul.
3. One server at a time, shut down version A, restart with version B. Wait until the server is healthy and has rejoined the cluster before moving on to the next server.
4. Once all the servers are upgraded, begin a rollout of clients following the same process.
5. Done! You are now running the latest Consul agent. You can verify this by running consul members to make sure all members have the latest build and highest protocol version.

## [»](https://www.consul.io/docs/upgrading.html" \l "backward-incompatible-upgrades)Backward Incompatible Upgrades

In some cases, a backwards incompatible update may be released. This has not been an issue yet, but to support upgrades we support setting an explicit protocol version. This disables incompatible features and enables a 2-phase upgrade.

For the steps below, assume you're running version A of Consul, and then version B comes out.

1. On each node, install version B of Consul.
2. One server at a time, shut down version A, and start version B with the -protocol=PREVIOUS flag, where "PREVIOUS" is the protocol version of version A (which can be discovered by running consul -v or consul members). Wait until the server is healthy and has rejoined the cluster before moving on to the next server.
3. Once all nodes are running version B, go through every node and restart the version B agent without the -protocolflag, again wait for each server to rejoin the cluster before continuing.
4. Done! You're now running the latest Consul agent speaking the latest protocol. You can verify this is the case by running consul members to make sure all members are speaking the same, latest protocol version.

The key to making this work is the [protocol compatibility](https://www.consul.io/docs/compatibility.html) of Consul. The protocol version system is discussed below.

## [»](https://www.consul.io/docs/upgrading.html" \l "protocol-versions)Protocol Versions

By default, Consul agents speak the latest protocol they can. However, each new version of Consul is also able to speak the previous protocol, if there were any protocol changes.

You can see what protocol versions your version of Consul understands by running consul -v. You'll see output similar to that below:

$ consul -v

Consul v0.7.0

Protocol 2 spoken by default, understands 2 to 3 (agent will automatically use protocol >2 when speaking to compatible agents)

This says the version of Consul as well as the protocol versions this agent speaks and can understand.

Sometimes Consul will default to speak a lower protocol version than it understands, in order to ease compatibility with older agents. For example, Consul agents that understand version 3 claim to speak version 2, and only send version 3 messages to agents that understand version 3. This allows features to upshift automatically as agents are upgraded, and is the strategy used whenever possible. If this is not possible, then you will need to do a backward incompatible upgrade using the instructions above, and such a requirement will be clearly outlined in the notes for a given release.

By specifying the -protocol flag on consul agent, you can tell the Consul agent to speak any protocol version that it can understand. This only specifies the protocol version to speak. Every Consul agent can always understand the entire range of protocol versions it claims to on consul -v.

**By running a previous protocol version**, some features of Consul, especially newer features, may not be available. If this is the case, Consul will typically warn you. In general, you should always upgrade your cluster so that you can run the latest protocol version.

# Protocol Compatibility Promise

We expect Consul to run in large clusters of long-running agents. Because safely upgrading agents in this sort of environment relies heavily on backwards compatibility, we have a strong commitment to keeping different Consul versions protocol-compatible with each other.

We promise that every subsequent release of Consul will remain backwards compatible with at least one prior version. Concretely: version 0.5 can speak to 0.4 (and vice versa) but may not be able to speak to 0.1.

Backwards compatibility is automatic unless otherwise noted. Consul agents by default will speak the latest protocol but can understand earlier ones.

**Note:** If speaking an earlier protocol, new features may not be available.

The ability for an agent to speak an earlier protocol is to ensure that any agent can be upgraded without cluster disruption. Consul agents can be updated one at a time, one version at a time.

For more details on the specifics of upgrading, see the [upgrading page](https://www.consul.io/docs/upgrading.html).

## [»](https://www.consul.io/docs/compatibility.html" \l "protocol-compatibility-table)Protocol Compatibility Table

|  |  |
| --- | --- |
| **Consul Version** | **Protocol Compatibility** |
| 0.1 - 0.3 | 1 |
| 0.4 | 1, 2 |
| 0.5 | 1, 2. 0.5.X servers cannot be mixed with older servers. |
| 0.6 | 1, 2, 3 |
| >= 0.7 | 2, 3. Will automatically use protocol > 2 when speaking to compatible agents |

**Note:** Raft Protocol is versioned separately, but maintains compatibility with at least one prior version. See [here](https://www.consul.io/docs/upgrade-specific.html#raft-protocol-version-compatibility) for details.

# Upgrading Specific Versions

The [upgrading page](https://www.consul.io/docs/upgrading.html) covers the details of doing a standard upgrade. However, specific versions of Consul may have more details provided for their upgrades as a result of new features or changed behavior. This page is used to document those details separately from the standard upgrade flow.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-1-4-0)Consul 1.4.0

There are two major features in Consul 1.4.0 that may impact upgrades: a [new ACL system](https://www.consul.io/docs/upgrade-specific.html#acl-upgrade) and [multi-datacenter support for Connect](https://www.consul.io/docs/upgrade-specific.html#connect-multi-datacenter) in the Enterprise version.

### [»](https://www.consul.io/docs/upgrade-specific.html" \l "acl-upgrade)ACL Upgrade

Consul 1.4.0 includes a [new ACL system](https://www.consul.io/docs/guides/acl.html) that is designed to have a smooth upgrade path but requires care to upgrade components in the right order.

**Note:** As with most major version upgrades, you cannot downgrade once the upgrade to 1.4.0 is complete as it adds new state to the raft store. As always it is strongly recommended that you test the upgrade first outside of production and ensure you take backup snapshots of all datacenters before upgrading.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "primary-datacenter)Primary Datacenter

The "ACL datacenter" in 1.3.x and earlier is now referred to as the "Primary datacenter". All configuration is backwards compatible and shouldn't need to change prior to upgrade although it's strongly recommended to migrate ACL configuration to the new syntax soon after upgrade. This includes moving to primary\_datacenter rather than acl\_datacenter and acl\_\* to the new [ACL block](https://www.consul.io/docs/agent/options.html#acl).

Datacenters can be upgraded in any order although secondaries will remain in [Legacy ACL mode](https://www.consul.io/docs/upgrade-specific.html#legacy-acl-mode) until the primary datacenter is fully ugraded.

Each datacenter should follow the [standard rolling upgrade procedure](https://www.consul.io/docs/upgrading.html#standard-upgrades).

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "legacy-acl-mode)Legacy ACL Mode

When a 1.4.0 server first starts, it runs in "Legacy ACL mode". In this mode, bootstrap requests and new ACL APIs will not be functional yet and will return an error. The server advertises it's ability to support 1.4.0 ACLs via gossip and waits.

In the primary datacenter, the servers all wait in legacy ACL mode until they see every server in the primary datacenter advertise 1.4.0 ACL support. Once this happens, the leader will complete the transition out of "legacy ACL mode" and write this into the state so future restarts don't need to go through the same transition.

In a secondary datacenter, the same process happens except that servers additionally wait for all servers in the primary datacenter making it safe to upgrade datacenters in any order.

It should be noted that even if you are not upgrading, starting a brand new 1.4.0 cluster will transition through legacy ACL mode so you may be unable to bootstrap ACLs until all the expected servers are up and healthy.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "legacy-token-accessor-migration)Legacy Token Accessor Migration

As soon as all servers in the primary datacenter have been upgraded to 1.4.0, the leader will begin the process of creating new accessor IDs for all existing ACL tokens.

This process completes in the background and is rate limited to ensure it doesn't overload the leader. It completes upgrades in batches of 128 tokens and will not upgrade more than one batch per second so on a cluster with 10,000 tokens, this may take several minutes.

While this is happening both old and new ACLs will work correctly with the caveat that new ACL [Token APIs](https://www.consul.io/api/acl/tokens.html) may not return an accessor ID for legacy tokens that are not yet migrated.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "migrating-existing-acls)Migrating Existing ACLs

New ACL policies have slightly different syntax designed to fix some shortcomings in old ACL syntax. During and after the upgrade process, any old ACL tokens will continue to work and grant exactly the same level of access.

After upgrade, it is still possible to create "legacy" tokens using the existing API so existing integrations that create tokens (e.g. Vault) will continue to work. The "legacy" tokens generated though will not be able to take advantage of new policy features. It's recommended that you complete migration of all tokens as soon as possible after upgrade, as well as updating any integrations to work with the the new ACL [Token](https://www.consul.io/api/acl/tokens.html) and [Policy](https://www.consul.io/api/acl/policies.html) APIs.

More complete details on how to upgrade "legacy" tokens is available [here](https://www.consul.io/docs/guides/acl-migrate-tokens.html).

### [»](https://www.consul.io/docs/upgrade-specific.html" \l "connect-multi-datacenter)Connect Multi-datacenter

This only applies to users upgrading from an older version of Consul Enterprise to Consul Enterprise 1.4.0 (all license types).

In addition, this upgrade will only affect clusters where [Connect is enabled](https://www.consul.io/docs/connect/configuration.html) on your servers before the migration.

Connect multi-datacenter uses the same primary/secondary approach as ACLs and will use the same [primary\_datacenter](https://www.consul.io/docs/upgrade-specific.html#primary-datacenter). When a secondary datacenter server restarts with 1.4.0 it will detect it is not the primary and begin an automatic bootstrap of multi-datacenter CA federation.

Datacenters can be upgraded in either order; secondary datacenters will not switch into multi-datacenter mode until all servers in both the secondary and primary datacenter are detected to be running at least Consul 1.4.0. Secondary datacenters monitor this periodically (every few minutes) and will automatically upgrade Connect to use a federated Certificate Authority when they do.

In general, migrating a Consul cluster from OSS to Enterprise will update the CA to be federated automatically and without impact on Connect traffic. When upgrading Consul Enterprise 1.3.x to Consul Enterprise 1.4.0 upgrades the CA upgrade is seamless, however depending on the size of the cluster, new connection attempts in the secondary datacenter might fail for a short window (typically seconds) while the update is propagated due to the 1.3.x Beta authorization endpoint validating originating cluster in a way that was not fully forwards compatible with migrating between cluster trust domains. That issue is fixed in 1.4.0 as part of General Availability.

Once migrated (typically a few seconds). Connect will use the primary datacenter's Certificate Authority as the root of trust for all other datacenters. CA migration or root key changes in the primary will now rotate automatically and without loss of connectivity throughout all datacenters and workloads.

For more information see [Connect Multi-datacenter](https://www.consul.io/docs/enterprise/connect-multi-datacenter/index.html).

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-1-3-0)Consul 1.3.0

This version added support for multiple tag filters in service discovery queries, however it introduced a subtle bug where API calls to /catalog/service/:name?tag=<tag> would ignore the tag filter only during the upgrade. It only occurs when clients are still running 1.2.3 or earlier but servers have been upgraded. The /health/service/:name?tag=<tag>endpoint and DNS interface were not affected.

For this reason, we recommend you upgrade directly to 1.3.1 which includes only a fix for this issue.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-1-1-0)Consul 1.1.0

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "removal-of-deprecated-features)Removal of Deprecated Features

The following previously deprecated fields and config options have been removed:

* [CheckID](https://www.consul.io/docs/upgrade-specific.html" \l "checkid) has been removed from config file check definitions (use id instead).

* [script](https://www.consul.io/docs/upgrade-specific.html" \l "script) has been removed from config file check definitions (use args instead).

* [enableTagOverride](https://www.consul.io/docs/upgrade-specific.html" \l "enabletagoverride) is no longer valid in service definitions (use enable\_tag\_override instead).
* The [deprecated set of metric names](https://www.consul.io/docs/upgrade-specific.html#metric-names-updated) (beginning with consul.consul.) has been removed along with the enable\_deprecated\_names option from the metrics configuration.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "new-defaults-for-raft-snapshot-creation)New defaults for Raft Snapshot Creation

Consul 1.0.1 (and earlier versions of Consul) checked for raft snapshots every 5 seconds, and created new snapshots for every 8192 writes. These defaults cause constant disk IO in large busy clusters. Consul 1.1.0 increases these to larger values, and makes them tunable via the [raft\_snapshot\_interval](https://www.consul.io/docs/agent/options.html#_raft_snapshot_interval) and [raft\_snapshot\_threshold](https://www.consul.io/docs/agent/options.html#_raft_snapshot_threshold) parameters. We recommend keeping the new defaults. However, operators can go back to the old defaults by changing their config if they prefer more frequent snapshots. See the documentation for [raft\_snapshot\_interval](https://www.consul.io/docs/agent/options.html#_raft_snapshot_interval) and [raft\_snapshot\_threshold](https://www.consul.io/docs/agent/options.html#_raft_snapshot_threshold) to understand the trade-offs when tuning these.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-1-0-7)Consul 1.0.7

When requesting a specific service (/v1/health/:service or /v1/catalog/:service endpoints), the X-Consul-Indexreturned is now the index at which that specific service was last modified. In version 1.0.6 and earlier the X-Consul-Indexreturned was the index at which any service was last modified. See [GH-3890](https://github.com/hashicorp/consul/issues/3890) for more details.

During upgrades from 1.0.6 or lower to 1.0.7 or higher, watchers are likely to see X-Consul-Index for these endpoints decrease between blocking calls.

Consul’s watch feature and consul-template should gracefully handle this case. Other tools relying on blocking service or health queries are also likely to work; some may require a restart. It is possible external tools could break and either stop working or continually re-request data without blocking if they have assumed indexes can never decrease or be reset and/or persist index values. Please test any blocking query integrations in a controlled environment before proceeding.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-1-0-1)Consul 1.0.1

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "carefully-check-and-remove-stale-servers-during-rolling-upgrades)Carefully Check and Remove Stale Servers During Rolling Upgrades

Consul 1.0 (and earlier versions of Consul when running with [Raft protocol 3](https://www.consul.io/docs/agent/options.html#_raft_protocol) had an issue where performing rolling updates of Consul servers could result in an outage from old servers remaining in the cluster. [Autopilot](https://www.consul.io/docs/guides/autopilot.html) would normally remove old servers when new ones come online, but it was also waiting to promote servers to voters in pairs to maintain an odd quorum size. The pairwise promotion feature was removed so that servers become voters as soon as they are stable, allowing Autopilot to remove old servers in a safer way.

When upgrading from Consul 1.0, you may need to manually [force-leave](https://www.consul.io/docs/commands/force-leave.html) old servers as part of a rolling update to Consul 1.0.1.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-1-0)Consul 1.0

Consul 1.0 has several important breaking changes that are documented here. Please be sure to read over all the details here before upgrading.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "raft-protocol-now-defaults-to-3)Raft Protocol Now Defaults to 3

The [-raft-protocol](https://www.consul.io/docs/agent/options.html#_raft_protocol) default has been changed from 2 to 3, enabling all [Autopilot](https://www.consul.io/docs/guides/autopilot.html) features by default.

Raft protocol version 3 requires Consul running 0.8.0 or newer on all servers in order to work, so if you are upgrading with older servers in a cluster then you will need to set this back to 2 in order to upgrade. See [Raft Protocol Version Compatibility](https://www.consul.io/docs/upgrade-specific.html#raft-protocol-version-compatibility) for more details. Also the format of peers.json used for outage recovery is different when running with the latest Raft protocol. See [Manual Recovery Using peers.json](https://www.consul.io/docs/guides/outage.html#manual-recovery-using-peers-json) for a description of the required format.

Please note that the Raft protocol is different from Consul's internal protocol as described on the [Protocol Compatibility Promise](https://www.consul.io/docs/compatibility.html) page, and as is shown in commands like consul members and consul version. To see the version of the Raft protocol in use on each server, use the consul operator raft list-peers command.

The easiest way to upgrade servers is to have each server leave the cluster, upgrade its Consul version, and then add it back. Make sure the new server joins successfully and that the cluster is stable before rolling the upgrade forward to the next server. It's also possible to stand up a new set of servers, and then slowly stand down each of the older servers in a similar fashion.

When using Raft protocol version 3, servers are identified by their [-node-id](https://www.consul.io/docs/agent/options.html#_node_id) instead of their IP address when Consul makes changes to its internal Raft quorum configuration. This means that once a cluster has been upgraded with servers all running Raft protocol version 3, it will no longer allow servers running any older Raft protocol versions to be added. If running a single Consul server, restarting it in-place will result in that server not being able to elect itself as a leader. To avoid this, either set the Raft protocol back to 2, or use [Manual Recovery Using peers.json](https://www.consul.io/docs/guides/outage.html#manual-recovery-using-peers-json) to map the server to its node ID in the Raft quorum configuration.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "config-files-require-an-extension)Config Files Require an Extension

As part of supporting the [HCL](https://github.com/hashicorp/hcl#syntax) format for Consul's config files, an .hcl or .json extension is required for all config files loaded by Consul, even when using the [-config-file](https://www.consul.io/docs/agent/options.html#_config_file) argument to specify a file directly.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "deprecated-options-have-been-removed)Deprecated Options Have Been Removed

All of Consul's previously deprecated command line flags and config options have been removed, so these will need to be mapped to their equivalents before upgrading. Here's the complete list of removed options and their equivalents:

| **Removed Option** | **Equivalent** |
| --- | --- |
| -dc | [-datacenter](https://www.consul.io/docs/agent/options.html#_datacenter) |
| -retry-join-azure-tag-name | [-retry-join](https://www.consul.io/docs/agent/options.html#microsoft-azure) |
| -retry-join-azure-tag-value | [-retry-join](https://www.consul.io/docs/agent/options.html#microsoft-azure) |
| -retry-join-ec2-region | [-retry-join](https://www.consul.io/docs/agent/options.html#amazon-ec2) |
| -retry-join-ec2-tag-key | [-retry-join](https://www.consul.io/docs/agent/options.html#amazon-ec2) |
| -retry-join-ec2-tag-value | [-retry-join](https://www.consul.io/docs/agent/options.html#amazon-ec2) |
| -retry-join-gce-credentials-file | [-retry-join](https://www.consul.io/docs/agent/options.html#google-compute-engine) |
| -retry-join-gce-project-name | [-retry-join](https://www.consul.io/docs/agent/options.html#google-compute-engine) |
| -retry-join-gce-tag-name | [-retry-join](https://www.consul.io/docs/agent/options.html#google-compute-engine) |
| -retry-join-gce-zone-pattern | [-retry-join](https://www.consul.io/docs/agent/options.html#google-compute-engine) |
| addresses.rpc | None, the RPC server for CLI commands is no longer supported. |
| advertise\_addrs | [ports](https://www.consul.io/docs/agent/options.html#ports) with [advertise\_addr](https://www.consul.io/docs/agent/options.html#advertise_addr) and/or [advertise\_addr\_wan](https://www.consul.io/docs/agent/options.html#advertise_addr_wan) |
| dogstatsd\_addr | [telemetry.dogstatsd\_addr](https://www.consul.io/docs/agent/options.html#telemetry-dogstatsd_addr) |
| dogstatsd\_tags | [telemetry.dogstatsd\_tags](https://www.consul.io/docs/agent/options.html#telemetry-dogstatsd_tags) |
| http\_api\_response\_headers | [http\_config.response\_headers](https://www.consul.io/docs/agent/options.html#response_headers) |
| ports.rpc | None, the RPC server for CLI commands is no longer supported. |
| recursor | [recursors](https://github.com/hashicorp/consul/blob/master/website/source/docs/agent/options.html.md#recursors) |
| retry\_join\_azure | [-retry-join](https://www.consul.io/docs/agent/options.html#microsoft-azure) |
| retry\_join\_ec2 | [-retry-join](https://www.consul.io/docs/agent/options.html#amazon-ec2) |
| retry\_join\_gce | [-retry-join](https://www.consul.io/docs/agent/options.html#google-compute-engine) |
| statsd\_addr | [telemetry.statsd\_address](https://github.com/hashicorp/consul/blob/master/website/source/docs/agent/options.html.md#telemetry-statsd_address) |
| statsite\_addr | [telemetry.statsite\_address](https://github.com/hashicorp/consul/blob/master/website/source/docs/agent/options.html.md#telemetry-statsite_address) |
| statsite\_prefix | [telemetry.metrics\_prefix](https://www.consul.io/docs/agent/options.html#telemetry-metrics_prefix) |
| telemetry.statsite\_prefix | [telemetry.metrics\_prefix](https://www.consul.io/docs/agent/options.html#telemetry-metrics_prefix) |
| (service definitions) serviceid | [service\_id](https://www.consul.io/docs/agent/services.html) |
| (service definitions) dockercontainerid | [docker\_container\_id](https://www.consul.io/docs/agent/services.html) |
| (service definitions) tlsskipverify | [tls\_skip\_verify](https://www.consul.io/docs/agent/services.html) |
| (service definitions) deregistercriticalserviceafter | [deregister\_critical\_service\_after](https://www.consul.io/docs/agent/services.html) |

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "statsite_prefix-renamed-to-metrics_prefix)statsite\_prefix Renamed to metrics\_prefix

Since the statsite\_prefix configuration option applied to all telemetry providers, statsite\_prefix was renamed to [metrics\_prefix](https://www.consul.io/docs/agent/options.html#telemetry-metrics_prefix). Configuration files will need to be updated when upgrading to this version of Consul.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "advertise_addrs-removed)advertise\_addrs Removed

This configuration option was removed since it was redundant with advertise\_addr and advertise\_addr\_wan in combination with ports and also wrongly stated that you could configure both host and port.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "escaping-behavior-changed-for-go-discover-configs)Escaping Behavior Changed for go-discover Configs

The format for [-retry-join](https://www.consul.io/docs/agent/options.html#retry-join) and [-retry-join-wan](https://www.consul.io/docs/agent/options.html#retry-join-wan) values that use [go-discover](https://github.com/hashicorp/go-discover) cloud auto joining has changed. Values in key=val sequences must no longer be URL encoded and can be provided as literals as long as they do not contain spaces, backslashes \ or double quotes ". If values contain these characters then use double quotes as in "some key"="some value". Special characters within a double quoted string can be escaped with a backslash \.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "http-verbs-are-enforced-in-many-http-apis)HTTP Verbs are Enforced in Many HTTP APIs

Many endpoints in the HTTP API that previously took any HTTP verb now check for specific HTTP verbs and enforce them. This may break clients relying on the old behavior. Here's the complete list of updated endpoints and required HTTP verbs:

| **Endpoint** | **Required HTTP Verb** |
| --- | --- |
| /v1/acl/info | GET |
| /v1/acl/list | GET |
| /v1/acl/replication | GET |
| /v1/agent/check/deregister | PUT |
| /v1/agent/check/fail | PUT |
| /v1/agent/check/pass | PUT |
| /v1/agent/check/register | PUT |
| /v1/agent/check/warn | PUT |
| /v1/agent/checks | GET |
| /v1/agent/force-leave | PUT |
| /v1/agent/join | PUT |
| /v1/agent/members | GET |
| /v1/agent/metrics | GET |
| /v1/agent/self | GET |
| /v1/agent/service/register | PUT |
| /v1/agent/service/deregister | PUT |
| /v1/agent/services | GET |
| /v1/catalog/datacenters | GET |
| /v1/catalog/deregister | PUT |
| /v1/catalog/node | GET |
| /v1/catalog/nodes | GET |
| /v1/catalog/register | PUT |
| /v1/catalog/service | GET |
| /v1/catalog/services | GET |
| /v1/coordinate/datacenters | GET |
| /v1/coordinate/nodes | GET |
| /v1/health/checks | GET |
| /v1/health/node | GET |
| /v1/health/service | GET |
| /v1/health/state | GET |
| /v1/internal/ui/node | GET |
| /v1/internal/ui/nodes | GET |
| /v1/internal/ui/services | GET |
| /v1/session/info | GET |
| /v1/session/list | GET |
| /v1/session/node | GET |
| /v1/status/leader | GET |
| /v1/status/peers | GET |
| /v1/operator/area/:uuid/members | GET |
| /v1/operator/area/:uuid/join | PUT |

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "unauthorized-kv-requests-return-403)Unauthorized KV Requests Return 403

When ACLs are enabled, reading a key with an unauthorized token returns a 403. This previously returned a 404 response.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "config-section-of-agent-self-endpoint-has-changed)Config Section of Agent Self Endpoint has Changed

The /v1/agent/self endpoint's Config section has often been in flux as it was directly returning one of Consul's internal data structures. This configuration structure has been moved under DebugConfig, and is documents as for debugging use and subject to change, and a small set of elements of Config have been maintained and documented. See [Read Configuration](https://www.consul.io/api/agent.html#read-configuration) endpoint documentation for details.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "deprecated-configtest-command-removed)Deprecated configtest Command Removed

The configtest command was deprecated and has been superseded by the validate command.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "undocumented-flags-in-validate-command-removed)Undocumented Flags in validate Command Removed

The validate command supported the -config-file and -config-dir command line flags but did not document them. This support has been removed since the flags are not required.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "metric-names-updated)Metric Names Updated

Metric names no longer start with consul.consul. To help with transitioning dashboards and other metric consumers, the field enable\_deprecated\_names has been added to the telemetry section of the config, which will enable metrics with the old naming scheme to be sent alongside the new ones. The following prefixes were affected:

| **Prefix** |
| --- |
| consul.consul.acl |
| consul.consul.autopilot |
| consul.consul.catalog |
| consul.consul.fsm |
| consul.consul.health |
| consul.consul.http |
| consul.consul.kvs |
| consul.consul.leader |
| consul.consul.prepared-query |
| consul.consul.rpc |
| consul.consul.session |
| consul.consul.session\_ttl |
| consul.consul.txn |

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "checks-validated-on-agent-startup)Checks Validated On Agent Startup

Consul agents now validate health check definitions in their configuration and will fail at startup if any checks are invalid. In previous versions of Consul, invalid health checks would get skipped.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-0-9-0)Consul 0.9.0

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "script-checks-are-now-opt-in)Script Checks Are Now Opt-In

A new [enable\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_script_checks) configuration option was added, and defaults to false, meaning that in order to allow an agent to run health checks that execute scripts, this will need to be configured and set to true. This provides a safer out-of-the-box configuration for Consul where operators must opt-in to allow script-based health checks.

If your cluster uses script health checks please be sure to set this to true as part of upgrading agents. If this is set to true, you should also enable [ACLs](https://www.consul.io/docs/guides/acl.html) to provide control over which users are allowed to register health checks that could potentially execute scripts on the agent machines.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "web-ui-is-no-longer-released-separately)Web UI Is No Longer Released Separately

Consul releases will no longer include a web\_ui.zip file with the compiled web assets. These have been built in to the Consul binary since the 0.7.x series and can be enabled with the [-ui](https://www.consul.io/docs/agent/options.html#_ui) configuration option. These built-in web assets have always been identical to the contents of the web\_ui.zip file for each release. The [-ui-dir](https://www.consul.io/docs/agent/options.html#_ui_dir) option is still available for hosting customized versions of the web assets, but the vast majority of Consul users can just use the built in web assets.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-0-8-0)Consul 0.8.0

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "upgrade-current-cluster-leader-last)Upgrade Current Cluster Leader Last

We identified a potential issue with Consul 0.8 that requires the current cluster leader to be upgraded last when updating multiple servers. Please see [this issue](https://github.com/hashicorp/consul/issues/2889) for more details.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "command-line-interface-rpc-deprecation)Command-Line Interface RPC Deprecation

The RPC client interface has been removed. All CLI commands that used RPC and the -rpc-addr flag to communicate with Consul have been converted to use the HTTP API and the appropriate flags for it, and the rpc field has been removed from the port and address binding configs. You will need to remove these fields from your config files and update any scripts that passed a custom -rpc-addr to the following commands:

* [force-leave](https://www.consul.io/docs/upgrade-specific.html" \l "force-leave)

* [info](https://www.consul.io/docs/upgrade-specific.html" \l "info)

* [join](https://www.consul.io/docs/upgrade-specific.html" \l "join)

* [keyring](https://www.consul.io/docs/upgrade-specific.html" \l "keyring)

* [leave](https://www.consul.io/docs/upgrade-specific.html" \l "leave)

* [members](https://www.consul.io/docs/upgrade-specific.html" \l "members)

* [monitor](https://www.consul.io/docs/upgrade-specific.html" \l "monitor)

* [reload](https://www.consul.io/docs/upgrade-specific.html" \l "reload)

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "version-8-acls-are-now-opt-out)Version 8 ACLs Are Now Opt-Out

The [acl\_enforce\_version\_8](https://www.consul.io/docs/agent/options.html#acl_enforce_version_8) configuration now defaults to true to enable [full version 8 ACL support](https://www.consul.io/docs/guides/acl.html#version_8_acls) by default. If you are upgrading an existing cluster with ACLs enabled, you will need to set this to false during the upgrade on **both Consul agents and Consul servers**. Version 8 ACLs were also changed so that [acl\_datacenter](https://www.consul.io/docs/agent/options.html#acl_datacenter) must be set on agents in order to enable the agent-side enforcement of ACLs. This makes for a smoother experience in clusters where ACLs aren't enabled at all, but where the agents would have to wait to contact a Consul server before learning that.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "remote-exec-is-now-opt-in)Remote Exec Is Now Opt-In

The default for [disable\_remote\_exec](https://www.consul.io/docs/agent/options.html#disable_remote_exec) was changed to "true", so now operators need to opt-in to having agents support running commands remotely via [consul exec](https://www.consul.io/docs/commands/exec.html).

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "raft-protocol-version-compatibility)Raft Protocol Version Compatibility

When upgrading to Consul 0.8.0 from a version lower than 0.7.0, users will need to set the [-raft-protocol](https://www.consul.io/docs/agent/options.html#_raft_protocol) option to 1 in order to maintain backwards compatibility with the old servers during the upgrade. After the servers have been migrated to version 0.8.0, -raft-protocol can be moved up to 2 and the servers restarted to match the default.

The Raft protocol must be stepped up in this way; only adjacent version numbers are compatible (for example, version 1 cannot talk to version 3). Here is a table of the Raft Protocol versions supported by each Consul version:

|  |  |
| --- | --- |
| **Version** | **Supported Raft Protocols** |
| 0.6 and earlier | 0 |
| 0.7 | 1 |
| 0.8 | 1, 2, 3 |

In order to enable all [Autopilot](https://www.consul.io/docs/guides/autopilot.html) features, all servers in a Consul cluster must be running with Raft protocol version 3 or later.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-0-7-1)Consul 0.7.1

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "child-process-reaping)Child Process Reaping

Child process reaping support has been removed, along with the reap configuration option. Reaping is also done via [dumb-init](https://github.com/Yelp/dumb-init) in the [Consul Docker image](https://github.com/hashicorp/docker-consul), so removing it from Consul itself simplifies the code and eases future maintenance for Consul. If you are running Consul as PID 1 in a container you will need to arrange for a wrapper process to reap child processes.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "dns-resiliency-defaults)DNS Resiliency Defaults

The default for [max\_stale](https://www.consul.io/docs/agent/options.html#max_stale) has been increased from 5 seconds to a near-indefinite threshold (10 years) to allow DNS queries to continue to be served in the event of a long outage with no leader. A new telemetry counter was added at consul.dns.stale\_queries to track when agents serve DNS queries that are stale by more than 5 seconds.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-0-7)Consul 0.7

Consul version 0.7 is a very large release with many important changes. Changes to be aware of during an upgrade are categorized below.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "performance-timing-defaults-and-tuning)Performance Timing Defaults and Tuning

Consul 0.7 now defaults the DNS configuration to allow for stale queries by defaulting [allow\_stale](https://www.consul.io/docs/agent/options.html#allow_stale) to true for better utilization of available servers. If you want to retain the previous behavior, set the following configuration:

{

"dns\_config": {

"allow\_stale": **false**

}

}

Consul also 0.7 introduced support for tuning Raft performance using a new [performance configuration block](https://www.consul.io/docs/agent/options.html#performance). Also, the default Raft timing is set to a lower-performance mode suitable for [minimal Consul servers](https://www.consul.io/docs/guides/performance.html#minimum).

To continue to use the high-performance settings that were the default prior to Consul 0.7 (recommended for production servers), add the following configuration to all Consul servers when upgrading:

{

"performance": {

"raft\_multiplier": 1

}

}

See the [Server Performance](https://www.consul.io/docs/guides/performance.html) guide for more details.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "leave-related-configuration-defaults)Leave-Related Configuration Defaults

The default behavior of [leave\_on\_terminate](https://www.consul.io/docs/agent/options.html#leave_on_terminate) and [skip\_leave\_on\_interrupt](https://www.consul.io/docs/agent/options.html#skip_leave_on_interrupt) are now dependent on whether or not the agent is acting as a server or client:

* For servers, leave\_on\_terminate defaults to "false" and skip\_leave\_on\_interrupt defaults to "true".
* For clients, leave\_on\_terminate defaults to "true" and skip\_leave\_on\_interrupt defaults to "false".

These defaults are designed to be safer for servers so that you must explicitly configure them to leave the cluster. This also results in a better experience for clients, especially in cloud environments where they may be created and destroyed often and users prefer not to wait for the 72 hour reap time for cleanup.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "dropped-support-for-protocol-version-1)Dropped Support for Protocol Version 1

Consul version 0.7 dropped support for protocol version 1, which means it is no longer compatible with versions of Consul prior to 0.3. You will need to upgrade all agents to a newer version of Consul before upgrading to Consul 0.7.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "prepared-query-changes)Prepared Query Changes

Consul version 0.7 adds a feature which allows prepared queries to store a [Near parameter](https://www.consul.io/api/query.html#near) in the query definition itself. This feature enables using the distance sorting features of prepared queries without explicitly providing the node to sort near in requests, but requires the agent servicing a request to send additional information about itself to the Consul servers when executing the prepared query. Agents prior to 0.7 do not send this information, which means they are unable to properly execute prepared queries configured with a Near parameter. Similarly, any server nodes prior to version 0.7 are unable to store the Near parameter, making them unable to properly serve requests for prepared queries using the feature. It is recommended that all agents be running version 0.7 prior to using this feature.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "wan-address-translation-in-http-endpoints)WAN Address Translation in HTTP Endpoints

Consul version 0.7 added support for translating WAN addresses in certain [HTTP endpoints](https://www.consul.io/docs/agent/options.html#translate_wan_addrs). The servers and the agents need to be running version 0.7 or later in order to use this feature.

These translated addresses could break HTTP endpoint consumers that are expecting local addresses, so a new [X-Consul-Translate-Addresses](https://www.consul.io/api/index.html#translate_header) header was added to allow clients to detect if translation is enabled for HTTP responses. A "lan" tag was added to TaggedAddresses for clients that need the local address regardless of translation.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "outage-recovery-and-peers-json-changes)Outage Recovery and peers.json Changes

The peers.json file is no longer present by default and is only used when performing recovery. This file will be deleted after Consul starts and ingests the file. Consul 0.7 also uses a new, automatically-created raft/peers.info file to avoid ingesting the peers.json file on the first start after upgrading (the peers.json file is simply deleted on the first start after upgrading).

Please be sure to review the [Outage Recovery Guide](https://www.consul.io/docs/guides/outage.html) before upgrading for more details.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-0-6-4)Consul 0.6.4

Consul 0.6.4 made some substantial changes to how ACLs work with prepared queries. Existing queries will execute with no changes, but there are important differences to understand about how prepared queries are managed before you upgrade. In particular, prepared queries with no Name defined will no longer require any ACL to manage them, and prepared queries with a Name defined are now governed by a new query ACL policy that will need to be configured after the upgrade.

See the [ACL Guide](https://www.consul.io/docs/guides/acl.html#prepared_query_acls) for more details about the new behavior and how it compares to previous versions of Consul.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-0-6)Consul 0.6

Consul version 0.6 is a very large release with many enhancements and optimizations. Changes to be aware of during an upgrade are categorized below.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "data-store-changes)Data Store Changes

Consul changed the format used to store data on the server nodes in version 0.5 (see 0.5.1 notes below for details). Previously, Consul would automatically detect data directories using the old LMDB format, and convert them to the newer BoltDB format. This automatic upgrade has been removed for Consul 0.6, and instead a safeguard has been put in place which will prevent Consul from booting if the old directory format is detected.

It is still possible to migrate from a 0.5.x version of Consul to 0.6+ using the [consul-migrate](https://github.com/hashicorp/consul-migrate) CLI utility. This is the same tool that was previously embedded into Consul. See the [releases](https://github.com/hashicorp/consul-migrate/releases) page for downloadable versions of the tool.

Also, in this release Consul switched from LMDB to a fully in-memory database for the state store. Because LMDB is a disk-based backing store, it was able to store more data than could fit in RAM in some cases (though this is not a recommended configuration for Consul). If you have an extremely large data set that won't fit into RAM, you may encounter issues upgrading to Consul 0.6.0 and later. Consul should be provisioned with physical memory approximately 2X the data set size to allow for bursty allocations and subsequent garbage collection.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "acl-enhancements)ACL Enhancements

Consul 0.6 introduces enhancements to the ACL system which may require special handling:

* Service ACLs are enforced during service discovery (REST + DNS)

Previously, service discovery was wide open, and any client could query information about any service without providing a token. Consul now requires read-level access at a minimum when ACLs are enabled to return service information over the REST or DNS interfaces. If clients depend on an open service discovery system, then the following should be added to all ACL tokens which require it:

# Enable discovery of all services

service "" {

policy = "read"

}

When the DNS interface is queried, the agent's [acl\_token](https://www.consul.io/docs/agent/options.html#acl_token) is used, so be sure that token has sufficient privileges to return the DNS records you expect to retrieve from it.

* Event and keyring ACLs

Similar to service discovery, the new event and keyring ACLs will block access to these operations if the acl\_default\_policy is set to deny. If clients depend on open access to these, then the following should be added to all ACL tokens which require them:

event "" {

policy = "write"

}

keyring = "write"

Unfortunately, these are new ACLs for Consul 0.6, so they must be added after the upgrade is complete.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "prepared-queries)Prepared Queries

Prepared queries introduce a new Raft log entry type that isn't supported on older versions of Consul. It's important to not use the prepared query features of Consul until all servers in a cluster have been upgraded to version 0.6.0.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "single-private-ip-enforcement)Single Private IP Enforcement

Consul will refuse to start if there are multiple private IPs available, so if this is the case you will need to configure Consul's advertise or bind addresses before upgrading.

#### [»](https://www.consul.io/docs/upgrade-specific.html" \l "new-web-ui-file-layout)New Web UI File Layout

The release .zip file for Consul's web UI no longer contains a dist sub-folder; everything has been moved up one level. If you have any automated scripts that expect the old layout you may need to update them.

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-0-5-1)Consul 0.5.1

Consul version 0.5.1 uses a different backend store for persisting the Raft log. Because of this change, a data migration is necessary to move the log entries out of LMDB and into the newer backend, BoltDB.

Consul version 0.5.1+ makes this transition seamless and easy. As a user, there are no special steps you need to take. When Consul starts, it checks for presence of the legacy LMDB data files, and migrates them automatically if any are found. You will see a log emitted when Raft data is migrated, like this:

==> Successfully migrated raft data in 5.839642ms

This automatic upgrade will only exist in Consul 0.5.1+ and it will be removed starting with Consul 0.6.0+. It will still be possible to upgrade directly from pre-0.5.1 versions by using the consul-migrate utility, which is available on the [Consul Tools page](https://www.consul.io/downloads_tools.html).

## [»](https://www.consul.io/docs/upgrade-specific.html" \l "consul-0-5)Consul 0.5

Consul version 0.5 adds two features that complicate the upgrade process:

* ACL system includes service discovery and registration
* Internal use of tombstones to fix behavior of blocking queries in certain edge cases.

Users of the ACL system need to be aware that deploying Consul 0.5 will cause service registration to be enforced. This means if an agent attempts to register a service without proper privileges it will be denied. If the acl\_default\_policy is "allow" then clients will continue to work without an updated policy. If the policy is "deny", then all clients will begin to have their registration rejected causing issues.

To avoid this situation, all the ACL policies should be updated to add something like this:

# Enable all services to be registered

service "" {

policy = "write"

}

This will set the service policy to write level for all services. The blank service name is the catch-all value. A more specific service can also be specified:

# Enable only the API service to be registered

service "api" {

policy = "write"

}

The ACL policy can be updated while running 0.4, and enforcement will being with the upgrade to 0.5. The policy updates will ensure the availability of the cluster.

The second major change is the new internal command used for tombstones. The details of the change are not important, however to function the leader node will replicate a new command to its followers. Consul is designed defensively, and when a command that is not recognized is received, the server will panic. This is a purposeful design decision to avoid the possibility of data loss, inconsistencies, or security issues caused by future incompatibility.

In practice, this means if a Consul 0.5 node is the leader, all of its followers must also be running 0.5. There are a number of ways to do this to ensure cluster availability:

* Add new 0.5 nodes, then remove the old servers. This will add the new nodes as followers, and once the old servers are removed, one of the 0.5 nodes will become leader.
* Upgrade the followers first, then the leader last. Using consul info, you can determine which nodes are followers. Do an in-place upgrade on them first, and finally upgrade the leader last.
* Upgrade them in any order, but ensure all are done within 15 minutes. Even if the leader is upgraded to 0.5 first, as long as all of the followers are running 0.5 within 15 minutes there will be no issues.

Finally, even if any of the methods above are not possible or the process fails for some reason, it is not fatal. The older version of the server will simply panic and stop. At that point, you can upgrade to the new version and restart the agent. There will be no data loss and the cluster will resume operations.

Consul Internals

This section covers some of the internals of Consul, such as the architecture, consensus and gossip protocols, and security model.

**Note:** Knowing about the internals of Consul is not necessary to successfully use it. We document it here to be completely transparent about how Consul works.

# Consul Architecture

Consul is a complex system that has many different moving parts. To help users and developers of Consul form a mental model of how it works, this page documents the system architecture.

**Advanced Topic!** This page covers technical details of the internals of Consul. You don't need to know these details to effectively operate and use Consul. These details are documented here for those who wish to learn about them without having to go spelunking through the source code.

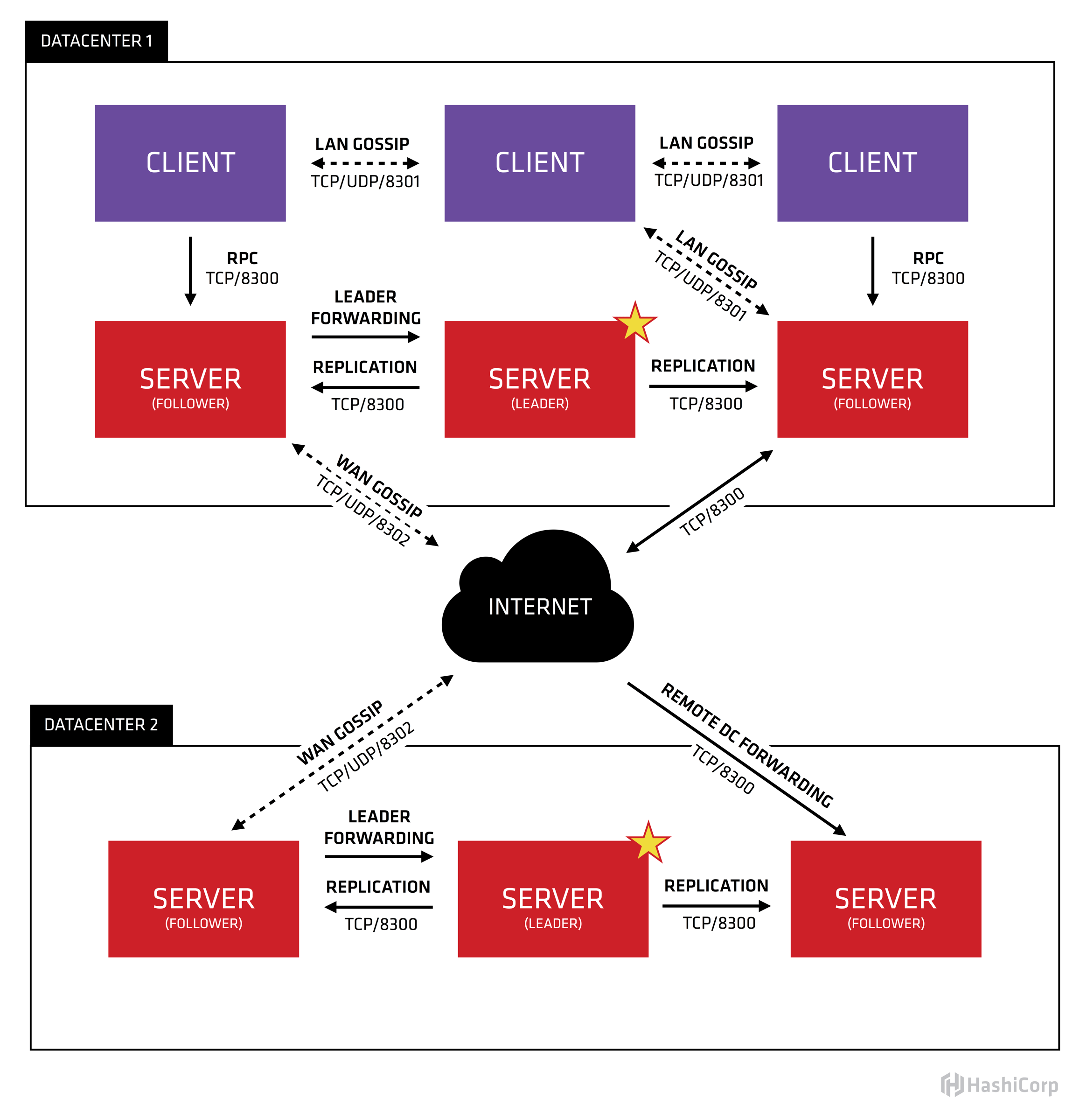
## [»](https://www.consul.io/docs/internals/architecture.html" \l "glossary)Glossary

Before describing the architecture, we provide a glossary of terms to help clarify what is being discussed:

* Agent - An agent is the long running daemon on every member of the Consul cluster. It is started by running consul agent. The agent is able to run in either client or server mode. Since all nodes must be running an agent, it is simpler to refer to the node as being either a client or server, but there are other instances of the agent. All agents can run the DNS or HTTP interfaces, and are responsible for running checks and keeping services in sync.
* Client - A client is an agent that forwards all RPCs to a server. The client is relatively stateless. The only background activity a client performs is taking part in the LAN gossip pool. This has a minimal resource overhead and consumes only a small amount of network bandwidth.
* Server - A server is an agent with an expanded set of responsibilities including participating in the Raft quorum, maintaining cluster state, responding to RPC queries, exchanging WAN gossip with other datacenters, and forwarding queries to leaders or remote datacenters.
* Datacenter - While the definition of a datacenter seems obvious, there are subtle details that must be considered. For example, in EC2, are multiple availability zones considered to comprise a single datacenter? We define a datacenter to be a networking environment that is private, low latency, and high bandwidth. This excludes communication that would traverse the public internet, but for our purposes multiple availability zones within a single EC2 region would be considered part of a single datacenter.
* Consensus - When used in our documentation we use consensus to mean agreement upon the elected leader as well as agreement on the ordering of transactions. Since these transactions are applied to a [finite-state machine](https://en.wikipedia.org/wiki/Finite-state_machine), our definition of consensus implies the consistency of a replicated state machine. Consensus is described in more detail on [Wikipedia](https://en.wikipedia.org/wiki/Consensus_(computer_science)), and our implementation is described [here](https://www.consul.io/docs/internals/consensus.html).
* Gossip - Consul is built on top of [Serf](https://www.serf.io/) which provides a full [gossip protocol](https://en.wikipedia.org/wiki/Gossip_protocol) that is used for multiple purposes. Serf provides membership, failure detection, and event broadcast. Our use of these is described more in the [gossip documentation](https://www.consul.io/docs/internals/gossip.html). It is enough to know that gossip involves random node-to-node communication, primarily over UDP.
* LAN Gossip - Refers to the LAN gossip pool which contains nodes that are all located on the same local area network or datacenter.
* WAN Gossip - Refers to the WAN gossip pool which contains only servers. These servers are primarily located in different datacenters and typically communicate over the internet or wide area network.
* RPC - Remote Procedure Call. This is a request / response mechanism allowing a client to make a request of a server.

## [»](https://www.consul.io/docs/internals/architecture.html" \l "10-000-foot-view)10,000 foot view

From a 10,000 foot altitude the architecture of Consul looks like this:

[](https://www.consul.io/assets/images/consul-arch-420ce04a.png)

Let's break down this image and describe each piece. First of all, we can see that there are two datacenters, labeled "one" and "two". Consul has first class support for [multiple datacenters](https://www.consul.io/docs/guides/datacenters.html) and expects this to be the common case.

Within each datacenter, we have a mixture of clients and servers. It is expected that there be between three to five servers. This strikes a balance between availability in the case of failure and performance, as consensus gets progressively slower as more machines are added. However, there is no limit to the number of clients, and they can easily scale into the thousands or tens of thousands.

All the nodes that are in a datacenter participate in a [gossip protocol](https://www.consul.io/docs/internals/gossip.html). This means there is a gossip pool that contains all the nodes for a given datacenter. This serves a few purposes: first, there is no need to configure clients with the addresses of servers; discovery is done automatically. Second, the work of detecting node failures is not placed on the servers but is distributed. This makes failure detection much more scalable than naive heartbeating schemes. Thirdly, it is used as a messaging layer to notify when important events such as leader election take place.

The servers in each datacenter are all part of a single Raft peer set. This means that they work together to elect a single leader, a selected server which has extra duties. The leader is responsible for processing all queries and transactions. Transactions must also be replicated to all peers as part of the [consensus protocol](https://www.consul.io/docs/internals/consensus.html). Because of this requirement, when a non-leader server receives an RPC request, it forwards it to the cluster leader.

The server nodes also operate as part of a WAN gossip pool. This pool is different from the LAN pool as it is optimized for the higher latency of the internet and is expected to contain only other Consul server nodes. The purpose of this pool is to allow datacenters to discover each other in a low-touch manner. Bringing a new datacenter online is as easy as joining the existing WAN gossip pool. Because the servers are all operating in this pool, it also enables cross-datacenter requests. When a server receives a request for a different datacenter, it forwards it to a random server in the correct datacenter. That server may then forward to the local leader.

This results in a very low coupling between datacenters, but because of failure detection, connection caching and multiplexing, cross-datacenter requests are relatively fast and reliable.

In general, data is not replicated between different Consul datacenters. When a request is made for a resource in another datacenter, the local Consul servers forward an RPC request to the remote Consul servers for that resource and return the results. If the remote datacenter is not available, then those resources will also not be available, but that won't otherwise affect the local datacenter. There are some special situations where a limited subset of data can be replicated, such as with Consul's built-in [ACL replication](https://www.consul.io/docs/guides/acl.html#outages-and-acl-replication) capability, or external tools like [consul-replicate](https://github.com/hashicorp/consul-replicate).

In some places, client agents may cache data from the servers to make it available locally for performance and reliability. Examples include Connect certificates and intentions which allow the client agent to make local decisions about inbound connection requests without a round trip to the servers. Some API endpoints also support optional result caching. This helps reliability because the local agent can continue to respond to some queries like service-discovery or Connect authorization from cache even if the connection to the servers is disrupted or the servers are temporarily unavailable.

## [»](https://www.consul.io/docs/internals/architecture.html" \l "getting-in-depth)Getting in depth

At this point we've covered the high level architecture of Consul, but there are many more details for each of the subsystems. The [consensus protocol](https://www.consul.io/docs/internals/consensus.html) is documented in detail as is the [gossip protocol](https://www.consul.io/docs/internals/gossip.html). The [documentation](https://www.consul.io/docs/internals/security.html) for the security model and protocols used are also available.

For other details, either consult the code, ask in IRC, or reach out to the mailing list.

# Consensus Protocol

Consul uses a [consensus protocol](https://en.wikipedia.org/wiki/Consensus_(computer_science)) to provide [Consistency (as defined by CAP)](https://en.wikipedia.org/wiki/CAP_theorem). The consensus protocol is based on ["Raft: In search of an Understandable Consensus Algorithm"](https://ramcloud.stanford.edu/wiki/download/attachments/11370504/raft.pdf). For a visual explanation of Raft, see [The Secret Lives of Data](http://thesecretlivesofdata.com/raft).

**Advanced Topic!** This page covers technical details of the internals of Consul. You don't need to know these details to effectively operate and use Consul. These details are documented here for those who wish to learn about them without having to go spelunking through the source code.

## [»](https://www.consul.io/docs/internals/consensus.html" \l "raft-protocol-overview)Raft Protocol Overview

Raft is a consensus algorithm that is based on [Paxos](https://en.wikipedia.org/wiki/Paxos_%28computer_science%29). Compared to Paxos, Raft is designed to have fewer states and a simpler, more understandable algorithm.

There are a few key terms to know when discussing Raft:

* Log - The primary unit of work in a Raft system is a log entry. The problem of consistency can be decomposed into a replicated log. A log is an ordered sequence of entries. We consider the log consistent if all members agree on the entries and their order.
* FSM - [Finite State Machine](https://en.wikipedia.org/wiki/Finite-state_machine). An FSM is a collection of finite states with transitions between them. As new logs are applied, the FSM is allowed to transition between states. Application of the same sequence of logs must result in the same state, meaning behavior must be deterministic.
* Peer set - The peer set is the set of all members participating in log replication. For Consul's purposes, all server nodes are in the peer set of the local datacenter.
* Quorum - A quorum is a majority of members from a peer set: for a set of size n, quorum requires at least (n/2)+1members. For example, if there are 5 members in the peer set, we would need 3 nodes to form a quorum. If a quorum of nodes is unavailable for any reason, the cluster becomes unavailable and no new logs can be committed.
* Committed Entry - An entry is considered committed when it is durably stored on a quorum of nodes. Once an entry is committed it can be applied.
* Leader - At any given time, the peer set elects a single node to be the leader. The leader is responsible for ingesting new log entries, replicating to followers, and managing when an entry is considered committed.

Raft is a complex protocol and will not be covered here in detail (for those who desire a more comprehensive treatment, the full specification is available in this [paper](https://ramcloud.stanford.edu/wiki/download/attachments/11370504/raft.pdf)). We will, however, attempt to provide a high level description which may be useful for building a mental model.

Raft nodes are always in one of three states: follower, candidate, or leader. All nodes initially start out as a follower. In this state, nodes can accept log entries from a leader and cast votes. If no entries are received for some time, nodes self-promote to the candidate state. In the candidate state, nodes request votes from their peers. If a candidate receives a quorum of votes, then it is promoted to a leader. The leader must accept new log entries and replicate to all the other followers. In addition, if stale reads are not acceptable, all queries must also be performed on the leader.

Once a cluster has a leader, it is able to accept new log entries. A client can request that a leader append a new log entry (from Raft's perspective, a log entry is an opaque binary blob). The leader then writes the entry to durable storage and attempts to replicate to a quorum of followers. Once the log entry is considered committed, it can be applied to a finite state machine. The finite state machine is application specific; in Consul's case, we use [MemDB](https://github.com/hashicorp/go-memdb) to maintain cluster state. Consul's writes block until it is both committed and applied. This achieves read after write semantics when used with the [consistent](https://www.consul.io/api/index.html#consistent) mode for queries.

Obviously, it would be undesirable to allow a replicated log to grow in an unbounded fashion. Raft provides a mechanism by which the current state is snapshotted and the log is compacted. Because of the FSM abstraction, restoring the state of the FSM must result in the same state as a replay of old logs. This allows Raft to capture the FSM state at a point in time and then remove all the logs that were used to reach that state. This is performed automatically without user intervention and prevents unbounded disk usage while also minimizing time spent replaying logs. One of the advantages of using MemDB is that it allows Consul to continue accepting new transactions even while old state is being snapshotted, preventing any availability issues.

Consensus is fault-tolerant up to the point where quorum is available. If a quorum of nodes is unavailable, it is impossible to process log entries or reason about peer membership. For example, suppose there are only 2 peers: A and B. The quorum size is also 2, meaning both nodes must agree to commit a log entry. If either A or B fails, it is now impossible to reach quorum. This means the cluster is unable to add or remove a node or to commit any additional log entries. This results in unavailability. At this point, manual intervention would be required to remove either A or B and to restart the remaining node in bootstrap mode.

A Raft cluster of 3 nodes can tolerate a single node failure while a cluster of 5 can tolerate 2 node failures. The recommended configuration is to either run 3 or 5 Consul servers per datacenter. This maximizes availability without greatly sacrificing performance. The [deployment table](https://www.consul.io/docs/internals/consensus.html#deployment_table) below summarizes the potential cluster size options and the fault tolerance of each.

In terms of performance, Raft is comparable to Paxos. Assuming stable leadership, committing a log entry requires a single round trip to half of the cluster. Thus, performance is bound by disk I/O and network latency. Although Consul is not designed to be a high-throughput write system, it should handle on the order of hundreds to thousands of transactions per second depending on network and hardware configuration.

## [»](https://www.consul.io/docs/internals/consensus.html" \l "raft-in-consul)Raft in Consul

Only Consul server nodes participate in Raft and are part of the peer set. All client nodes forward requests to servers. Part of the reason for this design is that, as more members are added to the peer set, the size of the quorum also increases. This introduces performance problems as you may be waiting for hundreds of machines to agree on an entry instead of a handful.

When getting started, a single Consul server is put into "bootstrap" mode. This mode allows it to self-elect as a leader. Once a leader is elected, other servers can be added to the peer set in a way that preserves consistency and safety. Eventually, once the first few servers are added, bootstrap mode can be disabled. See [this guide](https://www.consul.io/docs/guides/bootstrapping.html) for more details.

Since all servers participate as part of the peer set, they all know the current leader. When an RPC request arrives at a non-leader server, the request is forwarded to the leader. If the RPC is a query type, meaning it is read-only, the leader generates the result based on the current state of the FSM. If the RPC is a transaction type, meaning it modifies state, the leader generates a new log entry and applies it using Raft. Once the log entry is committed and applied to the FSM, the transaction is complete.

Because of the nature of Raft's replication, performance is sensitive to network latency. For this reason, each datacenter elects an independent leader and maintains a disjoint peer set. Data is partitioned by datacenter, so each leader is responsible only for data in their datacenter. When a request is received for a remote datacenter, the request is forwarded to the correct leader. This design allows for lower latency transactions and higher availability without sacrificing consistency.

## [»](https://www.consul.io/docs/internals/consensus.html" \l "consistency-modes)Consistency Modes

Although all writes to the replicated log go through Raft, reads are more flexible. To support various trade-offs that developers may want, Consul supports 3 different consistency modes for reads.

The three read modes are:

* [default](https://www.consul.io/docs/internals/consensus.html" \l "default) - Raft makes use of leader leasing, providing a time window in which the leader assumes its role is stable. However, if a leader is partitioned from the remaining peers, a new leader may be elected while the old leader is holding the lease. This means there are 2 leader nodes. There is no risk of a split-brain since the old leader will be unable to commit new logs. However, if the old leader services any reads, the values are potentially stale. The default consistency mode relies only on leader leasing, exposing clients to potentially stale values. We make this trade-off because reads are fast, usually strongly consistent, and only stale in a hard-to-trigger situation. The time window of stale reads is also bounded since the leader will step down due to the partition.

* [consistent](https://www.consul.io/docs/internals/consensus.html" \l "consistent) - This mode is strongly consistent without caveats. It requires that a leader verify with a quorum of peers that it is still leader. This introduces an additional round-trip to all server nodes. The trade-off is always consistent reads but increased latency due to the extra round trip.

* [stale](https://www.consul.io/docs/internals/consensus.html" \l "stale) - This mode allows any server to service the read regardless of whether it is the leader. This means reads can be arbitrarily stale but are generally within 50 milliseconds of the leader. The trade-off is very fast and scalable reads but with stale values. This mode allows reads without a leader meaning a cluster that is unavailable will still be able to respond.

For more documentation about using these various modes, see the [HTTP API](https://www.consul.io/api/index.html).

## [»](https://www.consul.io/docs/internals/consensus.html" \l "deployment-table)Deployment Table

Below is a table that shows quorum size and failure tolerance for various cluster sizes. The recommended deployment is either 3 or 5 servers. A single server deployment is ***highly*** discouraged as data loss is inevitable in a failure scenario.

|  |  |  |
| --- | --- | --- |
| **Servers** | **Quorum Size** | **Failure Tolerance** |
| 1 | 1 | 0 |
| 2 | 2 | 0 |
| 3 | 2 | 1 |
| 4 | 3 | 1 |
| 5 | 3 | 2 |
| 6 | 4 | 2 |
| 7 | 4 | 3 |

# Gossip Protocol

Consul uses a [gossip protocol](https://en.wikipedia.org/wiki/Gossip_protocol) to manage membership and broadcast messages to the cluster. All of this is provided through the use of the [Serf library](https://www.serf.io/). The gossip protocol used by Serf is based on ["SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol"](http://www.cs.cornell.edu/info/projects/spinglass/public_pdfs/swim.pdf), with a few minor adaptations. There are more details about [Serf's protocol here](https://www.serf.io/docs/internals/gossip.html).

**Advanced Topic!** This page covers technical details of the internals of Consul. You don't need to know these details to effectively operate and use Consul. These details are documented here for those who wish to learn about them without having to go spelunking through the source code.

## [»](https://www.consul.io/docs/internals/gossip.html" \l "gossip-in-consul)Gossip in Consul

Consul makes use of two different gossip pools. We refer to each pool as the LAN or WAN pool respectively. Each datacenter Consul operates in has a LAN gossip pool containing all members of the datacenter, both clients and servers. The LAN pool is used for a few purposes. Membership information allows clients to automatically discover servers, reducing the amount of configuration needed. The distributed failure detection allows the work of failure detection to be shared by the entire cluster instead of concentrated on a few servers. Lastly, the gossip pool allows for reliable and fast event broadcasts for events like leader election.

The WAN pool is globally unique, as all servers should participate in the WAN pool regardless of datacenter. Membership information provided by the WAN pool allows servers to perform cross datacenter requests. The integrated failure detection allows Consul to gracefully handle an entire datacenter losing connectivity, or just a single server in a remote datacenter.

All of these features are provided by leveraging [Serf](https://www.serf.io/). It is used as an embedded library to provide these features. From a user perspective, this is not important, since the abstraction should be masked by Consul. It can be useful however as a developer to understand how this library is leveraged.

## [»](https://www.consul.io/docs/internals/gossip.html" \l "lifeguard-enhancements)Lifeguard Enhancements

SWIM makes the assumption that the local node is healthy in the sense that soft real-time processing of packets is possible. However, in cases where the local node is experiencing CPU or network exhaustion this assumption can be violated. The result is that the serfHealth check status can occasionally flap, resulting in false monitoring alarms, adding noise to telemetry, and simply causing the overall cluster to waste CPU and network resources diagnosing a failure that may not truly exist.

Lifeguard completely resolves this issue with novel enhancements to SWIM.

For more details about Lifeguard, please see the [Making Gossip More Robust with Lifeguard](https://www.hashicorp.com/blog/making-gossip-more-robust-with-lifeguard/) blog post, which provides a high level overview of the HashiCorp Research paper [Lifeguard : SWIM-ing with Situational Awareness](https://arxiv.org/abs/1707.00788). The [Serf gossip protocol guide](https://www.serf.io/docs/internals/gossip.html#lifeguard) also provides some lower-level details about the gossip protocol and Lifeguard.

# Network Coordinates

Consul uses a [network tomography](https://en.wikipedia.org/wiki/Network_tomography) system to compute network coordinates for nodes in the cluster. These coordinates allow the network round trip time to be estimated between any two nodes using a very simple calculation. This allows for many useful applications, such as finding the service node nearest a requesting node, or failing over to services in the next closest datacenter.

All of this is provided through the use of the [Serf library](https://www.serf.io/). Serf's network tomography is based on ["Vivaldi: A Decentralized Network Coordinate System"](http://www.cs.ucsb.edu/~ravenben/classes/276/papers/vivaldi-sigcomm04.pdf), with some enhancements based on other research. There are more details about [Serf's network coordinates here](https://www.serf.io/docs/internals/coordinates.html).

**Advanced Topic!** This page covers the technical details of the internals of Consul. You don't need to know these details to effectively operate and use Consul. These details are documented here for those who wish to learn about them without having to go spelunking through the source code.

## [»](https://www.consul.io/docs/internals/coordinates.html" \l "network-coordinates-in-consul)Network Coordinates in Consul

Network coordinates manifest in several ways inside Consul:

* The [consul rtt](https://www.consul.io/docs/commands/rtt.html) command can be used to query for the network round trip time between any two nodes.
* The [Catalog endpoints](https://www.consul.io/api/catalog.html) and [Health endpoints](https://www.consul.io/api/health.html) can sort the results of queries based on the network round trip time from a given node using a "?near=" parameter.
* [Prepared queries](https://www.consul.io/api/query.html) can automatically fail over services to other Consul datacenters based on network round trip times. See the [Geo Failover](https://www.consul.io/docs/guides/geo-failover.html) for some examples.
* The [Coordinate endpoint](https://www.consul.io/api/coordinate.html) exposes raw network coordinates for use in other applications.

Consul uses Serf to manage two different gossip pools, one for the LAN with members of a given datacenter, and one for the WAN which is made up of just the Consul servers in all datacenters. It's important to note that **network coordinates are not compatible between these two pools**. LAN coordinates only make sense in calculations with other LAN coordinates, and WAN coordinates only make sense with other WAN coordinates.

## [»](https://www.consul.io/docs/internals/coordinates.html" \l "working-with-coordinates)Working with Coordinates

Computing the estimated network round trip time between any two nodes is simple once you have their coordinates. Here's a sample coordinate, as returned from the [Coordinate endpoint](https://www.consul.io/api/coordinate.html).

"Coord": {

"Adjustment": 0.1,

"Error": 1.5,

"Height": 0.02,

"Vec": [0.34,0.68,0.003,0.01,0.05,0.1,0.34,0.06]

}

All values are floating point numbers in units of seconds, except for the error term which isn't used for distance calculations.

Here's a complete example in Go showing how to compute the distance between two coordinates:

import (

"math"

"time"

"github.com/hashicorp/serf/coordinate"

)

func dist(a \*coordinate.Coordinate, b \*coordinate.Coordinate) time.Duration {

// Coordinates will always have the same dimensionality, so this is

// just a sanity check.

if len(a.Vec) != len(b.Vec) {

panic("dimensions aren't compatible")

}

// Calculate the Euclidean distance plus the heights.

sumsq := 0.0

for i := 0; i < len(a.Vec); i++ {

diff := a.Vec[i] - b.Vec[i]

sumsq += diff \* diff

}

rtt := math.Sqrt(sumsq) + a.Height + b.Height

// Apply the adjustment components, guarding against negatives.

adjusted := rtt + a.Adjustment + b.Adjustment

if adjusted > 0.0 {

rtt = adjusted

}

// Go's times are natively nanoseconds, so we convert from seconds.

const secondsToNanoseconds = 1.0e9

return time.Duration(rtt \* secondsToNanoseconds)

}

# Sessions

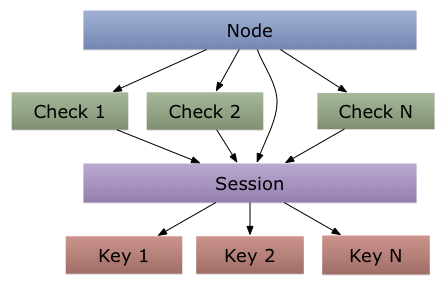
Consul provides a session mechanism which can be used to build distributed locks. Sessions act as a binding layer between nodes, health checks, and key/value data. They are designed to provide granular locking and are heavily inspired by [The Chubby Lock Service for Loosely-Coupled Distributed Systems](http://research.google.com/archive/chubby.html).

**Advanced Topic!** This page covers technical details of the internals of Consul. You don't need to know these details to effectively operate and use Consul. These details are documented here for those who wish to learn about them without having to go spelunking through the source code.

## [»](https://www.consul.io/docs/internals/sessions.html" \l "session-design)Session Design

A session in Consul represents a contract that has very specific semantics. When a session is constructed, a node name, a list of health checks, a behavior, a TTL, and a lock-delay may be provided. The newly constructed session is provided with a named ID that can be used to identify it. This ID can be used with the KV store to acquire locks: advisory mechanisms for mutual exclusion.

Below is a diagram showing the relationship between these components:



The contract that Consul provides is that under any of the following situations, the session will be invalidated:

* Node is deregistered
* Any of the health checks are deregistered
* Any of the health checks go to the critical state
* Session is explicitly destroyed
* TTL expires, if applicable

When a session is invalidated, it is destroyed and can no longer be used. What happens to the associated locks depends on the behavior specified at creation time. Consul supports a release and delete behavior. The release behavior is the default if none is specified.

If the release behavior is being used, any of the locks held in association with the session are released, and the ModifyIndex of the key is incremented. Alternatively, if the delete behavior is used, the key corresponding to any of the held locks is simply deleted. This can be used to create ephemeral entries that are automatically deleted by Consul.

While this is a simple design, it enables a multitude of usage patterns. By default, the [gossip based failure detector](https://www.consul.io/docs/internals/gossip.html) is used as the associated health check. This failure detector allows Consul to detect when a node that is holding a lock has failed and to automatically release the lock. This ability provides **liveness** to Consul locks; that is, under failure the system can continue to make progress. However, because there is no perfect failure detector, it's possible to have a false positive (failure detected) which causes the lock to be released even though the lock owner is still alive. This means we are sacrificing some **safety**.

Conversely, it is possible to create a session with no associated health checks. This removes the possibility of a false positive and trades liveness for safety. You can be absolutely certain Consul will not release the lock even if the existing owner has failed. Since Consul APIs allow a session to be force destroyed, this allows systems to be built that require an operator to intervene in the case of a failure while precluding the possibility of a split-brain.

A third health checking mechanism is session TTLs. When creating a session, a TTL can be specified. If the TTL interval expires without being renewed, the session has expired and an invalidation is triggered. This type of failure detector is also known as a heartbeat failure detector. It is less scalable than the gossip based failure detector as it places an increased burden on the servers but may be applicable in some cases. The contract of a TTL is that it represents a lower bound for invalidation; that is, Consul will not expire the session before the TTL is reached, but it is allowed to delay the expiration past the TTL. The TTL is renewed on session creation, on session renew, and on leader failover. When a TTL is being used, clients should be aware of clock skew issues: namely, time may not progress at the same rate on the client as on the Consul servers. It is best to set conservative TTL values and to renew in advance of the TTL to account for network delay and time skew.

The final nuance is that sessions may provide a lock-delay. This is a time duration, between 0 and 60 seconds. When a session invalidation takes place, Consul prevents any of the previously held locks from being re-acquired for the lock-delay interval; this is a safeguard inspired by Google's Chubby. The purpose of this delay is to allow the potentially still live leader to detect the invalidation and stop processing requests that may lead to inconsistent state. While not a bulletproof method, it does avoid the need to introduce sleep states into application logic and can help mitigate many issues. While the default is to use a 15 second delay, clients are able to disable this mechanism by providing a zero delay value.

## [»](https://www.consul.io/docs/internals/sessions.html" \l "k-v-integration)K/V Integration

Integration between the KV store and sessions is the primary place where sessions are used. A session must be created prior to use and is then referred to by its ID.

The KV API is extended to support an acquire and release operation. The acquire operation acts like a Check-And-Set operation except it can only succeed if there is no existing lock holder (the current lock holder can re-acquire, see below). On success, there is a normal key update, but there is also an increment to the LockIndex, and the Session value is updated to reflect the session holding the lock.

If the lock is already held by the given session during an acquire, then the LockIndex is not incremented but the key contents are updated. This lets the current lock holder update the key contents without having to give up the lock and reacquire it.

Once held, the lock can be released using a corresponding release operation, providing the same session. Again, this acts like a Check-And-Set operation since the request will fail if given an invalid session. A critical note is that the lock can be released without being the creator of the session. This is by design as it allows operators to intervene and force-terminate a session if necessary. As mentioned above, a session invalidation will also cause all held locks to be released or deleted. When a lock is released, the LockIndex does not change; however, the Session is cleared and the ModifyIndex increments.

These semantics (heavily borrowed from Chubby), allow the tuple of (Key, LockIndex, Session) to act as a unique "sequencer". This sequencer can be passed around and used to verify if the request belongs to the current lock holder. Because the LockIndex is incremented on each acquire, even if the same session re-acquires a lock, the sequencer will be able to detect a stale request. Similarly, if a session is invalided, the Session corresponding to the given LockIndex will be blank.

To be clear, this locking system is purely advisory. There is no enforcement that clients must acquire a lock to perform any operation. Any client can read, write, and delete a key without owning the corresponding lock. It is not the goal of Consul to protect against misbehaving clients.

## [»](https://www.consul.io/docs/internals/sessions.html" \l "leader-election)Leader Election

The primitives provided by sessions and the locking mechanisms of the KV store can be used to build client-side leader election algorithms. These are covered in more detail in the [Leader Election guide](https://www.consul.io/docs/guides/leader-election.html).

## [»](https://www.consul.io/docs/internals/sessions.html" \l "prepared-query-integration)Prepared Query Integration

Prepared queries may be attached to a session in order to automatically delete the prepared query when the session is invalidated.

# Anti-Entropy

Consul uses an advanced method of maintaining service and health information. This page details how services and checks are registered, how the catalog is populated, and how health status information is updated as it changes.

**Advanced Topic!** This page covers technical details of the internals of Consul. You don't need to know these details to effectively operate and use Consul. These details are documented here for those who wish to learn about them without having to go spelunking through the source code.

### [»](https://www.consul.io/docs/internals/anti-entropy.html" \l "components)Components

It is important to first understand the moving pieces involved in services and health checks: the [agent](https://www.consul.io/docs/internals/anti-entropy.html#agent) and the [catalog](https://www.consul.io/docs/internals/anti-entropy.html#catalog). These are described conceptually below to make anti-entropy easier to understand.

#### [»](https://www.consul.io/docs/internals/anti-entropy.html" \l "agent)Agent

Each Consul agent maintains its own set of service and check registrations as well as health information. The agents are responsible for executing their own health checks and updating their local state.

Services and checks within the context of an agent have a rich set of configuration options available. This is because the agent is responsible for generating information about its services and their health through the use of [health checks](https://www.consul.io/docs/agent/checks.html).

#### [»](https://www.consul.io/docs/internals/anti-entropy.html" \l "catalog)Catalog

Consul's service discovery is backed by a service catalog. This catalog is formed by aggregating information submitted by the agents. The catalog maintains the high-level view of the cluster, including which services are available, which nodes run those services, health information, and more. The catalog is used to expose this information via the various interfaces Consul provides, including DNS and HTTP.

Services and checks within the context of the catalog have a much more limited set of fields when compared with the agent. This is because the catalog is only responsible for recording and returning information about services, nodes, and health.

The catalog is maintained only by server nodes. This is because the catalog is replicated via the [Raft log](https://www.consul.io/docs/internals/consensus.html) to provide a consolidated and consistent view of the cluster.

### [»](https://www.consul.io/docs/internals/anti-entropy.html" \l "anti-entropy-1)Anti-Entropy

Entropy is the tendency of systems to become increasingly disordered. Consul's anti-entropy mechanisms are designed to counter this tendency, to keep the state of the cluster ordered even through failures of its components.

Consul has a clear separation between the global service catalog and the agent's local state as discussed above. The anti-entropy mechanism reconciles these two views of the world: anti-entropy is a synchronization of the local agent state and the catalog. For example, when a user registers a new service or check with the agent, the agent in turn notifies the catalog that this new check exists. Similarly, when a check is deleted from the agent, it is consequently removed from the catalog as well.

Anti-entropy is also used to update availability information. As agents run their health checks, their status may change in which case their new status is synced to the catalog. Using this information, the catalog can respond intelligently to queries about its nodes and services based on their availability.

During this synchronization, the catalog is also checked for correctness. If any services or checks exist in the catalog that the agent is not aware of, they will be automatically removed to make the catalog reflect the proper set of services and health information for that agent. Consul treats the state of the agent as authoritative; if there are any differences between the agent and catalog view, the agent-local view will always be used.

### [»](https://www.consul.io/docs/internals/anti-entropy.html" \l "periodic-synchronization)Periodic Synchronization

In addition to running when changes to the agent occur, anti-entropy is also a long-running process which periodically wakes up to sync service and check status to the catalog. This ensures that the catalog closely matches the agent's true state. This also allows Consul to re-populate the service catalog even in the case of complete data loss.

To avoid saturation, the amount of time between periodic anti-entropy runs will vary based on cluster size. The table below defines the relationship between cluster size and sync interval:

|  |  |
| --- | --- |
| **Cluster Size** | **Periodic Sync Interval** |
| 1 - 128 | 1 minute |
| 129 - 256 | 2 minutes |
| 257 - 512 | 3 minutes |
| 513 - 1024 | 4 minutes |
| ... | ... |

The intervals above are approximate. Each Consul agent will choose a randomly staggered start time within the interval window to avoid a thundering herd.

### [»](https://www.consul.io/docs/internals/anti-entropy.html" \l "best-effort-sync)Best-effort sync

Anti-entropy can fail in a number of cases, including misconfiguration of the agent or its operating environment, I/O problems (full disk, filesystem permission, etc.), networking problems (agent cannot communicate with server), among others. Because of this, the agent attempts to sync in best-effort fashion.

If an error is encountered during an anti-entropy run, the error is logged and the agent continues to run. The anti-entropy mechanism is run periodically to automatically recover from these types of transient failures.

### [»](https://www.consul.io/docs/internals/anti-entropy.html" \l "enable-tag-override)Enable Tag Override

Synchronization of service registration can be partially modified to allow external agents to change the tags for a service. This can be useful in situations where an external monitoring service needs to be the source of truth for tag information. For example, the Redis database and its monitoring service Redis Sentinel have this kind of relationship. Redis instances are responsible for much of their configuration, but Sentinels determine whether the Redis instance is a primary or a secondary. Using the Consul service configuration item [enable\_tag\_override](https://www.consul.io/docs/agent/services.html) you can instruct the Consul agent on which the Redis database is running to NOT update the tags during anti-entropy synchronization. For more information see[Services](https://www.consul.io/docs/agent/services.html#enable-tag-override-and-anti-entropy) page.

# Security Model

Consul relies on both a lightweight gossip mechanism and an RPC system to provide various features. Both of the systems have different security mechanisms that stem from their designs. However, the security mechanisms of Consul have a common goal: to provide [confidentiality, integrity, and authentication](https://en.wikipedia.org/wiki/Information_security).

The [gossip protocol](https://www.consul.io/docs/internals/gossip.html) is powered by [Serf](https://www.serf.io/), which uses a symmetric key, or shared secret, cryptosystem. There are more details on the security of [Serf here](https://www.serf.io/docs/internals/security.html). For details on how to enable Serf's gossip encryption in Consul, see the [encryption doc here](https://www.consul.io/docs/agent/encryption.html).

The RPC system supports using end-to-end TLS with optional client authentication. [TLS](https://en.wikipedia.org/wiki/Transport_Layer_Security) is a widely deployed asymmetric cryptosystem and is the foundation of security on the Web.

This means Consul communication is protected against eavesdropping, tampering, and spoofing. This makes it possible to run Consul over untrusted networks such as EC2 and other shared hosting providers.

**Advanced Topic!** This page covers the technical details of the security model of Consul. You don't need to know these details to operate and use Consul. These details are documented here for those who wish to learn about them without having to go spelunking through the source code.

## [»](https://www.consul.io/docs/internals/security.html" \l "secure-configuration)Secure Configuration

The Consul threat model is only applicable if Consul is running in a secure configuration. Consul does not operate in a secure-by-default configuration. If any of the settings below are not enabled, then parts of this threat model are going to be invalid. Additional security precautions must also be taken for items outside of Consul's threat model as noted in sections below.

* **ACLs enabled with default deny.** Consul must be configured to use ACLs with a whitelist (default deny) approach. This forces all requests to have explicit anonymous access or provide an ACL token.
* **Encryption enabled.** TCP and UDP encryption must be enabled and configured to prevent plaintext communication between Consul agents. At a minimum, verify\_outgoing should be enabled to verify server authenticity with each server having a unique TLS certificate. verify\_server\_hostname is also required to prevent a compromised agent restarting as a server and being given access to all secrets.

verify\_incoming provides additional agent verification via mutual authentication, but isn't strictly necessary to enforce the threat model since requests must also contain a valid ACL token. The subtlety is that currently verify\_incoming = false will allow servers to still accept un-encrypted connections from clients (to allow for gradual TLS rollout). That alone doesn't violate the threat model, but any misconfigured client that chooses not to use TLS will violate the model. We recommend setting this to true. If it is left as false care must be taken to ensure all consul clients use verify\_outgoing = true as noted above, but also all external API/UI access must be via HTTPS with HTTP listeners disabled.

### [»](https://www.consul.io/docs/internals/security.html" \l "known-insecure-configurations)Known Insecure Configurations

In addition to configuring the non-default settings above, Consul has several non-default options that potentially present additional security risks.

* **Script checks enabled with network-exposed API.** If a Consul agent (client or server) exposes its HTTP API to the network beyond localhost, [enable\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_script_checks) must be false otherwise, even with ACLs configured, script checks present a remote code execution threat. [enable\_local\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_local_script_checks) provides a secure alterative if the HTTP API must be exposed and is available from 1.3.0 on. This feature was also back-ported to patch releases 0.9.4, 1.1.1, and 1.2.4 [as described here](https://www.hashicorp.com/blog/protecting-consul-from-rce-risk-in-specific-configurations).
* **Remote exec enabled.** Consul includes a [consul exec feature](https://www.consul.io/docs/commands/exec.html) allowing execution of arbitrary commands across the cluster. This is disabled by default since 0.8.0. We recommend leaving it disabled. If enabled, extreme care must be taken to ensure correct ACLs restrict access, for example any management token grants access to execute arbitrary code on the cluster.
* **Verify Server Hostname Used Alone.** From version 0.5.1 to 1.4.0 we documented that verify\_server\_hostnamebeing true implied verify\_outgoing however due to a bug this was not the case so setting onlyverify\_server\_hostname results in plaintext communciation between client and server. See [CVE-2018-19653](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-19653) for more details. This is fixed in 1.4.1.

## [»](https://www.consul.io/docs/internals/security.html" \l "threat-model)Threat Model

The following are parts of the Consul threat model:

* **Consul agent-to-agent communication.** Communication between Consul agents should be secure from eavesdropping. This requires transport encryption to be enabled on the cluster and covers both TCP and UDP traffic.
* **Consul agent-to-CA communication.** Communication between the Consul server and the configured certificate authority provider for Connect is always encrypted.
* **Tampering of data in transit.** Any tampering should be detectable and cause Consul to avoid processing the request.
* **Access to data without authentication or authorization.** All requests must be authenticated and authorized. This requires that ACLs are enabled on the cluster with a default deny mode.
* **State modification or corruption due to malicious messages.** Ill-formatted messages are discarded and well-formatted messages require authentication and authorization.
* **Non-server members accessing raw data.** All servers must join the cluster (with proper authentication and authorization) to begin participating in Raft. Raft data is transmitted over TLS.
* **Denial of Service against a node.** DoS attacks against a node should not compromise the security stance of the software.
* **Connect-based Service-to-Service communication.** Communications between two Connect-enabled services (natively or by proxy) should be secure from eavesdropping and provide authentication. This is achieved via mutual TLS.

The following are not part of the Consul threat model for Consul server agents:

* **Access (read or write) to the Consul data directory.** All Consul servers, including non-leaders, persist the full set of Consul state to this directory. The data includes all KV, service registrations, ACL tokens, Connect CA configuration, and more. Any read or write to this directory allows an attacker to access and tamper with that data.
* **Access (read or write) to the Consul configuration directory.** Consul configuration can enable or disable the ACL system, modify data directory paths, and more. Any read or write of this directory allows an attacker to reconfigure many aspects of Consul. By disabling the ACL system, this may give an attacker access to all Consul data.
* **Memory access to a running Consul server agent.** If an attacker is able to inspect the memory state of a running Consul server agent the confidentiality of almost all Consul data may be compromised. If you're using an external Connect CA, the root private key material is never available to the Consul process and can be considered safe. Service Connect TLS certificates should be considered compromised; they are never persisted by server agents but do exist in-memory during at least the duration of a Sign request.

The following are not part of the Consul threat model for Consul client agents:

* **Access (read or write) to the Consul data directory.** Consul clients will use the data directory to cache local state. This includes local services, associated ACL tokens, Connect TLS certificates, and more. Read or write access to this directory will allow an attacker to access this data. This data is typically a smaller subset of the full data of the cluster.
* **Access (read or write) to the Consul configuration directory.** Consul client configuration files contain the address and port information of services, default ACL tokens for the agent, and more. Access to Consul configuration could enable an attacker to change the port of a service to a malicious port, register new services, and more. Further, some service definitions have ACL tokens attached that could be used cluster-wide to impersonate that service. An attacker cannot change cluster-wide configurations such as disabling the ACL system.
* **Memory access to a running Consul client agent.** The blast radius of this is much smaller than a server agent but the confidentiality of a subset of data can still be compromised. Particularly, any data requested against the agent's API including services, KV, and Connect information may be compromised. If a particular set of data on the server was never requested by the agent, it never enters the agent's memory since replication only exists between servers. An attacker could also potentially extract ACL tokens used for service registration on this agent, since the tokens must be stored in-memory alongside the registered service.
* **Network access to a local Connect proxy or service.** Communications between a service and a Connect-aware proxy are generally unencrypted and must happen over a trusted network. This is typically a loopback device. This requires that other processes on the same machine are trusted, or more complex isolation mechanisms are used such as network namespaces. This also requires that external processes cannot communicate to the Connect service or proxy (except on the inbound port). Therefore, non-native Connect applications should only bind to non-public addresses.
* **Improperly Implemented Connect proxy or service.** A Connect proxy or natively integrated service must correctly serve a valid leaf certificate, verify the inbound TLS client certificate, and call the Consul agent-local authorize endpoint. If any of this isn't performed correctly, the proxy or service may allow unauthenticated or unauthorized connections.

## [»](https://www.consul.io/docs/internals/security.html" \l "external-threat-overview)External Threat Overview

There are four components that affect the Consul threat model: the server agent, the client agent, the Connect CA, and Consul API clients (including proxies for Connect).

The server agent participates in leader election and data replication via Raft. All communications with other agents is encrypted. Data is stored at rest unencrypted in the configured data directory. The stored data includes ACL tokens and TLS certificates. If the built-in CA is used with Connect, root certificate private keys are also stored on disk. External CA providers do not store data in this directory. This data directory must be carefully protected to prevent an attacker from impersonating a server or specific ACL user. We plan to introduce further mitigations (including at least partial data encryption) to the data directory over time, but the data directory should always be considered secret.

For a client agent to join a cluster, it must provide a valid ACL token with node:write capabilities. The join request and all other API requests between the client and server agents communicate via TLS. Clients serve the Consul API and forward all requests to a server over a shared TLS connection. Each request contains an ACL token which is used for both authentication and authorization. Requests that do not provide an ACL token inherit the agent-configurable default ACL token.

The Connect CA provider is responsible for storing the private key of the root (or intermediate) certificate used to sign and verify connections established via Connect. Consul server agents communicate with the CA provider via an encrypted method. This method is dependent on the CA provider in use. Consul provides a built-in CA which performs all operations locally on the server agent. Consul itself does not store any private key material except for the built-in CA.

Consul API clients (the agent itself, the built-in UI, external software) must communicate to a Consul agent over TLS and must provide an ACL token per request for authentication and authorization.

## [»](https://www.consul.io/docs/internals/security.html" \l "network-ports)Network Ports

For configuring network rules to support Consul, please see [Ports Used](https://www.consul.io/docs/agent/options.html#ports) for a listing of network ports used by Consul and details about which features they are used for.

# Jepsen Testing

[Jepsen](http://aphyr.com/posts/281-call-me-maybe-carly-rae-jepsen-and-the-perils-of-network-partitions) is a tool, written by Kyle Kingsbury, designed to test the partition tolerance of distributed systems. It creates network partitions while fuzzing the system with random operations. The results are analyzed to see if the system violates any of the consistency properties it claims to have.

As part of our Consul testing, we ran a Jepsen test to determine if any consistency issues could be uncovered. In our testing, Consul gracefully recovered from partitions without introducing any consistency issues.

## [»](https://www.consul.io/docs/internals/jepsen.html" \l "running-the-tests)Running the tests

At the moment, testing with Jepsen is rather complex as it requires setting up multiple virtual machines, SSH keys, DNS configuration, and a working Clojure environment. We hope to contribute our Consul testing code upstream and to provide a Vagrant environment for Jepsen testing soon.

## [»](https://www.consul.io/docs/internals/jepsen.html" \l "output)Output

Below is the output captured from Jepsen. We ran Jepsen multiple times, and it passed each time. This output is only representative of a single run.

略

# Consul Commands (CLI)

Consul is controlled via a very easy to use command-line interface (CLI). Consul is only a single command-line application: consul. This application then takes a subcommand such as "agent" or "members". The complete list of subcommands is in the navigation to the left.

The consul CLI is a well-behaved command line application. In erroneous cases, a non-zero exit status will be returned. It also responds to -h and --help as you'd most likely expect. And some commands that expect input accept "-" as a parameter to tell Consul to read the input from stdin.

To view a list of the available commands at any time, just run consul with no arguments:

$ consul

Usage: consul [--version] [--help] <command> [<args>]

Available commands are:

agent Runs a Consul agent

catalog Interact with the catalog

connect Interact with Consul Connect

event Fire a new event

exec Executes a command on Consul nodes

force-leave Forces a member of the cluster to enter the "left" state

info Provides debugging information for operators.

intention Interact with Connect service intentions

join Tell Consul agent to join cluster

keygen Generates a new encryption key

keyring Manages gossip layer encryption keys

kv Interact with the key-value store

leave Gracefully leaves the Consul cluster and shuts down

lock Execute a command holding a lock

maint Controls node or service maintenance mode

members Lists the members of a Consul cluster

monitor Stream logs from a Consul agent

operator Provides cluster-level tools for Consul operators

reload Triggers the agent to reload configuration files

rtt Estimates network round trip time between nodes

services Interact with services

snapshot Saves, restores and inspects snapshots of Consul server state

validate Validate config files/directories

version Prints the Consul version

watch Watch for changes in Consul

To get help for any specific command, pass the -h flag to the relevant subcommand. For example, to see help about the join subcommand:

$ consul join -h

Usage: consul join [options] address ...

Tells a running Consul agent (with "consul agent") to join the cluster

by specifying at least one existing member.

HTTP API Options

-http-addr=<address>

The `address` and port of the Consul HTTP agent. The value can be

an IP address or DNS address, but it must also include the port.

This can also be specified via the CONSUL\_HTTP\_ADDR environment

variable. The default value is http://127.0.0.1:8500. The scheme

can also be set to HTTPS by setting the environment variable

CONSUL\_HTTP\_SSL=true.

-token=<value>

ACL token to use in the request. This can also be specified via the

CONSUL\_HTTP\_TOKEN environment variable. If unspecified, the query

will default to the token of the Consul agent at the HTTP address.

Command Options

-wan

Joins a server to another server in the WAN pool.

## [»](https://www.consul.io/docs/commands/index.html" \l "autocompletion)Autocompletion

The consul command features opt-in subcommand autocompletion that you can enable for your shell with consul -autocomplete-install. After doing so, you can invoke a new shell and use the feature.

For example, assume a tab is typed at the end of each prompt line:

$ consul e

event exec

$ consul r

reload rtt

$ consul operator raft

list-peers remove-peer

## [»](https://www.consul.io/docs/commands/index.html" \l "environment-variables)Environment Variables

In addition to CLI flags, Consul reads environment variables for behavior defaults. CLI flags always take precedence over environment variables, but it is often helpful to use environment variables to configure the Consul agent, particularly with configuration management and init systems.

These environment variables and their purpose are described below:

## [»](https://www.consul.io/docs/commands/index.html" \l "consul_http_addr)CONSUL\_HTTP\_ADDR

This is the HTTP API address to the local Consul agent (not the remote server) specified as a URI with optional scheme:

CONSUL\_HTTP\_ADDR=127.0.0.1:8500

or as a Unix socket path:

CONSUL\_HTTP\_ADDR=unix://var/run/consul\_http.sock

If the https:// scheme is used, CONSUL\_HTTP\_SSL is implied to be true.

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_http_token)CONSUL\_HTTP\_TOKEN

This is the API access token required when access control lists (ACLs) are enabled, for example:

CONSUL\_HTTP\_TOKEN=aba7cbe5-879b-999a-07cc-2efd9ac0ffe

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_http_auth)CONSUL\_HTTP\_AUTH

This specifies HTTP Basic access credentials as a username:password pair:

CONSUL\_HTTP\_AUTH=operations:JPIMCmhDHzTukgO6

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_http_ssl)CONSUL\_HTTP\_SSL

This is a boolean value (default is false) that enables the HTTPS URI scheme and SSL connections to the HTTP API:

CONSUL\_HTTP\_SSL=true

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_http_ssl_verify)CONSUL\_HTTP\_SSL\_VERIFY

This is a boolean value (default true) to specify SSL certificate verification; setting this value to false is not recommended for production use. Example for development purposes:

CONSUL\_HTTP\_SSL\_VERIFY=false

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_cacert)CONSUL\_CACERT

Path to a CA file to use for TLS when communicating with Consul.

CONSUL\_CACERT=ca.crt

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_capath)CONSUL\_CAPATH

Path to a directory of CA certificates to use for TLS when communicating with Consul.

CONSUL\_CAPATH=ca\_certs/

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_client_cert)CONSUL\_CLIENT\_CERT

Path to a client cert file to use for TLS when verify\_incoming is enabled.

CONSUL\_CLIENT\_CERT=client.crt

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_client_key)CONSUL\_CLIENT\_KEY

Path to a client key file to use for TLS when verify\_incoming is enabled.

CONSUL\_CLIENT\_KEY=client.key

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_tls_server_name)CONSUL\_TLS\_SERVER\_NAME

The server name to use as the SNI host when connecting via TLS.

CONSUL\_TLS\_SERVER\_NAME=consulserver.domain

### [»](https://www.consul.io/docs/commands/index.html" \l "consul_grpc_addr)CONSUL\_GRPC\_ADDR

Like [CONSUL\_HTTP\_ADDR](https://www.consul.io/docs/commands/index.html#consul_http_addr) but configures the address the local agent is listening for gRPC requests. Currently gRPC is only used for integrating [Envoy proxy](https://www.consul.io/docs/connect/proxies/envoy.html) and must be [enabled explicitly](https://www.consul.io/docs/agent/options.html#grpc_port) in agent configuration.

CONSUL\_GRPC\_ADDR=127.0.0.1:8502

or as a Unix socket path:

CONSUL\_GRPC\_ADDR=unix://var/run/consul\_grpc.sock

If the agent is [configured with TLS certificates](https://www.consul.io/docs/agent/encryption.html#rpc-encryption-with-tls), then the gRPC listener will require TLS and present the same certificate as the https listener. As with CONSUL\_HTTP\_ADDR, if TLS is enabled either the https:// scheme should be used, or CONSUL\_HTTP\_SSL set.

# Consul ACLs

Command: consul acl

The acl command is used to interact with Consul's ACLs via the command line. It exposes top-level commands for bootstrapping the ACL system, managing tokens and policies, translating legacy rules, and setting the tokens for use by an agent.

ACLs are also accessible via the [HTTP API](https://www.consul.io/api/acl/acl.html).

Bootstrap Consul's ACLs:

$ consul acl bootstrap

AccessorID: 4d123dff-f460-73c3-02c4-8dd64d136e01

SecretID: 86cddfb9-2760-d947-358d-a2811156bf31

Description: Bootstrap Token **(**Global Management**)**

Local: false

Create Time: 2018-10-22 11:27:04.479026 -0400 EDT

Policies:

00000000-0000-0000-0000-000000000001 - global-management

Create a policy:

$ consul acl policy create -name "acl-replication" -description "Token capable of replicating ACL policies" -rules 'acl = "read"'

ID: 35b8ecb0-707c-ee18-2002-81b238b54b38

Name: acl-replication

Description: Token capable of replicating ACL policies

Datacenters:

Rules:

acl **=** "read"

Create a token:

$ consul acl token create -description "Agent Policy Replication - my-agent" -policy-name "acl-replication"

AccessorID: c24c11aa-4e08-e25c-1a67-705a2e8d75a4

SecretID: e7024f9c-f016-02dd-6217-daedbffb86ac

Description: Agent Policy Replication - my-agent

Local: false

Create Time: 2018-10-22 11:34:49.960482 -0400 EDT

Policies:

35b8ecb0-707c-ee18-2002-81b238b54b38 - acl-replication

For more examples, ask for subcommand help or view the subcommand documentation by clicking on one of the links in the sidebar.

## [»](https://www.consul.io/docs/commands/acl.html" \l "usage)Usage

Usage: consul acl <subcommand>

For the exact documentation for your Consul version, run consul acl -h to view the complete list of subcommands.

Usage: consul acl <subcommand> [options] [args]

This command has subcommands for interacting with Consul's ACLs.

Here are some simple examples, and more detailed examples are available

in the subcommands or the documentation.

Bootstrap ACLs:

$ consul acl bootstrap

List all ACL Tokens:

$ consul acl token list

Create a new ACL Policy:

$ consul acl policy create -name "new-policy" \

-description "This is an example policy" \

-datacenter "dc1" \

-datacenter "dc2" \

-rules @rules.hcl

Set the default agent token:

$ consul acl set-agent-token default 0bc6bc46-f25e-4262-b2d9-ffbe1d96be6f

For more examples, ask for subcommand help or view the documentation.

Subcommands:

bootstrap Bootstrap Consul's ACL system

policy Manage Consul's ACL Policies

set-agent-token Interact with the Consul's ACLs

token Manage Consul's ACL Tokens

translate-rules Translate the legacy rule syntax into the current syntax

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar or one of the links below:

# Consul ACL Bootstrap

Command: consul acl bootstrap

The acl bootstrap command will request Consul to generate a new token with unlimited privileges to use for management purposes and output its details. This can only be done once and afterwards bootstrapping will be disabled. If all tokens are lost and you need to bootstrap again you can follow the bootstrap [reset procedure](https://learn.hashicorp.com/consul/advanced/day-1-operations/acl-guide#ensure-the-acl-system-is-configured-properly).

The ACL system can also be bootstrapped via the [HTTP API](https://www.consul.io/api/acl/acl.html#bootstrap-acls).

## [»](https://www.consul.io/docs/commands/acl/acl-bootstrap.html#usage)Usage

Usage: consul acl bootstrap [options]

#### [»](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.

* [-ca-path=<value>](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.

* [-client-cert=<value>](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.

* [-client-key=<value>](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.

* [-http-addr=<addr>](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.

* [-tls-server-name=<value>](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.

* [-token=<value>](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

* [-datacenter=<name>](https://www.consul.io/docs/commands/acl/acl-bootstrap.html" \l "datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/acl/acl-bootstrap.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

The output looks like this:

AccessorID: 4d123dff-f460-73c3-02c4-8dd64d136e01

SecretID: 86cddfb9-2760-d947-358d-a2811156bf31

Description: Bootstrap Token (Global Management)

Local: false

Create Time: 2018-10-22 11:27:04.479026 -0400 EDT

Policies:

00000000-0000-0000-0000-000000000001 - global-management

# Consul ACL Policies

Command: consul acl policy

The acl policy command is used to manage Consul's ACL policies. There are subcommands for the individual operations that can be performed.

* [create](https://www.consul.io/docs/commands/acl/acl-policy.html#create)
* [read](https://www.consul.io/docs/commands/acl/acl-policy.html#read)
* [update](https://www.consul.io/docs/commands/acl/acl-policy.html#update)
* [delete](https://www.consul.io/docs/commands/acl/acl-policy.html#delete)
* [list](https://www.consul.io/docs/commands/acl/acl-policy.html#list)

ACL policies are also accessible via the [HTTP API](https://www.consul.io/api/acl/acl.html).

Usage: consul acl policy <subcommand> [options] [args]

**Note:** All of the examples show for the subcommands will require a valid Consul token with the appropriate permissions. Either set the CONSUL\_HTTP\_TOKEN environment variable to the tokens secret ID or pass the secret ID as the value of the -token parameter.

## [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "identitying-polices)Identitying Polices

In several of the subcommands a policy will have to be identified to be read, modified or deleted. Those subcommands support specifying the policy by its ID using the -id parameter or by name using the -name parameter. When specifying the policy by its ID a unique policy ID prefix may be specified instead of the entire UUID. As long as it is unique it will be resolved to the full UUID and used. Additionally builtin policy names will be accepted as the value to the -id parameter. Even if the builtin policies are renamed their original name can be used to operate on them.

Builtin Policies:

| **Policy UUID** | **Policy Name** |
| --- | --- |
| 00000000-0000-0000-0000-000000000001 | global-management |

## [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "common-subcommand-options)Common Subcommand Options

All of the consul acl policy subcommands support the following options:

* [-ca-file=<value>](https://www.consul.io/docs/commands/acl/acl-policy.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/acl/acl-policy.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/acl/acl-policy.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/acl/acl-policy.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/acl/acl-policy.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/acl/acl-policy.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/acl/acl-policy.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/acl/acl-policy.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/acl/acl-policy.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

## [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "create)create

Command: consul acl policy create

This command creates new policies. The policies rules can either be set explicitly or the -from-token parameter may be used to load the rules from a legacy ACL token. When loading the rules from an existing legacy ACL token, the rules get translated from the legacy syntax to the new syntax.

Both the -rules and -from-token parameter values allow loading the value from stdin, a file or the raw value. To use stdin pass - as the value. To load the value from a file prefix the value with an @. Any other values will be used directly.

**Deprecated:** The -from-token and -token-secret arguments exist only as a convenience to make legacy ACL migration easier. These will be removed in a future major release when support for the legacy ACL system is removed.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html#usage)Usage

Usage: consul acl policy create [options] [args]

#### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "options)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-policy.html#common-subcommand-options)

* [-description=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "description-lt-string-gt-) - A description of the policy.

* [-from-token=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "from-token-lt-string-gt-) - The legacy token to retrieve the rules for when creating this policy. When this is specified no other rules should be given. Similar to the -rules option the token to use can be loaded from stdin or from a file.

* [-meta](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "meta) - Indicates that policy metadata such as the content hash and raft indices should be shown for each entry.

* [-name=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "name-lt-string-gt-) - The new policy's name. This flag is required.

* [-rules=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "rules-lt-string-gt-) - The policy rules. May be prefixed with '@' to indicate that the value is a file path to load the rules from. '-' may also be given to indicate that the rules are available on stdin.

* [-token-secret](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "token-secret) - Indicates the token provided with -from-token is a SecretID and not an AccessorID.

* [-valid-datacenter=<value>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "valid-datacenter-lt-value-gt-) - Datacenter that the policy should be valid within. This flag may be specified multiple times.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "examples)Examples

Create a new policy that is valid in all datacenters:

$ consul acl policy create -name "acl-replication" -description "Policy capable of replicating ACL policies" -rules 'acl = "read"'

ID: 35b8ecb0-707c-ee18-2002-81b238b54b38

Name: acl-replication

Description: Policy capable of replicating ACL policies

Datacenters:

Rules:

acl **=** "read"

Create a new policy valid only in specific datacenters with rules read from a file:

$ consul acl policy create -name "replication" -description "Replication" -rules @rules.hcl -valid-datacenter dc1 -valid-datacenter dc2

ID: ca44555b-a2d8-94de-d763-88caffdaf11f

Name: replication

Description: Replication

Datacenters: dc1, dc2

Rules:

acl **=** "read"

service\_prefix "" **{**

policy **=** "read"

intentions **=** "read"

**}**

Create a new policy with rules equivalent to that of a legacy ACL token:

$ consul acl policy create -name "node-services-read" -from-token 5793a5ce -description "Can read any node and service"

ID: 06acc965-df4b-5a99-58cb-3250930c6324

Name: node-services-read

Description: Can read any node and service

Datacenters:

Rules:

service\_prefix "" **{**

policy **=** "read"

**}**

node\_prefix "" **{**

policy **=** "read"

**}**

## [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "read)read

Command: consul acl policy read

This command reads and displays a policies details.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "usage-1)Usage

Usage: consul acl policy read [options] [args]

#### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "options-1)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-policy.html#common-subcommand-options)

* [-id=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "id-lt-string-gt-) - The ID of the policy to read. It may be specified as a unique ID prefix but will error if the prefix matches multiple policy IDs.

* [-meta](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "meta-1) - Indicates that policy metadata such as the content hash and raft indices should be shown for each entry.

* [-name=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "name-lt-string-gt--1) - The name of the policy to read.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "examples-1)Examples

Get policy details:

$ consul acl policy read -id 00000000-0000-0000-0000-000000000001

ID: 00000000-0000-0000-0000-000000000001

Name: global-management

Description: Builtin Policy that grants unlimited access

Datacenters:

Rules:

acl **=** "write"

agent\_prefix "" **{**

policy **=** "write"

**}**

event\_prefix "" **{**

policy **=** "write"

**}**

key\_prefix "" **{**

policy **=** "write"

**}**

keyring **=** "write"

node\_prefix "" **{**

policy **=** "write"

**}**

operator **=** "write"

query\_prefix "" **{**

policy **=** "write"

**}**

service\_prefix "" **{**

policy **=** "write"

intentions **=** "write"

**}**

session\_prefix "" **{**

policy **=** "write"

**}**

Get policy details by name:

$ consul acl policy read -name "acl-replication"

ID: 35b8ecb0-707c-ee18-2002-81b238b54b38

Name: acl-replication

Description: Token capable of replicating ACL policies

Datacenters:

Rules:

acl **=** "read"

Get policy details (Builtin Policies):

Builtin policies can be accessed by specifying their original name as the value to the -id parameter.

$ consul acl policy read -id global-management

ID: 00000000-0000-0000-0000-000000000001

Name: global-management

Description: Builtin Policy that grants unlimited access

Datacenters:

Hash: b30210b7aba9facd1c57891e3df27669174a08b690cb2905e0797535f75eba69

Create Index: 4

Modify Index: 4

Rules:

acl **=** "write"

agent\_prefix "" **{**

policy **=** "write"

**}**

event\_prefix "" **{**

policy **=** "write"

**}**

key\_prefix "" **{**

policy **=** "write"

**}**

keyring **=** "write"

node\_prefix "" **{**

policy **=** "write"

**}**

operator **=** "write"

query\_prefix "" **{**

policy **=** "write"

**}**

service\_prefix "" **{**

policy **=** "write"

intentions **=** "write"

**}**

session\_prefix "" **{**

policy **=** "write"

**}**

## [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "update)update

Command: consul acl policy update

This command is used to update a policy. The default operations is to merge the current policy with those values provided to the command invocation. Therefore to update just one field, only the -id or -name options and the option to modify must be provided. Note that renaming policies requires both the -id and -name as the new name cannot yet be used to lookup the policy.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "usage-2)Usage

Usage: consul acl policy update [options] [args]

#### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "options-2)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-policy.html#common-subcommand-options)

* [-description=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "description-lt-string-gt--1) - A description of the policy.

* [-id=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "id-lt-string-gt--1) - The ID of the policy to update. It may be specified as a unique ID prefix but will error if the prefix matches multiple policy IDs

* [-meta](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "meta-2) - Indicates that policy metadata such as the content hash and raft indices should be shown for each entry

* [-name=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "name-lt-string-gt--2) - The policies name.

* [-no-merge](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "no-merge) - Do not merge the current policy information with what is provided to the command. Instead overwrite all fields with the exception of the policy ID which is immutable.

* [-rules=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "rules-lt-string-gt--1) - The policy rules. May be prefixed with @ to indicate that the value is a file path to load the rules from. - may also be given to indicate that the rules are available on stdin.

* [-valid-datacenter=<value>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "valid-datacenter-lt-value-gt--1) - Datacenter that the policy should be valid within. This flag may be specified multiple times.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "examples-2)Examples

Update a policy:

$ consul acl policy update -id 35b8 -name "replication" -description "Policy capable of replication ACL policies and Intentions" -rules @rules.hcl

Policy updated successfully

ID: 35b8ecb0-707c-ee18-2002-81b238b54b38

Name: replication

Description: Policy capable of replication ACL policies and Intentions

Datacenters:

Rules:

acl **=** "read"

service\_prefix "" **{**

policy **=** "read"

intentions **=** "read"

**}**

Rename a policy:

$ consul acl policy update -id 35b8 -name "dc1-replication"

Policy updated successfully

ID: 35b8ecb0-707c-ee18-2002-81b238b54b38

Name: dc1-replication

Description: Policy capable of replication ACL policies and Intentions

Datacenters: dc1

Rules:

acl **=** "read"

service\_prefix "" **{**

policy **=** "read"

intentions **=** "read"

**}**

## [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "delete)delete

Command: consul acl policy delete

This command deletes a policy. Policies may be deleted by their ID or by name.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "usage-3)Usage

Usage: consul acl policy delete [options]

#### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "options-3)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-policy.html#common-subcommand-options)

* [-id=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "id-lt-string-gt--2) - The ID of the policy to delete. It may be specified as a unique ID prefix but will error if the prefix matches multiple policy IDs.

* [-name=<string>](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "name-lt-string-gt--3) - The Name of the policy to delete.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "examples-3)Examples

Delete a policy:

$ consul acl policy delete -id 35b8

Policy "35b8ecb0-707c-ee18-2002-81b238b54b38" deleted successfully

Delete a policy by name:

$ consul acl policy delete -name acl-replication

Policy "35b8ecb0-707c-ee18-2002-81b238b54b38" deleted successfully

## [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "list)list

Command: consul acl policy list

This command lists all policies. By default it will not show metadata.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "usage-4)Usage

Usage: consul acl policy list

#### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "options-4)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-policy.html#common-subcommand-options)

* [-meta](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "meta-3) - Indicates that policy metadata such as the content hash and Raft indices should be shown for each entry.

### [»](https://www.consul.io/docs/commands/acl/acl-policy.html" \l "examples-4)Examples

Default listing.

$ consul acl policy list

global-management:

ID: 00000000-0000-0000-0000-000000000001

Description: Builtin Policy that grants unlimited access

Datacenters:

acl-replication:

ID: 35b8ecb0-707c-ee18-2002-81b238b54b38

Description: Policy capable of replicating ACL policies

Datacenters:

Show Metadata.

$ consul acl policy list -meta

global-management:

ID: 00000000-0000-0000-0000-000000000001

Description: Builtin Policy that grants unlimited access

Datacenters:

Hash: b30210b7aba9facd1c57891e3df27669174a08b690cb2905e0797535f75eba69

Create Index: 4

Modify Index: 4

node-services-read:

ID: 06acc965-df4b-5a99-58cb-3250930c6324

Description: Can read any node and service

Datacenters:

Hash: 19d2a73dcd315506af73bfff1492779a0dc0235066fcac07f432fb2cc3402133

Create Index: 244

Modify Index: 244

acl-replication:

ID: ca44555b-a2d8-94de-d763-88caffdaf11f

Description: Token capable of replicating ACL policies

Datacenters: dc1, dc2

Hash: b94669679cc24e0d064412e4aa90b470b7f900a8e0801f65feaf1f7d716a5390

Create Index: 198

Modify Index: 198

# Consul ACL Set Agent Token

Command: consul acl set-agent-token

This command updates the ACL tokens currently in use by the agent. It can be used to introduce ACL tokens to the agent for the first time, or to update tokens that were initially loaded from the agent's configuration. Tokens are not persisted, so will need to be updated again if the agent is restarted.

## [»](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#usage)Usage

Usage: consul acl set-agent-token [options] TYPE TOKEN

### [»](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html" \l "token-types)Token Types

* [default](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#default) - The default token is the token that the agent will use for both internal agent operations and operations initiated by the HTTP and DNS interfaces when no specific token is provided. If not set the agent will use the anonymous token.
* [agent](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#agent) - The token that the agent will use for internal agent operations. If not given then the default token is used for these operations.

* [master](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html" \l "master) - This sets the token that can be used to access the Agent APIs in the event that the ACL datacenter cannot be reached.

* [replication](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html" \l "replication) - This is the token that the agent will use for replication operations. This token will need to be configured with read access to whatever data is being replicated.

### [»](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

## [»](https://www.consul.io/docs/commands/acl/acl-set-agent-token.html#examples)Examples

Set the default token:

$ consul acl set-agent-token default c4d0f8df-3aba-4ab6-a7a0-35b760dc29a1

# Consul ACL Tokens

Command: consul acl token

The acl token command is used to manage Consul's ACL tokens. There are subcommands for the individual operations that can be performed.

* [create](https://www.consul.io/docs/commands/acl/acl-token.html#create)
* [clone](https://www.consul.io/docs/commands/acl/acl-token.html#clone)
* [read](https://www.consul.io/docs/commands/acl/acl-token.html#read)
* [update](https://www.consul.io/docs/commands/acl/acl-token.html#update)
* [delete](https://www.consul.io/docs/commands/acl/acl-token.html#delete)
* [list](https://www.consul.io/docs/commands/acl/acl-token.html#list)

ACL tokens are also accessible via the [HTTP API](https://www.consul.io/api/acl/acl.html).

Usage: consul acl token <subcommand> [options] [args]

**Note:** All of the examples show for the subcommands will require a valid Consul token with the appropriate permissions. Either set the CONSUL\_HTTP\_TOKEN environment variable to the tokens secret ID or pass the secret ID as the value of the -token parameter.

## [»](https://www.consul.io/docs/commands/acl/acl-token.html" \l "identitying-tokens)Identitying Tokens

In several of the subcommands a token will have to be identified to be read, modified or deleted. Those subcommands support specifying the token by its ID using the -id parameter. The ID may be specified as a unique UUID prefix instead of the entire UUID. As long as it is unique it will be resolve to the full UUID and used. Additionally builtin token names will be accepted as the value of the -id.

Builtin Policies:

| **Token UUID** | **Token Name** |
| --- | --- |
| 00000000-0000-0000-0000-000000000002 | anonymous |

## [»](https://www.consul.io/docs/commands/acl/acl-token.html#common-subcommand-options)Common Subcommand Options

All of the consul acl token subcommands support the following options:

* [-ca-file=<value>](https://www.consul.io/docs/commands/acl/acl-token.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/acl/acl-token.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/acl/acl-token.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/acl/acl-token.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/acl/acl-token.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/acl/acl-token.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/acl/acl-token.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/acl/acl-token.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/acl/acl-token.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

## [»](https://www.consul.io/docs/commands/acl/acl-token.html#create)create

Command: consul acl token create

This command creates new tokens. When creating a new token, policies may be linked using either the -policy-id or the `-policy-name options. When specifying policies by IDs you may use a unique prefix of the UUID as a shortcut for specifying the entire UUID.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#usage)Usage

#### [»](https://www.consul.io/docs/commands/acl/acl-token.html#options)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-token.html#common-subcommand-options)
* [-description=<string>](https://www.consul.io/docs/commands/acl/acl-token.html#description-lt-string-gt-) - A description of the token.

* [-local](https://www.consul.io/docs/commands/acl/acl-token.html" \l "local) - Create this as a datacenter local token.

* [-policy-id=<value>](https://www.consul.io/docs/commands/acl/acl-token.html" \l "policy-id-lt-value-gt-) - ID of a policy to use for this token. May be specified multiple times.

* [-policy-name=<value>](https://www.consul.io/docs/commands/acl/acl-token.html" \l "policy-name-lt-value-gt-) - Name of a policy to use for this token. May be specified multiple times.
* [-meta](https://www.consul.io/docs/commands/acl/acl-token.html#meta) - Indicates that token metadata such as the content hash and raft indices should be shown for each entry.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#examples)Examples

Create a new token:

$ consul acl token create -description "Read Nodes and Services" -policy-id 06acc965

AccessorID: 986193b5-e2b5-eb26-6264-b524ea60cc6d

SecretID: ec15675e-2999-d789-832e-8c4794daa8d7

Description: Read Nodes and Services

Local: false

Create Time: 2018-10-22 15:33:39.01789 -0400 EDT

Policies:

06acc965-df4b-5a99-58cb-3250930c6324 - node-services-read

Create a new local token:

$ consul acl token create -description "Read Nodes and Services" -policy-id 06acc965 -local

AccessorID: 4fdf0ec8-d251-3865-079c-7247c974fc50

SecretID: 02143514-abf2-6c23-0aa1-ec2107e68f6b

Description: Read Nodes and Services

Local: true

Create Time: 2018-10-22 15:34:19.330265 -0400 EDT

Policies:

06acc965-df4b-5a99-58cb-3250930c6324 - node-services-read

Create a new policy and link with policies by name:

$ consul acl token create -description "Super User" -policy-name global-management

AccessorID: 59f86a9b-d3b6-166c-32a0-be4ab3f94caa

SecretID: ada7f751-f654-8872-7f93-498e799158b6

Description: Super User

Local: false

Create Time: 2018-10-22 15:35:28.787003 -0400 EDT

Policies:

00000000-0000-0000-0000-000000000001 - global-management

## [»](https://www.consul.io/docs/commands/acl/acl-token.html" \l "clone)clone

Command: consul acl token clone

This command clones an existing token.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#usage-1)Usage

Usage: `consul acl token clone [options]

#### [»](https://www.consul.io/docs/commands/acl/acl-token.html#options-1)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-token.html#common-subcommand-options)
* [-description=<string>](https://www.consul.io/docs/commands/acl/acl-token.html#description-lt-string-gt--1) - A description of the new cloned token.
* [-id=<string>](https://www.consul.io/docs/commands/acl/acl-token.html#id-lt-string-gt-) - The Accessor ID of the token to clone. It may be specified as a unique ID prefix but will error if the prefix matches multiple token Accessor IDs. The special value of 'anonymous' may be provided instead of the anonymous tokens accessor ID

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#examples-1)Examples

Clone a token:

$ consul acl token clone -id 59f8 -description "Clone of Super User"

Token cloned successfully.

AccessorID: dcfa52ed-9288-b3ff-056d-255ef69d2d88

SecretID: 0005d17e-5bb2-7e8b-7bfa-15f2eee9ad14

Description: Clone of Super User

Local: false

Create Time: 2018-10-22 16:26:02.909096 -0400 EDT

Policies:

00000000-0000-0000-0000-000000000001 - global-management

## [»](https://www.consul.io/docs/commands/acl/acl-token.html#read)read

Command: consul acl token read

This command reads and displays a token details.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#usage-2)Usage

Usage: consul acl token read [options] [args]

#### [»](https://www.consul.io/docs/commands/acl/acl-token.html#options-2)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-token.html#common-subcommand-options)
* [-id=<string>](https://www.consul.io/docs/commands/acl/acl-token.html#id-lt-string-gt--1) - The ID of the policy to read. It may be specified as a unique ID prefix but will error if the prefix matches multiple policy IDs.
* [-meta](https://www.consul.io/docs/commands/acl/acl-token.html#meta-1) - Indicates that policy metadata such as the content hash and raft indices should be shown for each entry.

* [-self](https://www.consul.io/docs/commands/acl/acl-token.html" \l "self) - Indicates that the current HTTP token should be read by secret ID instead of expecting a -id option.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#examples-2)Examples

Get token details:

$ consul acl token read -id 986

AccessorID: 986193b5-e2b5-eb26-6264-b524ea60cc6d

SecretID: ec15675e-2999-d789-832e-8c4794daa8d7

Description: Read Nodes and Services

Local: false

Create Time: 2018-10-22 15:33:39.01789 -0400 EDT

Policies:

06acc965-df4b-5a99-58cb-3250930c6324 - node-services-read

Get token details using the token secret ID:

$consul acl token read -self

AccessorID: 4d123dff-f460-73c3-02c4-8dd64d136e01

SecretID: 86cddfb9-2760-d947-358d-a2811156bf31

Description: Bootstrap Token **(**Global Management**)**

Local: false

Create Time: 2018-10-22 11:27:04.479026 -0400 EDT

Policies:

00000000-0000-0000-0000-000000000001 - global-management

Get token details (Builtin Tokens)

$ consul acl token read -id anonymous

AccessorID: 00000000-0000-0000-0000-000000000002

SecretID: anonymous

Description: Anonymous Token

Local: false

Create Time: 0001-01-01 00:00:00 +0000 UTC

Policies:

## [»](https://www.consul.io/docs/commands/acl/acl-token.html#update)update

Command: consul acl token update

This command will update a token. Some parts of the token like whether the token is local to the datacenter cannot be changed.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#usage-3)Usage

Usage: consul acl token update [options]

#### [»](https://www.consul.io/docs/commands/acl/acl-token.html#options-3)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-token.html#common-subcommand-options)

* [-description=<string>](https://www.consul.io/docs/commands/acl/acl-token.html" \l "description-lt-string-gt--2) - A description of the token
* [-id=<string>](https://www.consul.io/docs/commands/acl/acl-token.html#id-lt-string-gt--2) - The Accessor ID of the token to read. It may be specified as a unique ID prefix but will error if the prefix matches multiple token Accessor IDs

* [-merge-policies](https://www.consul.io/docs/commands/acl/acl-token.html" \l "merge-policies) - Merge the new policies with the existing policies
* [-meta](https://www.consul.io/docs/commands/acl/acl-token.html#meta-2) - Indicates that token metadata such as the content hash and Raft indices should be shown for each entry.

* [-policy-id=<value>](https://www.consul.io/docs/commands/acl/acl-token.html" \l "policy-id-lt-value-gt--1) - ID of a policy to use for this token. May be specified multiple times.

* [-policy-name=<value>](https://www.consul.io/docs/commands/acl/acl-token.html" \l "policy-name-lt-value-gt--1) - Name of a policy to use for this token. May be specified multiple times.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#examples-3)Examples

Update the anonymous token:

$ consul acl token update -id anonymous -policy-id 06acc

Token updated successfully.

AccessorID: 00000000-0000-0000-0000-000000000002

SecretID: anonymous

Description: Anonymous Token

Local: false

Create Time: 0001-01-01 00:00:00 +0000 UTC

Policies:

06acc965-df4b-5a99-58cb-3250930c6324 - node-services-read

Update a token description and take the policies from the existing token:

$ consul acl token update -id 986193 -description "WonderToken" -merge-policies

Token updated successfully.

AccessorID: 986193b5-e2b5-eb26-6264-b524ea60cc6d

SecretID: ec15675e-2999-d789-832e-8c4794daa8d7

Description: WonderToken

Local: false

Create Time: 2018-10-22 15:33:39.01789 -0400 EDT

Policies:

06acc965-df4b-5a99-58cb-3250930c6324 - node-services-read

## [»](https://www.consul.io/docs/commands/acl/acl-token.html#delete)delete

Command: consul acl token delete

This command deletes a token.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#usage-4)Usage

Usage: consul acl token delete [options]

#### [»](https://www.consul.io/docs/commands/acl/acl-token.html#options-4)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-token.html#common-subcommand-options)

* [-id=<string>](https://www.consul.io/docs/commands/acl/acl-token.html" \l "id-lt-string-gt--3) - The ID of the token to delete. It may be specified as a unique ID prefix but will error if the prefix matches multiple token IDs.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html#examples-4)Examples

Delete a token:

$ consul acl token delete -id 35b8

Token "35b8ecb0-707c-ee18-2002-81b238b54b38" deleted successfully

## [»](https://www.consul.io/docs/commands/acl/acl-token.html#list)list

Command: consul acl token list

This command lists all tokens. By default it will not show metadata.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html" \l "usage-5)Usage

Usage: consul acl token list

#### [»](https://www.consul.io/docs/commands/acl/acl-token.html" \l "options-5)Options

* [Common Subcommand Options](https://www.consul.io/docs/commands/acl/acl-token.html#common-subcommand-options)
* [-meta](https://www.consul.io/docs/commands/acl/acl-token.html#meta-3) - Indicates that token metadata such as the content hash and Raft indices should be shown for each entry.

### [»](https://www.consul.io/docs/commands/acl/acl-token.html" \l "examples-5)Examples

Default listing.

$ consul acl token list

AccessorID: 4d123dff-f460-73c3-02c4-8dd64d136e01

Description: Bootstrap Token **(**Global Management**)**

Local: false

Create Time: 2018-10-22 11:27:04.479026 -0400 EDT

Legacy: false

Policies:

00000000-0000-0000-0000-000000000001 - global-management

AccessorID: 59f86a9b-d3b6-166c-32a0-be4ab3f94caa

Description: Super User

Local: false

Create Time: 2018-10-22 15:35:28.787003 -0400 EDT

Legacy: false

Policies:

00000000-0000-0000-0000-000000000001 - global-management

AccessorID: 00000000-0000-0000-0000-000000000002

Description: Anonymous Token

Local: false

Create Time: 0001-01-01 00:00:00 +0000 UTC

Legacy: false

Policies:

06acc965-df4b-5a99-58cb-3250930c6324 - node-services-read

AccessorID: 986193b5-e2b5-eb26-6264-b524ea60cc6d

Description: WonderToken

Local: false

Create Time: 2018-10-22 15:33:39.01789 -0400 EDT

Legacy: false

Policies:

06acc965-df4b-5a99-58cb-3250930c6324 - node-services-read

**Deprecated:** This command exists only as a convenience to make legacy ACL migration easier. It will be removed in a future major release when support for the legacy ACL system is removed.

# Consul ACL Translate Rules

Command: consul acl translate-rules

This command translates the legacy ACL rule syntax into the new syntax.

### [»](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#usage)Usage

Usage: consul acl translate rules [options] TRANSLATE

#### [»](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/acl/acl-translate-rules.html" \l "command-options)Command Options

* [TRANSLATE](https://www.consul.io/docs/commands/acl/acl-translate-rules.html" \l "translate) - The rules to translate. If - is used, then the rules will be read from stdin. If @ is prefixed to the value then the value is considered to be a file and the rules will be read from that file.
* [-token-secret](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#token-secret) - Specifies that what the TRANSLATE argument holds is not a rule set but rather the token secret ID of a legacy ACL token that holds the rule set.

* [-token-accessor](https://www.consul.io/docs/commands/acl/acl-translate-rules.html" \l "token-accessor) - Specifies that what the TRANSLATE argument holds is not a rule set but rather the token accessor ID of a legacy ACL token that holds the rule set.

### [»](https://www.consul.io/docs/commands/acl/acl-translate-rules.html#examples)Examples

Translate rules within a file:

$ consul acl translate-rules @rules.hcl

Translate rules from stdin:

$ consul acl translate-rules -

Translate rules from a string argument:

$ consul acl translate-rules 'key "" { policy = "write"}'

Translate rules for a legacy ACL token using its SecretID passed from stdin:

$ consul acl translate-rules --token-secret -

Translate rules for a legacy ACL token using its AccessorID:

$ consul acl translate-rules 429cd746-03d5-4bbb-a83a-18b164171c89

# Consul Agent

The consul agent command is the heart of Consul: it runs the agent that performs the important task of maintaining membership information, running checks, announcing services, handling queries, etc.

Due to the power and flexibility of this command, the Consul agent is documented in its own section. See the [Consul Agent](https://www.consul.io/docs/agent/basics.html) section for more information on how to use this command and the options it has.

# Consul Catalog

Command: consul catalog

The catalog command is used to interact with Consul's catalog via the command line. It exposes top-level commands for reading and filtering data from the registry.

The catalog is also accessible via the [HTTP API](https://www.consul.io/api/catalog.html).

## [»](https://www.consul.io/docs/commands/catalog.html" \l "basic-examples)Basic Examples

List all datacenters:

$ consul catalog datacenters

dc1

dc2

dc3

List all nodes:

$ consul catalog nodes

Node ID Address DC

worker-01 1b662d97 10.4.5.31 dc1

List all nodes which provide a particular service:

$ consul catalog nodes -service=redis

Node ID Address DC

worker-01 1b662d97 10.4.5.31 dc1

worker-02 d407a592 10.4.4.158 dc1

List all services:

$ consul catalog services

consul

postgresql

redis

List all services on a node:

$ consul catalog services -node=worker-01

consul

postgres

For more examples, ask for subcommand help or view the subcommand documentation by clicking on one of the links in the sidebar.

## [»](https://www.consul.io/docs/commands/catalog.html#usage)Usage

Usage: consul catalog <subcommand>

For the exact documentation for your Consul version, run consul catalog -h to view the complete list of subcommands.

Usage: consul catalog <subcommand> [options] [args]

# ...

Subcommands:

datacenters Lists all known datacenters for this agent

nodes Lists all nodes in the given datacenter

services Lists all registered services in a datacenter

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar or one of the links below:

# Consul Catalog List Datacenters

Command: consul catalog datacenters

The catalog datacenters command prints all known datacenters.

## [»](https://www.consul.io/docs/commands/catalog/datacenters.html#examples)Examples

List all datacenters:

$ consul catalog datacenters

dc1

dc2

dc3

## [»](https://www.consul.io/docs/commands/catalog/datacenters.html#usage)Usage

Usage: consul catalog datacenters [options]

#### [»](https://www.consul.io/docs/commands/catalog/datacenters.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/catalog/datacenters.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/catalog/datacenters.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/catalog/datacenters.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/catalog/datacenters.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/catalog/datacenters.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/catalog/datacenters.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/catalog/datacenters.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/catalog/datacenters.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/catalog/datacenters.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

# Consul Catalog List Nodes

Command: consul catalog nodes

The catalog nodes command prints all known nodes and metadata about them. It can also query for nodes that match a particular metadata or provide a particular service.

## [»](https://www.consul.io/docs/commands/catalog/nodes.html#examples)Examples

List all nodes:

$ consul catalog nodes

Node ID Address DC

worker-01 1b662d97 10.4.5.31 dc1

Print detailed node information such as tagged addresses and node metadata:

$ consul catalog nodes -detailed

Node ID Address DC TaggedAddresses Meta

worker-01 1b662d97-8b5c-3cc2-0ac0-96f55ad423b5 10.4.5.31 dc1 lan=10.4.5.31, wan=10.4.5.31

List nodes which provide the service name "web":

$ consul catalog nodes -service=web

Node ID Address DC TaggedAddresses Meta

worker-01 1b662d97-8b5c-3cc2-0ac0-96f55ad423b5 10.4.5.31 dc1 lan=10.4.5.31, wan=10.4.5.31

Sort the resulting node list by estimated round trip time to worker-05:

$ consul catalog nodes -near=web-05

Node ID Address DC TaggedAddresses Meta

worker-01 1b662d97-8b5c-3cc2-0ac0-96f55ad423b5 10.4.5.31 dc1 lan=10.4.5.31, wan=10.4.5.31

worker-02 d407a592-e93c-4d8e-8a6d-aba853d1e067 10.4.4.158 dc1 lan=10.4.4.158, wan=10.4.4.158

## [»](https://www.consul.io/docs/commands/catalog/nodes.html#usage)Usage

Usage: consul catalog nodes [options]

#### [»](https://www.consul.io/docs/commands/catalog/nodes.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/catalog/nodes.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/catalog/nodes.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/catalog/nodes.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/catalog/nodes.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/catalog/nodes.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/catalog/nodes.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/catalog/nodes.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/catalog/nodes.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/catalog/nodes.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/catalog/nodes.html" \l "catalog-list-nodes-options)Catalog List Nodes Options

* [-detailed](https://www.consul.io/docs/commands/catalog/nodes.html" \l "detailed) - Output detailed information about the nodes including their addresses and metadata.

* [-near=<string>](https://www.consul.io/docs/commands/catalog/nodes.html" \l "near-lt-string-gt-)- Node name to sort the node list in ascending order based on estimated round-trip time from that node. Passing "\_agent" will use this agent's node for sorting.

* [-node-meta=<key=value>](https://www.consul.io/docs/commands/catalog/nodes.html" \l "node-meta-lt-key-value-gt-) - Metadata to filter nodes with the given key=value pairs. This flag may be specified multiple times to filter on multiple sources of metadata.

* [-service=<id or name>](https://www.consul.io/docs/commands/catalog/nodes.html" \l "service-lt-id-or-name-gt-) - Service id or name to filter nodes. Only nodes which are providing the given service will be returned.

# Consul Catalog List Services

Command: consul catalog services

The catalog services command prints all known services. It can also query for services that match particular metadata or list the services that a particular node provides.

## [»](https://www.consul.io/docs/commands/catalog/services.html#examples)Examples

List all services:

$ consul catalog services

consul

postgresql

redis

Show all services with their tags:

$ consul catalog services -tags

consul

postgresql leader

redis primary,v1

List services for the node "worker-01":

$ consul catalog services -node=worker-01

consul

redis

## [»](https://www.consul.io/docs/commands/catalog/services.html#usage)Usage

Usage: consul catalog services [options]

#### [»](https://www.consul.io/docs/commands/catalog/services.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/catalog/services.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/catalog/services.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/catalog/services.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/catalog/services.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/catalog/services.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/catalog/services.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/catalog/services.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/catalog/services.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/catalog/services.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/catalog/services.html#catalog-list-nodes-options)Catalog List Nodes Options

* [-node=<id or name>](https://www.consul.io/docs/commands/catalog/services.html" \l "node-lt-id-or-name-gt-) - Node id or name for which to list services.
* [-node-meta=<key=value>](https://www.consul.io/docs/commands/catalog/services.html#node-meta-lt-key-value-gt-) - Metadata to filter nodes with the given key=value pairs. If specified, only services running on nodes matching the given metadata will be returned. This flag may be specified multiple times to filter on multiple sources of metadata.

* [-tags](https://www.consul.io/docs/commands/catalog/services.html" \l "tags) - Display each service's tags as a comma-separated list beside each service entry.

# Consul Connect

Command: consul connect

The connect command is used to interact with Connect [Connect](https://www.consul.io/docs/connect/intentions.html) subsystems. It exposes commands for running the built-in mTLS proxy and viewing/updating the Certificate Authority (CA) configuration. This command is available in Consul 1.2 and later.

## [»](https://www.consul.io/docs/commands/connect.html#usage)Usage

Usage: consul connect <subcommand>

For the exact documentation for your Consul version, run consul connect -h to view the complete list of subcommands.

Usage: consul connect <subcommand> [options] [args]

This command has subcommands for interacting with Consul Connect.

Here are some simple examples, and more detailed examples are available

in the subcommands or the documentation.

Run the built-in Connect mTLS proxy

$ consul connect proxy

For more examples, ask for subcommand help or view the documentation.

Subcommands:

ca Interact with the Consul Connect Certificate Authority (CA)

proxy Runs a Consul Connect proxy

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar.

# Consul Connect Certificate Authority (CA)

Command: consul connect ca

The CA connect command is used to interact with Consul Connect's Certificate Authority subsystem. The command can be used to view or modify the current CA configuration. See the [Connect CA documentation](https://www.consul.io/docs/connect/ca.html) for more information.

Usage: consul connect ca <subcommand> [options] [args]

This command has subcommands for interacting with Consul Connect's

Certificate Authority (CA).

Here are some simple examples, and more detailed examples are available

in the subcommands or the documentation.

Get the configuration:

$ consul connect ca get-config

Update the configuration:

$ consul connect ca set-config -config-file ca.json

For more examples, ask for subcommand help or view the documentation.

Subcommands:

get-config Display the current Connect Certificate Authority (CA) configuration

set-config Modify the current Connect CA configuration

## [»](https://www.consul.io/docs/commands/connect/ca.html" \l "get-config)get-config

This command displays the current CA configuration.

Usage: consul connect ca get-config [options]

#### [»](https://www.consul.io/docs/commands/connect/ca.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/connect/ca.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/connect/ca.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/connect/ca.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/connect/ca.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/connect/ca.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/connect/ca.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/connect/ca.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/connect/ca.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/connect/ca.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

The output looks like this:

{

"Provider": "consul",

"Config": {},

"CreateIndex": 5,

"ModifyIndex": 197

}

## [»](https://www.consul.io/docs/commands/connect/ca.html" \l "set-config)set-config

Modifies the current CA configuration. If this results in a new root certificate being used, the [Root Rotation](https://www.consul.io/docs/connect/ca.html#root-certificate-rotation) process will be triggered.

Usage: consul connect ca set-config [options]

#### [»](https://www.consul.io/docs/commands/connect/ca.html" \l "api-options-1)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/connect/ca.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/connect/ca.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/connect/ca.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/connect/ca.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/connect/ca.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/connect/ca.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/connect/ca.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/connect/ca.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/connect/ca.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/connect/ca.html#command-options)Command Options

* [-config-file](https://www.consul.io/docs/commands/connect/ca.html" \l "config-file) - (required) Specifies a JSON-formatted file to use for the new configuration.

The output looks like this:

Configuration updated!

The return code will indicate success or failure.

# Consul Connect Proxy

Command: consul connect proxy

The connect proxy command is used to run Consul's built-in mTLS proxy for use with Connect. This can be used in production to enable a Connect-unaware application to accept and establish Connect-based connections. This proxy can also be used in development to connect to Connect-enabled services.

## [»](https://www.consul.io/docs/commands/connect/proxy.html#usage)Usage

Usage: consul connect proxy [options]

#### [»](https://www.consul.io/docs/commands/connect/proxy.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/connect/proxy.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/connect/proxy.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/connect/proxy.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/connect/proxy.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/connect/proxy.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/connect/proxy.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/connect/proxy.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/connect/proxy.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/connect/proxy.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/connect/proxy.html" \l "proxy-options)Proxy Options

* [-sidecar-for](https://www.consul.io/docs/commands/connect/proxy.html" \l "sidecar-for) - The ID (not name if they differ) of the service instance this proxy will represent. The target service doesn't need to exist on the local agent yet but a [sidecar proxy registration](https://www.consul.io/docs/connect/proxies.html#sidecar-proxy-fields) with proxy.destination\_service\_idequal to the passed value must be present. If multiple proxy registrations targeting the same local service instance are present the command will error and -proxy-id should be used instead.

* [-proxy-id](https://www.consul.io/docs/commands/connect/proxy.html" \l "proxy-id) - The [proxy service](https://www.consul.io/docs/connect/proxies.html#proxy-service-definitions) ID on the local agent. This must already be present on the local agent.

* [-log-level](https://www.consul.io/docs/commands/connect/proxy.html" \l "log-level) - Specifies the log level.

* [-pprof-addr](https://www.consul.io/docs/commands/connect/proxy.html" \l "pprof-addr) - Enable debugging via pprof. Providing a host:port (or just ':port') enables profiling HTTP endpoints on that address.

* [-service](https://www.consul.io/docs/commands/connect/proxy.html" \l "service) - Name of the service this proxy is representing. This service doesn't need to actually exist in the Consul catalog, but proper ACL permissions (service:write) are required. This and the remaining options can be used to setup a proxy that is not registered already with local config [useful for development](https://www.consul.io/docs/connect/dev.html).

* [-upstream](https://www.consul.io/docs/commands/connect/proxy.html" \l "upstream) - Upstream service to support connecting to. The format should be 'name:addr', such as 'db:8181'. This will make 'db' available on port 8181. When a regular TCP connection is made to port 8181, the proxy will service discover "db" and establish a Connect mTLS connection identifying as the -service value. This flag can be repeated multiple times.

* [-listen](https://www.consul.io/docs/commands/connect/proxy.html" \l "listen) - Address to listen for inbound connections to the proxied service. Must be specified with -service and -service-addr. If this isn't specified, an inbound listener is not started.

* [-service-addr](https://www.consul.io/docs/commands/connect/proxy.html" \l "service-addr) - Address of the local service to proxy. Required for -listen.

* [-register](https://www.consul.io/docs/commands/connect/proxy.html" \l "register) - Self-register with the local Consul agent, making this proxy available as Connect-capable service in the catalog. This is only useful with -listen.

* [-register-id](https://www.consul.io/docs/commands/connect/proxy.html" \l "register-id) - Optional ID suffix for the service when -register is set to disambiguate the service ID. By default the service ID is "-proxy" where <service> is the -service value. In most cases it is now preferable to use [consul services register](https://www.consul.io/docs/commands/services/register.html) to register a fully configured proxy instance rather than specify config and registration via this command.

## [»](https://www.consul.io/docs/commands/connect/proxy.html#examples)Examples

The example below shows how to start a local proxy for establishing outbound connections to "db" representing the frontend service. Once running, any process that creates a TCP connection to the specified port (8181) will establish a mutual TLS connection to "db" identified as "frontend".

$ consul connect proxy -service frontend -upstream db:8181

The next example starts a local proxy that also accepts inbound connections on port 8443, authorizes the connection, then proxies it to port 8080:

$ consul connect proxy \

-service frontend \

-service-addr 127.0.0.1:8080 \

-listen ':8443'

# Consul Connect Envoy

Command: consul connect envoy

The connect Envoy command is used to generate a bootstrap configuration for [Envoy proxy](https://envoyproxy.io/) for use with [Consul Connect](https://www.consul.io/docs/connect/).

The default behaviour is to generate the necessary bootstrap configuration for Envoy based on the environment variables and options provided and by taking to the local Consul agent. It execs an external Envoy binary with that configuration leaving the Envoy process running in the foreground. An error is returned on operating systems other than linux or macOS since Envoy does not build for other platforms currently.

If the -bootstrap option is specified, the bootstrap config is generated in the same way and then printed to stdout. This allows it to be redirected to a file and used with envoy -c bootstrap.json. This works on all operating systems allowing configuration to be generated on a host that Envoy doesn't build on but then used in a virtualized environment that can run Envoy.

## [»](https://www.consul.io/docs/commands/connect/envoy.html#usage)Usage

Usage: consul connect envoy [options] [-- pass-through options]

#### [»](https://www.consul.io/docs/commands/connect/envoy.html#api-options)API Options

The standard API options are used to connect to the local agent to discover the proxy configuration needed.

* [-grpc-addr=<addr>](https://www.consul.io/docs/commands/connect/envoy.html" \l "grpc-addr-lt-addr-gt-) - Address of the Consul agent with grpc port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_GRPC\_ADDR environment variable. In Consul 1.3 and later, the default value is [http://127.0.0.1:8502](http://127.0.0.1:8502/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket using unix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.

**Note:** gRPC uses the same TLS settings as the HTTPS API. If HTTPS is enabled then gRPC will require HTTPS as well.

* [-ca-file=<value>](https://www.consul.io/docs/commands/connect/envoy.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/connect/envoy.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/connect/envoy.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/connect/envoy.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/connect/envoy.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/connect/envoy.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/connect/envoy.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/connect/envoy.html" \l "envoy-options)Envoy Options

* [-sidecar-for](https://www.consul.io/docs/commands/connect/envoy.html#sidecar-for) - The ID (not name if they differ) of the service instance this proxy will represent. The target service doesn't need to exist on the local agent yet but a [sidecar proxy registration](https://www.consul.io/docs/connect/proxies.html#sidecar-proxy-fields) with proxy.destination\_service\_idequal to the passed value must be present. If multiple proxy registrations targeting the same local service instance are present the command will error and -proxy-id should be used instead.
* [-proxy-id](https://www.consul.io/docs/commands/connect/envoy.html#proxy-id) - The [proxy service](https://www.consul.io/docs/connect/proxies.html#proxy-service-definitions) ID on the local agent. This must already be present on the local agent.

**Note:** If ACLs are enabled, a token granting service:write for the target service (configured in proxy.destination\_service\_name) must be passed using the -token option or CONSUL\_HTTP\_TOKEN environment variable. This token authorizes the proxy to obtain TLS certificates representing the target service.

* [-envoy-binary](https://www.consul.io/docs/commands/connect/envoy.html" \l "envoy-binary) - The full path to a specific Envoy binary to exec. By default the current $PATH is searched for envoy.

* [-admin-bind](https://www.consul.io/docs/commands/connect/envoy.html" \l "admin-bind) - The host:port to bind Envoy's admin HTTP API. Default is localhost:19000. Envoy requires that this be enabled. The host part must be resolvable DNS name or IP address.

* [-bootstrap](https://www.consul.io/docs/commands/connect/envoy.html" \l "bootstrap) - If present, the command will simply output the generated bootstrap config to stdout in JSON protobuf form. This can be directed to a file and used to start Envoy with envoy -c bootstrap.json.

**Security Note:** If ACLs are enabled the bootstrap JSON will contain the ACL token from -token or the environment and so should be handled as a secret. This token grants the identity of any service it has service:write permission for and so can be used to access any upstream service that that service is allowed to access by [Connect intentions](https://www.consul.io/docs/connect/intentions.html).

* [-- [pass-through options]](https://www.consul.io/docs/commands/connect/envoy.html" \l "pass-through-options-) - Any options given after a double dash are passed directly through to the envoyinvocation. See [Envoy's documentation](https://www.envoyproxy.io/docs) for more details. The command always specifies --config-file and --v2-config-only and by default passes --disable-hot-restart see [hot restart](https://www.consul.io/docs/commands/connect/envoy.html#hot-restart).

## [»](https://www.consul.io/docs/commands/connect/envoy.html#examples)Examples

Assume a local service instance is registratered on the local agent with a sidecar proxy (using the [sidecar service registration](https://www.consul.io/docs/connect/proxies/sidecar-service.html) helper) as below.

service {

name **=** "web"

port **=** 8080

connect { sidecar\_service {} }

}

The sidecar Envoy process can be started with.

$ consul connect envoy -sidecar-for web

This example assumes that the correct [environment variables](https://www.consul.io/docs/commands/connect/envoy.html#api-options) are used to set the local agent connection information and ACL token, or that the agent is using all-default configuration.

To pass additional arguments directly to Envoy, for example output logging level, you can use:

$ consul connect envoy -sidecar-for web -- -l debug

To run multiple different proxy instances on the same host, you will need to use -admin-bind on all but one to ensure they don't attempt to bind to the same port as in the following example.

$ consul connect envoy -sidecar-for db -admin-bind localhost:19001

## [»](https://www.consul.io/docs/commands/connect/envoy.html" \l "exec-security-details)Exec Security Details

The command needs to pass the bootstrap config through to Envoy. Envoy currently only supports passing this as a file path or passing a whole string on the command line with --config-yaml. Since the bootstrap needs to contain the ACL token to authorize the proxy, this secret needs careful handling.

Passing a secret via command option is unacceptable as on many unix systems these are readable to any user on the host for example via /proc or via a setuid process like ps.

Creating a temporary file is more secure in that it can only be read by the current user but risks leaving secret material on disk for an unbounded length of time and in a location that is opaque to the operator.

To work around these issues, the command currently creates a temporary file and immediately unlinks it so it can't be read by any other process that doesn't already have the file descriptor. It then writes the bootstrap JSON, and unsets the CLOEXEC bit on the file handle so that it remains available to the Envoy process after exec. Finally it execs Envoy with --config-file /dev/fd/X where X is the the file descriptor number of the temp file.

This ensures that Envoy can read the file without any other normal user process being able to (assuming they don't have privileged access to /proc). Once the Envoy process stops, there is no longer any reference to the file to clean up.

## [»](https://www.consul.io/docs/commands/connect/envoy.html" \l "envoy-hot-restart)Envoy Hot Restart

Envoy supports hot restart which requires simple external coordination. By default, this command will add --disable-hot-restart when it runs Envoy.

The reason for this default behavior is to make it easy to test and run local demonstrations with multiple Envoy instances outside of cgroups or network namespaces.

To use hot restart, Envoy needs to be started with either the --restart-epoch option. If this command detects that option in the pass-through flags it will not add --disable-hot-restart allowing hot restart to work normally.

The only difference to note over running Envoy directly is that --restart-epoch must be explicitly set to 0 for the initial launch of the Envoy instance to avoid disabling hot restart entirely. The official hot-restarter.py always sets this option so should work as recommended.

# Consul Debug

Command: consul debug

The consul debug command monitors a Consul agent for the specified period of time, recording information about the agent, cluster, and environment to an archive written to the current directory.

Providing support for complex issues encountered by Consul operators often requires a large amount of debugging information to be retrieved. This command aims to shortcut that coordination and provide a simple workflow for accessing data about Consul agent, cluster, and environment to enable faster isolation and debugging of issues.

This command requires an operator:read ACL token in order to retrieve the data from the target agent, if ACLs are enabled.

If the command is interrupted, as it could be given a long duration but require less time than expected, it will attempt to archive the current captured data.

## [»](https://www.consul.io/docs/commands/debug.html" \l "security-and-privacy)Security and Privacy

By default, ACL tokens, private keys, and other sensitive material related to Consul is sanitized and not available in this archive. However, other information about the environment the target agent is running in is available in plain text within the archive.

It is recommended to validate the contents of the archive and redact any material classified as sensitive to the target environment, or use the -capture flag to not retrieve it initially.

Additionally, we recommend securely transmitting this archive via encryption or otherwise.

## [»](https://www.consul.io/docs/commands/debug.html#usage)Usage

Usage: consul debug [options]

By default, the debug command will capture an archive at the current path for all targets for 2 minutes.

#### [»](https://www.consul.io/docs/commands/debug.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/debug.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/debug.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/debug.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/debug.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/debug.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/debug.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/debug.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/debug.html#command-options)Command Options

* [-duration](https://www.consul.io/docs/commands/debug.html" \l "duration) - Optional, the total time to capture data for from the target agent. Must be greater than the interval and longer than 10 seconds. Defaults to 2 minutes.

* [-interval](https://www.consul.io/docs/commands/debug.html" \l "interval) - Optional, the interval at which to capture dynamic data, such as logs and metrics. Must be longer than 5 seconds. Defaults to 30 seconds.

* [-capture](https://www.consul.io/docs/commands/debug.html" \l "capture) - Optional, can be specified multiple times for each [capture target](https://www.consul.io/docs/commands/debug.html#capture-targets) and will only record that information in the archive.
* [-output](https://www.consul.io/docs/commands/debug.html#output) - Optional, the full path of where to write the directory of data and resulting archive. Defaults to the current directory.

* [-archive](https://www.consul.io/docs/commands/debug.html" \l "archive) - Optional, if the tool show archive the directory of data into a compressed tar file. Defaults to true.

## [»](https://www.consul.io/docs/commands/debug.html" \l "capture-targets)Capture Targets

The -capture flag can be specified multiple times to capture specific information when debug is running. By default, it captures all information.

| **Target** | **Description** |
| --- | --- |
| agent | Version and configuration information about the agent. |
| host | Information about resources on the host running the target agent such as CPU, memory, and disk. |
| cluster | A list of all the WAN and LAN members in the cluster. |
| metrics | Metrics from the in-memory metrics endpoint in the target, captured at the interval. |
| logs | DEBUG level logs for the target agent, captured for the interval. |
| pprof | Golang heap, CPU, goroutine, and trace profiling. This information is not retrieved unless [enable\_debug](https://www.consul.io/docs/agent/options.html#enable_debug) is set to true on the target agent. |

## [»](https://www.consul.io/docs/commands/debug.html#examples)Examples

This command can be run from any host with the Consul binary, but requires network access to the target agent in order to retrieve data. Once retrieved, the data is written to the the specified path (defaulting to the current directory) on the host where the command runs.

By default the command will capture all available data from the default agent address on loopback for 2 minutes at 30 second intervals.

$ consul debug

...

In this example, the archive is collected from a different agent on the network using the standard Consul CLI flag to change the API address.

$ consul debug -http-addr=10.0.1.10:8500

...

The capture flag can be specified to only record a subset of data about the agent and environment.

$ consul debug -capture agent -capture host -capture logs

...

The duration of the command and interval of capturing dynamic information (such as metrics) can be specified with the -interval and -duration flags.

$ consul debug -interval=15s -duration=1m

...

# Consul Event

Command: consul event

The event command provides a mechanism to fire a custom user event to an entire datacenter. These events are opaque to Consul, but they can be used to build scripting infrastructure to do automated deploys, restart services, or perform any other orchestration action. Events can be handled by [using a watch](https://www.consul.io/docs/agent/watches.html).

Under the hood, events are propagated using the [gossip protocol](https://www.consul.io/docs/internals/gossip.html).

While the details are not important for using events, an understanding of the semantics is useful. The gossip layer will make a best-effort to deliver the event, but there is **no guaranteed delivery**. Unlike most Consul data, which is replicated using [consensus](https://www.consul.io/docs/internals/consensus.html), event data is purely peer-to-peer over gossip. This means it is not persisted and does not have a total ordering. In practice, this means you cannot rely on the order of message delivery. An advantage however is that events can still be used even in the absence of server nodes or during an outage.

The underlying gossip also sets limits on the size of a user event message. It is hard to give an exact number, as it depends on various parameters of the event, but the payload should be kept very small (< 100 bytes). Specifying too large of an event will return an error.

## [»](https://www.consul.io/docs/commands/event.html#usage)Usage

Usage: consul event [options] [payload]

The only required option is -name which specifies the event name. An optional payload can be provided as the final argument.

#### [»](https://www.consul.io/docs/commands/event.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/event.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/event.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/event.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/event.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/event.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/event.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/event.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/event.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/event.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/event.html#command-options)Command Options

* [-name](https://www.consul.io/docs/commands/event.html" \l "name) - The name of the event.

* [-node](https://www.consul.io/docs/commands/event.html" \l "node) - Regular expression to filter nodes which should evaluate the event.
* [-service](https://www.consul.io/docs/commands/event.html#service) - Regular expression to filter to only nodes with matching services.

* [-tag](https://www.consul.io/docs/commands/event.html" \l "tag) - Regular expression to filter to only nodes with a service that has a matching tag. This must be used with -service. As an example, you may do -service mysql -tag secondary.

# Consul Exec

Command: consul exec

The exec command provides a mechanism for remote execution. For example, this can be used to run the uptimecommand across all machines providing the web service.

Remote execution works by specifying a job, which is stored in the KV store. Agents are informed about the new job using the [event system](https://www.consul.io/docs/commands/event.html), which propagates messages via the [gossip protocol](https://www.consul.io/docs/internals/gossip.html). As a result, delivery is best-effort, and there is **no guarantee** of execution.

While events are purely gossip driven, remote execution relies on the KV store as a message broker. As a result, the execcommand will not be able to properly function during a Consul outage.

**Verbose output warning:** use care to make sure that your command does not produce a large volume of output. Writes to the KV store for this output go through the Consul servers and the Raft consensus algorithm, so having a large number of nodes in the cluster flow a large amount of data through the KV store could make the cluster unavailable.

The table below shows the [required ACLs](https://www.consul.io/api/index.html#acls) in order to execute this command.

| **ACL Required** | **Scope** |
| --- | --- |
| agent:read | local agent |
| session:write | local agent |
| key:write | "\_rexec" prefix |
| event:write | "\_rexec" prefix |

## [»](https://www.consul.io/docs/commands/exec.html#usage)Usage

Usage: consul exec [options] [-|command...]

The only required option is a command to execute. This is either given as trailing arguments, or by specifying -; STDIN will be read to completion as a script to evaluate.

#### [»](https://www.consul.io/docs/commands/exec.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/exec.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/exec.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/exec.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/exec.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/exec.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/exec.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/exec.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/exec.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/exec.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/exec.html#command-options)Command Options

* [-prefix](https://www.consul.io/docs/commands/exec.html" \l "prefix) - Key prefix in the KV store to use for storing request data. Defaults to \_rexec.
* [-node](https://www.consul.io/docs/commands/exec.html#node) - Regular expression to filter nodes which should evaluate the event.
* [-service](https://www.consul.io/docs/commands/exec.html#service) - Regular expression to filter to only nodes with matching services.

* [-shell](https://www.consul.io/docs/commands/exec.html" \l "shell) - Optional, use a shell to run the command. The default value is true.
* [-tag](https://www.consul.io/docs/commands/exec.html#tag) - Regular expression to filter to only nodes with a service that has a matching tag. This must be used with -service. As an example, you may do -service mysql -tag secondary.

* [-wait](https://www.consul.io/docs/commands/exec.html" \l "wait) - Specifies the period of time in which no agent's respond before considering the job finished. This is basically the quiescent time required to assume completion. This period is not a hard deadline, and the command will wait longer depending on various heuristics.

* [-wait-repl](https://www.consul.io/docs/commands/exec.html" \l "wait-repl) - Period to wait after writing the job specification for replication. This is a heuristic value and enables agents to do a stale read of the job. Defaults to 200 msec.

* [-verbose](https://www.consul.io/docs/commands/exec.html" \l "verbose) - Enables verbose output.

# Consul Force Leave

Command: consul force-leave

The force-leave command forces a member of a Consul cluster to enter the "left" state. If the member is still actually alive, it will eventually rejoin the cluster. The true purpose of this method is to force remove "failed" nodes.

Consul periodically tries to reconnect to "failed" nodes in case it is a network partition. After some configured amount of time (by default 72 hours), Consul will reap "failed" nodes and stop trying to reconnect. The force-leave command can be used to transition the "failed" nodes to "left" nodes more quickly.

This can be particularly useful for a node that was running as a server, as it will be removed from the Raft quorum. Note that force-leave cannot be used to force removal of nodes that are outside of the datacenter.

## [»](https://www.consul.io/docs/commands/force-leave.html#usage)Usage

Usage: consul force-leave [options] node

#### [»](https://www.consul.io/docs/commands/force-leave.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/force-leave.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/force-leave.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/force-leave.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/force-leave.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/force-leave.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/force-leave.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/force-leave.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

# Consul Info

Command: consul info

The info command provides various debugging information that can be useful to operators. Depending on if the agent is a client or server, information about different sub-systems will be returned.

There are currently the top-level keys for:

* agent: Provides information about the agent
* consul: Information about the consul library (client or server)
* raft: Provides info about the Raft [consensus library](https://www.consul.io/docs/internals/consensus.html)
* serf\_lan: Provides info about the LAN [gossip pool](https://www.consul.io/docs/internals/gossip.html)
* serf\_wan: Provides info about the WAN [gossip pool](https://www.consul.io/docs/internals/gossip.html)

Here is an example output:

agent:

check\_monitors = 0

check\_ttls = 0

checks = 0

services = 0

consul:

bootstrap = true

known\_datacenters = 1

leader = true

server = true

raft:

applied\_index = 45832

commit\_index = 45832

fsm\_pending = 0

last\_log\_index = 45832

last\_log\_term = 4

last\_snapshot\_index = 45713

last\_snapshot\_term = 1

num\_peers = 2

state = Leader

term = 4

serf\_lan:

event\_queue = 0

event\_time = 2

failed = 0

intent\_queue = 0

left = 0

member\_time = 7

members = 3

query\_queue = 0

query\_time = 1

serf\_wan:

event\_queue = 0

event\_time = 1

failed = 0

intent\_queue = 0

left = 0

member\_time = 1

members = 1

query\_queue = 0

query\_time = 1

## [»](https://www.consul.io/docs/commands/info.html#usage)Usage

Usage: consul info

#### [»](https://www.consul.io/docs/commands/info.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/info.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/info.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/info.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/info.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/info.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/info.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/info.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

# Consul Intention

Command: consul intention

The intention command is used to interact with Connect [intentions](https://www.consul.io/docs/connect/intentions.html). It exposes commands for creating, updating, reading, deleting, checking, and managing intentions. This command is available in Consul 1.2 and later.

Intentions may also be managed via the [HTTP API](https://www.consul.io/api/connect/intentions.html).

## [»](https://www.consul.io/docs/commands/intention.html#usage)Usage

Usage: consul intention <subcommand>

For the exact documentation for your Consul version, run consul intention -h to view the complete list of subcommands.

Usage: consul intention <subcommand> [options] [args]

...

Subcommands:

check Check whether a connection between two services is allowed.

create Create intentions for service connections.

delete Delete an intention.

get Show information about an intention.

match Show intentions that match a source or destination.

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar.

## [»](https://www.consul.io/docs/commands/intention.html#basic-examples)Basic Examples

Create an intention to allow "web" to talk to "db":

$ consul intention create web db

Test whether a "web" is allowed to connect to "db":

$ consul intention check web db

Find all intentions for communicating to the "db" service:

$ consul intention match db

# Consul Intention Check

Command: consul intention check

The intention check command checks whether a connection attempt between two services would be authorized given the current set of intentions and Consul configuration.

This command requires less ACL permissions than other intention-related tasks because no information about the intention is revealed. Therefore, callers only need to have service:read access for the destination. Richer commands like [match](https://www.consul.io/docs/commands/intention/match.html) require full intention read permissions and don't evaluate the result.

## [»](https://www.consul.io/docs/commands/intention/check.html#usage)Usage

Usage: consul intention check [options] SRC DST

#### [»](https://www.consul.io/docs/commands/intention/check.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/intention/check.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/intention/check.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/intention/check.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/intention/check.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/intention/check.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/intention/check.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/intention/check.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

## [»](https://www.consul.io/docs/commands/intention/check.html#examples)Examples

$ consul intention check web db

Denied

$ consul intention check web billing

Allowed

# Consul Intention Create

Command: consul intention create

The intention create command creates or updates an intention.

## [»](https://www.consul.io/docs/commands/intention/create.html#usage)Usage

Usage: consul intention create [options] SRC DST Usage: consul intention create [options] -f FILE...

#### [»](https://www.consul.io/docs/commands/intention/create.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/intention/create.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/intention/create.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/intention/create.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/intention/create.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/intention/create.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/intention/create.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/intention/create.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/intention/create.html" \l "intention-create-options)Intention Create Options

* [-allow](https://www.consul.io/docs/commands/intention/create.html" \l "allow) - Set the action to "allow" for intentions. This is the default.

* [-deny](https://www.consul.io/docs/commands/intention/create.html" \l "deny) - Set the action to "deny" for intentions. This cannot be specified with -allow.

* [-file](https://www.consul.io/docs/commands/intention/create.html" \l "file) - Read intention data one or more files specified by the command line arguments, instead of source/destination pairs.

* [-meta key=value](https://www.consul.io/docs/commands/intention/create.html" \l "meta-key-value) - Specify arbitrary KV metadata to associate with the intention.

* [-replace](https://www.consul.io/docs/commands/intention/create.html" \l "replace) - Replace any matching intention. The replacement is done atomically per intention.

## [»](https://www.consul.io/docs/commands/intention/create.html#examples)Examples

Create an intention web => db:

$ consul intention create web db

Create intentions from a set of files:

$ consul intention create -file one.json two.json

Create intentions from a directory using shell expansion:

$ consul intention create -file intentions/\*.json

# Consul Intention Delete

Command: consul intention delete

The intention delete command deletes a matching intention.

## [»](https://www.consul.io/docs/commands/intention/delete.html#usage)Usage

Usage:

* [consul intention delete [options] SRC DST](https://www.consul.io/docs/commands/intention/delete.html" \l "consul-intention-delete-options-src-dst)

* [consul intention delete [options] ID](https://www.consul.io/docs/commands/intention/delete.html" \l "consul-intention-delete-options-id)

#### [»](https://www.consul.io/docs/commands/intention/delete.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/intention/delete.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/intention/delete.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/intention/delete.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/intention/delete.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/intention/delete.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/intention/delete.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/intention/delete.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

## [»](https://www.consul.io/docs/commands/intention/delete.html#examples)Examples

Delete an intention from "web" to "db" with any action:

$ consul intention delete web db

Delete an intention by unique ID:

$ consul intention delete 4ffed935-439c-695d-4f51-f4fc0b12a7a7

# Consul Intention Get

Command: consul intention get

The intention get command shows a single intention.

## [»](https://www.consul.io/docs/commands/intention/get.html#usage)Usage

Usage:

* [consul intention get [options] SRC DST](https://www.consul.io/docs/commands/intention/get.html" \l "consul-intention-get-options-src-dst)

* [consul intention get [options] ID](https://www.consul.io/docs/commands/intention/get.html" \l "consul-intention-get-options-id)

#### [»](https://www.consul.io/docs/commands/intention/get.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/intention/get.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/intention/get.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/intention/get.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/intention/get.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/intention/get.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/intention/get.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/intention/get.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

## [»](https://www.consul.io/docs/commands/intention/get.html#examples)Examples

$ consul intention get web db

Source: web

Destination: db

Action: deny

ID: 20edfa56-9cd4-51db-8c22-db09fdec61ef

Created At: Thursday, 24-May-18 17:07:49 PDT

# Consul Intention Match

Command: consul intention match

The intention match command shows the list of intentions that match a given source or destination. The list of intentions is listed in evaluation order: the first intention that matches a request would be evaluated.

The [check](https://www.consul.io/docs/commands/intention/check.html) command can be used to check whether a connection would be authorized between any two services.

## [»](https://www.consul.io/docs/commands/intention/match.html#usage)Usage

Usage: consul intention match [options] SRC\_OR\_DST

#### [»](https://www.consul.io/docs/commands/intention/match.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/intention/match.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/intention/match.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/intention/match.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/intention/match.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/intention/match.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/intention/match.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/intention/match.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/intention/match.html" \l "intention-match-options)Intention Match Options

* [-destination](https://www.consul.io/docs/commands/intention/match.html" \l "destination) - Match by destination.

* [-source](https://www.consul.io/docs/commands/intention/match.html" \l "source) - Match by source.

## [»](https://www.consul.io/docs/commands/intention/match.html#examples)Examples

$ consul intention match -source web

web => db (deny)

web => \* (allow)

# Consul Join

Command: consul join

The join command tells a Consul agent to join an existing cluster. A new Consul agent must join with at least one existing member of a cluster in order to join an existing cluster. After joining that one member, the gossip layer takes over, propagating the updated membership state across the cluster.

If you don't join an existing cluster, then that agent is part of its own isolated cluster. Other nodes can join it.

Agents can join other agents multiple times without issue. If a node that is already part of a cluster joins another node, then the clusters of the two nodes join to become a single cluster.

## [»](https://www.consul.io/docs/commands/join.html#usage)Usage

Usage: consul join [options] address ...

You may call join with multiple addresses if you want to try to join multiple clusters. Consul will attempt to join all addresses, and the join command will fail only if Consul was unable to join with any.

#### [»](https://www.consul.io/docs/commands/join.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/join.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/join.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/join.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/join.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/join.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/join.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/join.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/join.html#command-options)Command Options

* [-wan](https://www.consul.io/docs/commands/join.html" \l "wan) - For agents running in server mode, the agent will attempt to join other servers gossiping in a WAN cluster. This is used to form a bridge between multiple datacenters.

# Consul Keygen

Command: consul keygen

The keygen command generates an encryption key that can be used for [Consul agent traffic encryption](https://www.consul.io/docs/agent/encryption.html). The keygen command uses a cryptographically strong pseudo-random number generator to generate the key.

# Consul Keyring

Command: consul keyring

The keyring command is used to examine and modify the encryption keys used in Consul's [Gossip Pools](https://www.consul.io/docs/internals/gossip.html). It is capable of distributing new encryption keys to the cluster, retiring old encryption keys, and changing the keys used by the cluster to encrypt messages.

Consul allows multiple encryption keys to be in use simultaneously. This is intended to provide a transition state while the cluster converges. It is the responsibility of the operator to ensure that only the required encryption keys are installed on the cluster. You can review the installed keys using the -list argument, and remove unneeded keys with -remove.

All operations performed by this command can only be run against server nodes, and affect both the LAN and WAN keyrings in lock-step.

All variations of the keyring command return 0 if all nodes reply and there are no errors. If any node fails to reply or reports failure, the exit code will be 1.

## [»](https://www.consul.io/docs/commands/keyring.html#usage)Usage

Usage: consul keyring [options]

Only one actionable argument may be specified per run, including -list, -install, -remove, and -use.

#### [»](https://www.consul.io/docs/commands/keyring.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/keyring.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/keyring.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/keyring.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/keyring.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/keyring.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/keyring.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/keyring.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/keyring.html#command-options)Command Options

* [-list](https://www.consul.io/docs/commands/keyring.html#list) - List all keys currently in use within the cluster.

* [-install](https://www.consul.io/docs/commands/keyring.html" \l "install) - Install a new encryption key. This will broadcast the new key to all members in the cluster.

* [-use](https://www.consul.io/docs/commands/keyring.html" \l "use) - Change the primary encryption key, which is used to encrypt messages. The key must already be installed before this operation can succeed.

* [-remove](https://www.consul.io/docs/commands/keyring.html" \l "remove) - Remove the given key from the cluster. This operation may only be performed on keys which are not currently the primary key.

* [-relay-factor](https://www.consul.io/docs/commands/keyring.html" \l "relay-factor) - Added in Consul 0.7.4, setting this to a non-zero value will cause nodes to relay their response to the operation through this many randomly-chosen other nodes in the cluster. The maximum allowed value is 5.

## [»](https://www.consul.io/docs/commands/keyring.html#output)Output

The output of the consul keyring -list command consolidates information from all nodes and all datacenters to provide a simple and easy to understand view of the cluster. The following is some example output from a cluster with two datacenters, each which consist of one server and one client:

==> Gathering installed encryption keys...

==> Done!

WAN:

a1i101sMY8rxB+0eAKD/gw== [2/2]

dc2 (LAN):

a1i101sMY8rxB+0eAKD/gw== [2/2]

dc1 (LAN):

a1i101sMY8rxB+0eAKD/gw== [2/2]

dc1 (LAN) [alpha]:

a1i101sMY8rxB+0eAKD/gw== [2/2]

As you can see, the output above is divided first by gossip pool, including any network segments, and then by encryption key. The indicator to the right of each key displays the number of nodes the key is installed on over the total number of nodes in the pool.

## [»](https://www.consul.io/docs/commands/keyring.html" \l "errors)Errors

If any errors are encountered while performing a keyring operation, no key information is displayed, but instead only error information. The error information is arranged in a similar fashion, organized first by datacenter, followed by a simple list of nodes which had errors, and the actual text of the error. Below is sample output from the same cluster as above, if we try to do something that causes an error; in this case, trying to remove the primary key:

==> Removing gossip encryption key...

dc1 (LAN) error: 2/2 nodes reported failure

server1: Removing the primary key is not allowed

client1: Removing the primary key is not allowed

WAN error: 2/2 nodes reported failure

server1.dc1: Removing the primary key is not allowed

server2.dc2: Removing the primary key is not allowed

dc2 (LAN) error: 2/2 nodes reported failure

server2: Removing the primary key is not allowed

client2: Removing the primary key is not allowed

As you can see, each node with a failure reported what went wrong.

# Consul KV

Command: consul kv

The kv command is used to interact with Consul's KV store via the command line. It exposes top-level commands for inserting, updating, reading, and deleting from the store. This command is available in Consul 0.7.1 and later.

The KV store is also accessible via the [HTTP API](https://www.consul.io/api/kv.html).

## [»](https://www.consul.io/docs/commands/kv.html#usage)Usage

Usage: consul kv <subcommand>

For the exact documentation for your Consul version, run consul kv -h to view the complete list of subcommands.

Usage: consul kv <subcommand> [options] [args]

# ...

Subcommands:

delete Removes data from the KV store

export Exports part of the KV tree in JSON format

get Retrieves or lists data from the KV store

import Imports part of the KV tree in JSON format

put Sets or updates data in the KV store

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar or one of the links below:

* [delete](https://www.consul.io/docs/commands/kv/delete.html)
* [export](https://www.consul.io/docs/commands/kv/export.html)
* [get](https://www.consul.io/docs/commands/kv/get.html)
* [import](https://www.consul.io/docs/commands/kv/import.html)
* [put](https://www.consul.io/docs/commands/kv/put.html)

## [»](https://www.consul.io/docs/commands/kv.html#basic-examples)Basic Examples

To create or update the key named "redis/config/connections" to the value "5" in Consul's KV store:

$ consul kv put redis/config/connections 5

Success! Data written to: redis/config/connections

To read a value back from Consul:

$ consul kv get redis/config/connections

5

Or you can query for detailed information:

$ consul kv get -detailed redis/config/connections

CreateIndex 336

Flags 0

Key redis/config/connections

LockIndex 0

ModifyIndex 336

Session -

Value 5

Finally, deleting a key is just as easy:

$ consul kv delete redis/config/connections

Success! Data deleted at key: redis/config/connections

For more examples, ask for subcommand help or view the subcommand documentation by clicking on one of the links in the sidebar.

# Consul KV Delete

Command: consul kv delete

The kv delete command removes the value from Consul's KV store at the given path. If no key exists at the path, no action is taken.

## [»](https://www.consul.io/docs/commands/kv/delete.html#usage)Usage

Usage: consul kv delete [options] KEY\_OR\_PREFIX

#### [»](https://www.consul.io/docs/commands/kv/delete.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/kv/delete.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/kv/delete.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/kv/delete.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/kv/delete.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/kv/delete.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/kv/delete.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/kv/delete.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/kv/delete.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/kv/delete.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/kv/delete.html" \l "kv-delete-options)KV Delete Options

* [-cas](https://www.consul.io/docs/commands/kv/delete.html" \l "cas) - Perform a Check-And-Set operation. Specifying this value also requires the -modify-index flag to be set. The default value is false.

* [-modify-index=<int>](https://www.consul.io/docs/commands/kv/delete.html" \l "modify-index-lt-int-gt-) - Unsigned integer representing the ModifyIndex of the key. This is used in combination with the -cas flag.

* [-recurse](https://www.consul.io/docs/commands/kv/delete.html" \l "recurse) - Recursively delete all keys with the path. The default value is false.

## [»](https://www.consul.io/docs/commands/kv/delete.html#examples)Examples

To remove the value for the key named "redis/config/connections" in the KV store:

$ consul kv delete redis/config/connections

Success! Deleted key: redis/config/connections

If the key does not exist, the command will not error, and a success message will be returned:

$ consul kv delete not-a-real-key

Success! Deleted key: not-a-real-key

To only delete a key if it has not been modified since a given index, specify the -cas and -modify-index flags:

$ consul kv get -detailed redis/config/connections | grep ModifyIndex

ModifyIndex 456

$ consul kv delete -cas -modify-index=123 redis/config/connections

Error! Did not delete key redis/config/connections: CAS failed

$ consul kv delete -cas -modify-index=456 redis/config/connections

Success! Deleted key: redis/config/connections

To recursively delete all keys that start with a given prefix, specify the -recurse flag:

$ consul kv delete -recurse redis/

Success! Deleted keys with prefix: redis/

**Trailing slashes are important** in the recursive delete operation, since Consul performs a greedy match on the provided prefix. If you were to use "foo" as the key, this would recursively delete any key starting with those letters such as "foo", "food", and "football" not just "foo". To ensure you are deleting a folder, always use a trailing slash.

It is not valid to combine the -cas option with -recurse, since you are deleting multiple keys under a prefix in a single operation:

$ consul kv delete -cas -recurse redis/

Cannot specify both -cas and -recurse!

# Consul KV Export

Command: consul kv export

The kv export command is used to retrieve KV pairs for the given prefix from Consul's KV store, and write a JSON representation to stdout. This can be used with the command "consul kv import" to move entire trees between Consul clusters.

## [»](https://www.consul.io/docs/commands/kv/export.html#usage)Usage

Usage: consul kv export [options] [PREFIX]

#### [»](https://www.consul.io/docs/commands/kv/export.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/kv/export.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/kv/export.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/kv/export.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/kv/export.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/kv/export.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/kv/export.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/kv/export.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/kv/export.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/kv/export.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

## [»](https://www.consul.io/docs/commands/kv/export.html#examples)Examples

To export the tree at "vault/" in the key value store:

$ consul kv export vault/

# JSON output

# Consul KV Get

Command: consul kv get

The kv get command is used to retrieve the value from Consul's KV store at the given key name. If no key exists with that name, an error is returned. If a key exists with that name but has no data, nothing is returned. A key name or prefix is required.

## [»](https://www.consul.io/docs/commands/kv/get.html#usage)Usage

Usage: consul kv get [options] [KEY\_OR\_PREFIX]

#### [»](https://www.consul.io/docs/commands/kv/get.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/kv/get.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/kv/get.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/kv/get.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/kv/get.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/kv/get.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/kv/get.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/kv/get.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/kv/get.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/kv/get.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/kv/get.html" \l "kv-get-options)KV Get Options

* [-base64](https://www.consul.io/docs/commands/kv/get.html" \l "base64) - Base 64 encode the value. The default value is false.
* [-detailed](https://www.consul.io/docs/commands/kv/get.html#detailed) - Provide additional metadata about the key in addition to the value such as the ModifyIndex and any flags that may have been set on the key. The default value is false.

* [-keys](https://www.consul.io/docs/commands/kv/get.html" \l "keys) - List keys which start with the given prefix, but not their values. This is especially useful if you only need the key names themselves. This option is commonly combined with the -separator option. The default value is false.
* [-recurse](https://www.consul.io/docs/commands/kv/get.html#recurse) - Recursively look at all keys prefixed with the given path. The default value is false.

* [-separator=<string>](https://www.consul.io/docs/commands/kv/get.html" \l "separator-lt-string-gt-) - String to use as a separator for recursive lookups. The default value is "/", and only used when paired with the -keys flag. This will limit the prefix of keys returned, only up to the given separator.

## [»](https://www.consul.io/docs/commands/kv/get.html#examples)Examples

To retrieve the value for the key named "redis/config/connections" in the KV store:

$ consul kv get redis/config/connections

5

This will return the original, raw value stored in Consul. To view detailed information about the key, specify the "-detailed" flag. This will output all known metadata about the key including ModifyIndex and any user-supplied flags:

$ consul kv get -detailed redis/config/connections

CreateIndex 336

Flags 0

Key redis/config/connections

LockIndex 0

ModifyIndex 336

Session -

Value 5

If the key with the given name does not exist, an error is returned:

$ consul kv get not-a-real-key

Error! No key exists at: not-a-real-key

To treat the path as a prefix and list all keys which start with the given prefix, specify the "-recurse" flag:

$ consul kv get -recurse redis/

redis/config/connections:5

redis/config/cpu:128

redis/config/memory:512

Or list detailed information about all pairs under a prefix:

$ consul kv get -recurse -detailed redis

CreateIndex 336

Flags 0

Key redis/config/connections

LockIndex 0

ModifyIndex 336

Session -

Value 5

CreateIndex 472

Flags 0

Key redis/config/cpu

LockIndex 0

ModifyIndex 472

Session -

Value 128

CreateIndex 471

Flags 0

Key redis/config/memory

LockIndex 0

ModifyIndex 471

Session -

Value 512

To just list the keys which start with the specified prefix, use the "-keys" option instead. This is more performant and results in a smaller payload:

$ consul kv get -keys redis/config/

redis/config/connections

redis/config/cpu

redis/config/memory

By default, the -keys operation uses a separator of "/", meaning it will not recurse beyond that separator. You can choose a different separator by setting -separator="<string>".

$ consul kv get -keys -separator="c" redis

redis/c

Alternatively, you can disable the separator altogether by setting it to the empty string:

$ consul kv get -keys -separator="" redis

redis/config/connections

redis/config/cpu

redis/config/memory

To list all keys at the root, simply omit the prefix parameter:

$ consul kv get -keys

memcached/

redis/

# Consul KV Import

Command: consul kv import

The kv import command is used to import KV pairs from the JSON representation generated by the kv exportcommand.

## [»](https://www.consul.io/docs/commands/kv/import.html#usage)Usage

Usage: consul kv import [options] [DATA]

#### [»](https://www.consul.io/docs/commands/kv/import.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/kv/import.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/kv/import.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/kv/import.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/kv/import.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/kv/import.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/kv/import.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/kv/import.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/kv/import.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/kv/import.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

## [»](https://www.consul.io/docs/commands/kv/import.html#examples)Examples

To import from a file, prepend the filename with @:

$ consul kv import @values.json

# Output

To import from stdin, use - as the data parameter:

$ cat values.json | consul kv import -

# Output

You can also pass the JSON directly, however care must be taken with shell escaping:

$ consul kv import "$(cat values.json)"

# Output

# Consul KV Put

Command: consul kv put

The kv put command writes the data to the given path in the KV store.

## [»](https://www.consul.io/docs/commands/kv/put.html#usage)Usage

Usage: consul kv put [options] KEY [DATA]

#### [»](https://www.consul.io/docs/commands/kv/put.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/kv/put.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/kv/put.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/kv/put.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/kv/put.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/kv/put.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/kv/put.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/kv/put.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/kv/put.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/kv/put.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/kv/put.html" \l "kv-put-options)KV Put Options

* [-acquire](https://www.consul.io/docs/commands/kv/put.html" \l "acquire) - Obtain a lock on the key. If the key does not exist, this operation will create the key and obtain the lock. The session must already exist and be specified via the -session flag. The default value is false.
* [-base64](https://www.consul.io/docs/commands/kv/put.html#base64) - Treat the data as base 64 encoded. The default value is false.
* [-cas](https://www.consul.io/docs/commands/kv/put.html#cas) - Perform a Check-And-Set operation. Specifying this value also requires the -modify-index flag to be set. The default value is false.

* [-flags=<int>](https://www.consul.io/docs/commands/kv/put.html" \l "flags-lt-int-gt-) - Unsigned integer value to assign to this KV pair. This value is not read by Consul, so clients can use this value however makes sense for their use case. The default value is 0 (no flags).
* [-modify-index=<int>](https://www.consul.io/docs/commands/kv/put.html#modify-index-lt-int-gt-) - Unsigned integer representing the ModifyIndex of the key. This is used in combination with the -cas flag.

* [-release](https://www.consul.io/docs/commands/kv/put.html" \l "release) - Forfeit the lock on the key at the given path. This requires the -session flag to be set. The key must be held by the session in order to be unlocked. The default value is false.

* [-session=<string>](https://www.consul.io/docs/commands/kv/put.html" \l "session-lt-string-gt-) - User-defined identifer for this session as a string. This is commonly used with the -acquire and -release operations to build robust locking, but it can be set on any key. The default value is empty (no session).

## [»](https://www.consul.io/docs/commands/kv/put.html#examples)Examples

To insert a value of "5" for the key named "redis/config/connections" in the KV store:

$ consul kv put redis/config/connections 5

Success! Data written to: redis/config/connections

If no data is specified, the key will be created with empty data:

$ consul kv put redis/config/connections

Success! Data written to: redis/config/connections

If the -base64 flag is set, the data will be decoded before writing:

$ consul kv put -base64 foo/encoded aGVsbG8gd29ybGQK

Success! Data written to: foo/encoded

**Be careful when overwriting data!** The above operation would overwrite the value at the key to the empty value.

For longer or sensitive values, it is possible to read from a file by prefixing with the @ symbol:

$ consul kv put redis/config/password @password.txt

Success! Data written to: redis/config/connections

Or read values from stdin by specifying the - symbol:

$ echo "5" | consul kv put redis/config/password -

Success! Data written to: redis/config/connections

$ consul kv put redis/config/password -

5

<CTRL+D>

Success! Data written to: redis/config/connections

For secret and sensitive values, you should consider using a secret management solution like [**HashiCorp's Vault**](https://www.vaultproject.io/). While it is possible to secure values in Consul's KV store, Vault provides a more robust interface for secret management.

To only update a key if it has not been modified since a given index, specify the -cas and -modify-index flags:

$ consul kv get -detailed redis/config/connections | grep ModifyIndex

ModifyIndex 456

$ consul kv put -cas -modify-index=123 redis/config/connections 10

Error! Did not write to redis/config/connections: CAS failed

$ consul kv put -cas -modify-index=456 redis/config/connections 10

Success! Data written to: redis/config/connections

To specify flags on the key, use the -flags option. These flags are completely controlled by the user:

$ consul kv put -flags=42 redis/config/password s3cr3t

Success! Data written to: redis/config/password

To create or tune a lock, use the -acquire and -session flags. The session must already exist (this command will not create it or manage it):

$ consul kv put -acquire -session=abc123 redis/lock/update

Success! Lock acquired on: redis/lock/update

When you are finished, release the lock:

$ consul kv put -release -session=acb123 redis/lock/update

Success! Lock released on: redis/lock/update

**Warning!** If you are trying to build a locking mechanism with these low-level primitives, you may want to look at the [consul lock](https://www.consul.io/docs/commands/lock.html) command. It provides higher-level functionality without exposing the internal APIs of Consul.

# Consul Leave

Command: consul leave

The leave command triggers a graceful leave and shutdown of the agent. It is used to ensure other nodes see the agent as "left" instead of "failed". Nodes that leave will not attempt to re-join the cluster on restarting with a snapshot.

For nodes in server mode, the node is removed from the Raft peer set in a graceful manner. This is critical, as in certain situations a non-graceful leave can affect cluster availability.

Running consul leave on a server explicitly will reduce the quorum size. Even if the cluster used bootstrap\_expect to set a quorum size initially, issuing consul leave on a server will reconfigure the cluster to have fewer servers. This means you could end up with just one server that is still able to commit writes because quorum is only 1, but those writes might be lost if that server fails before more are added.

## [»](https://www.consul.io/docs/commands/leave.html#usage)Usage

Usage: consul leave [options]

#### [»](https://www.consul.io/docs/commands/leave.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/leave.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/leave.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/leave.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/leave.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/leave.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/leave.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/leave.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

# Consul License

Command: consul license

**ENTERPRISE**

This feature requires [Consul Enterprise](https://www.hashicorp.com/products/consul/)

The license command provides datacenter-level management of the Consul Enterprise license. This was added in Consul 1.1.0.

If ACLs are enabled then a token with operator privileges may be required in order to use this command. Requests are forwarded internally to the leader if required, so this can be run from any Consul node in a cluster. See the [ACL Guide](https://www.consul.io/docs/guides/acl.html#operator) for more information.

Usage: consul license <subcommand> [options] [args]

This command has subcommands for managing the Consul Enterprise license

Here are some simple examples, and more detailed examples are

available in the subcommands or the documentation.

Install a new license from a file:

$ consul license put @consul.license

Install a new license from stdin:

$ consul license put -

Install a new license from a string:

$ consul license put "<license blob>"

Retrieve the current license:

$ consul license get

For more examples, ask for subcommand help or view the documentation.

Subcommands:

get Get the current license

put Puts a new license in the datacenter

## [»](https://www.consul.io/docs/commands/license.html" \l "put)put

This command sets the Consul Enterprise license.

Usage: consul license put [options] LICENSE

#### [»](https://www.consul.io/docs/commands/license.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/license.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/license.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/license.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/license.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/license.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/license.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/license.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/license.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/license.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

The output looks like this:

License is valid

License ID: 2afbf681-0d1a-0649-cb6c-333ec9f0989c

Customer ID: 0259271d-8ffc-e85e-0830-c0822c1f5f2b

Expires At: 2019-05-22 03:59:59.999 +0000 UTC

Datacenter: \*

Package: premium

Licensed Features:

Automated Backups

Automated Upgrades

Enhanced Read Scalability

Network Segments

Redundancy Zone

Advanced Network Federation

## [»](https://www.consul.io/docs/commands/license.html" \l "get)get

This command gets the Consul Enterprise license.

Usage: consul license get [options]

#### [»](https://www.consul.io/docs/commands/license.html#api-options-1)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/license.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/license.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/license.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/license.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/license.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/license.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/license.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/license.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/license.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

The output looks like this:

License is valid

License ID: 2afbf681-0d1a-0649-cb6c-333ec9f0989c

Customer ID: 0259271d-8ffc-e85e-0830-c0822c1f5f2b

Expires At: 2019-05-22 03:59:59.999 +0000 UTC

Datacenter: \*

Package: premium

Licensed Features:

Automated Backups

Automated Upgrades

Enhanced Read Scalability

Network Segments

Redundancy Zone

Advanced Network Federation

# Consul Lock

Command: consul lock

The lock command provides a mechanism for simple distributed locking. A lock (or semaphore) is created at a given prefix in the KV store, and only when held, is a child process invoked. If the lock is lost or communication is disrupted, the child process is terminated.

The number of lock holders is configurable with the -n flag. By default, a single holder is allowed, and a lock is used for mutual exclusion. This uses the [leader election algorithm](https://www.consul.io/docs/guides/leader-election.html).

If the lock holder count is more than one, then a semaphore is used instead. A semaphore allows more than a single holder, but this is less efficient than a simple lock. This follows the [semaphore algorithm](https://www.consul.io/docs/guides/semaphore.html).

All locks using the same prefix must agree on the value of -n. If conflicting values of -n are provided, an error will be returned.

An example use case is for highly-available N+1 deployments. In these cases, if N instances of a service are required, N+1 are deployed and use consul lock with -n=N to ensure only N instances are running. For singleton services, a hot standby waits until the current leader fails to take over.

## [»](https://www.consul.io/docs/commands/lock.html#usage)Usage

Usage: consul lock [options] prefix child...

The only required options are the key prefix and the command to execute. The prefix must be writable. The child is invoked only when the lock is held, and the CONSUL\_LOCK\_HELD environment variable will be set to true.

If the lock is lost, communication is disrupted, or the parent process interrupted, the child process will receive a SIGTERM. After a grace period of 5 seconds, a SIGKILL will be used to force termination. For Consul agents on Windows, the child process is always terminated with a SIGKILL, since Windows has no POSIX compatible notion for SIGTERM.

#### [»](https://www.consul.io/docs/commands/lock.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/lock.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/lock.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/lock.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/lock.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/lock.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/lock.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/lock.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/lock.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/lock.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/lock.html#command-options)Command Options

* [-child-exit-code](https://www.consul.io/docs/commands/lock.html" \l "child-exit-code) - Exit 2 if the child process exited with an error if this is true, otherwise this doesn't propagate an error from the child. The default value is false.

* [-monitor-retry](https://www.consul.io/docs/commands/lock.html" \l "monitor-retry) - Retry up to this number of times if Consul returns a 500 error while monitoring the lock. This allows riding out brief periods of unavailability without causing leader elections, but increases the amount of time required to detect a lost lock in some cases. Defaults to 3, with a 1s wait between retries. Set to 0 to disable.

* [-n](https://www.consul.io/docs/commands/lock.html" \l "n) - Optional, limit of lock holders. Defaults to 1. The underlying implementation switches from a lock to a semaphore when increased past one. All locks on the same prefix must use the same value.
* [-name](https://www.consul.io/docs/commands/lock.html#name) - Optional name to associate with the underlying session. If not provided, one is generated based on the child command.
* [-shell](https://www.consul.io/docs/commands/lock.html#shell) - Optional, use a shell to run the command (can set a custom shell via the SHELL environment variable). The default value is true.

* [-pass-stdin](https://www.consul.io/docs/commands/lock.html" \l "pass-stdin) - Pass stdin to child process.

* [-timeout](https://www.consul.io/docs/commands/lock.html" \l "timeout) - Maximum amount of time to wait to acquire the lock, specified as a duration like 1s or 3h. The default value is 0.

* [-try](https://www.consul.io/docs/commands/lock.html" \l "try) - Attempt to acquire the lock up to the given timeout. The timeout is a positive decimal number, with unit suffix, such as "500ms". Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", "h".
* [-verbose](https://www.consul.io/docs/commands/lock.html#verbose) - Enables verbose output.

## [»](https://www.consul.io/docs/commands/lock.html" \l "shell-1)SHELL

Consul lock launches its children in a shell. By default, Consul will use the shell defined in the environment variable SHELL. If SHELL is not defined, it will default to /bin/sh. It should be noted that not all shells terminate child processes when they receive SIGTERM. Under Ubuntu, /bin/sh is linked to dash, which does **not** terminate its children. In order to ensure that child processes are killed when the lock is lost, be sure to set the SHELL environment variable appropriately, or run without a shell by setting -shell=false.

# Consul Maint

Command: consul maint

The maint command provides control of service maintenance mode. Using the command, it is possible to mark a service provided by a node or all the services on the node as a whole as "under maintenance". In this mode of operation, the service will not appear in DNS query results, or API results. This effectively takes the service out of the pool of available "healthy" nodes of a service.

Under the hood, maintenance mode is activated by registering a health check in critical status against a service, and deactivated by deregistering the health check.

## [»](https://www.consul.io/docs/commands/maint.html#usage)Usage

Usage: consul maint [options]

#### [»](https://www.consul.io/docs/commands/maint.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/maint.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/maint.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/maint.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/maint.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/maint.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/maint.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/maint.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/maint.html#command-options)Command Options

* [-enable](https://www.consul.io/docs/commands/maint.html" \l "enable) - Enable maintenance mode on all services on a node. If combined with the -service flag, we operate on a specific service ID.

* [-disable](https://www.consul.io/docs/commands/maint.html" \l "disable) - Disable maintenance mode on all services on a node. If combined with the -service flag, we operate on a specific service ID.

* [-reason](https://www.consul.io/docs/commands/maint.html" \l "reason) - An optional reason for placing the service into maintenance mode. If provided, this reason will be visible in the newly- registered critical check's "Notes" field.
* [-service](https://www.consul.io/docs/commands/maint.html#service) - An optional service ID to control maintenance mode for a given service. By providing this flag, the -enable and -disable flags functionality is modified to operate on the given service ID.

## [»](https://www.consul.io/docs/commands/maint.html" \l "list-mode)List mode

If neither -enable nor -disable are passed, the maint command will switch to "list mode", displaying any current maintenances. This may return blank if nothing is currently under maintenance. The output will look like:

$ consul maint

Node:

Name: node1.local

Reason: This node is broken.

Service:

ID: redis

Reason: Redis is currently offline.

# Consul Members

Command: consul members

The members command outputs the current list of members that a Consul agent knows about, along with their state. The state of a node can only be "alive", "left", or "failed".

Nodes in the "failed" state are still listed because Consul attempts to reconnect with failed nodes for a certain amount of time in the case that the failure is actually just a network partition.

## [»](https://www.consul.io/docs/commands/members.html#usage)Usage

Usage: consul members [options]

#### [»](https://www.consul.io/docs/commands/members.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/members.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/members.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/members.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/members.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/members.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/members.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/members.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/members.html#command-options)Command Options

* [-detailed](https://www.consul.io/docs/commands/members.html#detailed) - If provided, output shows more detailed information about each node.

* [-segment](https://www.consul.io/docs/commands/members.html" \l "segment) - (Enterprise-only) The segment to show members in. If not provided, members in all segments visible to the agent will be listed.

* [-status](https://www.consul.io/docs/commands/members.html" \l "status) - If provided, output is filtered to only nodes matching the regular expression for status
* [-wan](https://www.consul.io/docs/commands/members.html#wan) - For agents in Server mode, this will return the list of nodes in the WAN gossip pool. These are generally all the server nodes in each datacenter.

# Consul Monitor

Command: consul monitor

The monitor command is used to connect and follow the logs of a running Consul agent. Monitor will show the recent logs and then continue to follow the logs, not exiting until interrupted or until the remote agent quits.

The power of the monitor command is that it allows you to log the agent at a relatively high log level (such as "warn"), but still access debug logs and watch the debug logs if necessary.

## [»](https://www.consul.io/docs/commands/monitor.html#usage)Usage

Usage: consul monitor [options]

#### [»](https://www.consul.io/docs/commands/monitor.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/monitor.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/monitor.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/monitor.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/monitor.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/monitor.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/monitor.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/monitor.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/monitor.html#command-options)Command Options

* [-log-level](https://www.consul.io/docs/commands/monitor.html#log-level) - The log level of the messages to show. By default this is "info". This log level can be more verbose than what the agent is configured to run at. Available log levels are "trace", "debug", "info", "warn", and "err".

# Consul Operator

Command: consul operator

The operator command provides cluster-level tools for Consul operators, such as interacting with the Raft subsystem. This was added in Consul 0.7.

Use this command with extreme caution, as improper use could lead to a Consul outage and even loss of data.

If ACLs are enabled then a token with operator privileges may be required in order to use this command. Requests are forwarded internally to the leader if required, so this can be run from any Consul node in a cluster. See the [ACL Guide](https://www.consul.io/docs/guides/acl.html#operator) for more information.

See the [Outage Recovery](https://www.consul.io/docs/guides/outage.html) guide for some examples of how this command is used. For an API to perform these operations programmatically, please see the documentation for the [Operator](https://www.consul.io/api/operator.html) endpoint.

## [»](https://www.consul.io/docs/commands/operator.html#usage)Usage

Usage: consul operator <subcommand> [options]

# ...

Subcommands:

area Provides tools for working with network areas (Enterprise-only)

autopilot Provides tools for modifying Autopilot configuration

raft Provides cluster-level tools for Consul operators

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar or one of the links below:

* [area](https://www.consul.io/docs/commands/operator/area.html)
* [autopilot](https://www.consul.io/docs/commands/operator/autopilot.html)
* [raft](https://www.consul.io/docs/commands/operator/raft.html)

# Consul Operator Area

Command: consul operator area

**ENTERPRISE**

This feature requires [Consul Enterprise](https://www.hashicorp.com/products/consul/)

Consul Enterprise version supports network areas, which are operator-defined relationships between servers in two different Consul datacenters. The operator area command is used to interact with Consul's network area subsystem.

Unlike Consul's WAN feature, network areas use just the server RPC port for communication, and relationships can be made between independent pairs of datacenters, so not all servers need to be fully connected. This allows for complex topologies among Consul datacenters like hub/spoke and more general trees.

See the [Network Areas Guide](https://www.consul.io/docs/guides/areas.html) for more details.

Usage: consul operator area <subcommand> [options]

The operator area command is used to interact with Consul's network area

subsystem. Network areas are used to link together Consul servers in different

Consul datacenters. With network areas, Consul datacenters can be linked

together in ways other than a fully-connected mesh, as is required for Consul's

WAN.

Subcommands:

create Create a new network area

delete Remove a network area

join Join Consul servers into an existing network area

list List network areas

members Display Consul server members present in network areas

update Update the configuration of a network area

If ACLs are enabled, the client will need to supply an ACL Token with operator read or write privileges to use these commands.

## [»](https://www.consul.io/docs/commands/operator/area.html#create)create

This command creates a new network area.

Usage: consul operator area create [options]

#### [»](https://www.consul.io/docs/commands/operator/area.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/operator/area.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/operator/area.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/operator/area.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/operator/area.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/operator/area.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/operator/area.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/operator/area.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/operator/area.html#command-options)Command Options

* [-peer-datacenter=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "peer-datacenter-lt-value-gt-) - Declares the peer Consul datacenter that will make up the other side of this network area. Network areas always involve a pair of datacenters: the datacenter where the area was created, and the peer datacenter. This is required.

* [-retry-join=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "retry-join-lt-value-gt-) Specifies the address of a Consul server to join to, such as an IP or hostname with an optional port number. This is optional and can be specified multiple times.

* [-use-tls=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "use-tls-lt-value-gt-) Specifies whether gossip over this area should be encrypted with TLS if possible. Must be either true or false.

The output looks like this, displaying the ID of the newly-created network area:

Created area "d2872ec5-68ea-b862-b75d-0bee99aca100" with peer datacenter "other"!

The return code will indicate success or failure.

## [»](https://www.consul.io/docs/commands/operator/area.html#delete)delete

This command deletes an existing network area.

Usage: consul operator area delete [options]

#### [»](https://www.consul.io/docs/commands/operator/area.html#api-options-1)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/operator/area.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/operator/area.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/operator/area.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/operator/area.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/operator/area.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/operator/area.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/operator/area.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/operator/area.html" \l "command-options-1)Command Options

* [-id=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "id-lt-value-gt-) - Looks up the area to operate on by its ID. This can be given instead of a peer datacenter.

* [-peer-datacenter=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "peer-datacenter-lt-value-gt--1) - Looks up the area to operate on by its peer datacenter. This can be given instead of an ID.

The output looks like this:

Deleted area "154941b0-80e2-9d69-c560-ab2c02807332"!

The return code will indicate success or failure.

## [»](https://www.consul.io/docs/commands/operator/area.html#join)join

This command joins Consul servers into an existing network area by address, such as an IP or hostname with an optional port. Multiple addresses may be given.

Usage: consul operator area join [options] ADDRESSES

#### [»](https://www.consul.io/docs/commands/operator/area.html" \l "api-options-2)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/operator/area.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/operator/area.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/operator/area.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/operator/area.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/operator/area.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/operator/area.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/operator/area.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/operator/area.html" \l "command-options-2)Command Options

* [-id=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "id-lt-value-gt--1) - Looks up the area to operate on by its ID. This can be given instead of a peer datacenter.

* [-peer-datacenter=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "peer-datacenter-lt-value-gt--2) - Looks up the area to operate on by its peer datacenter. This can be given instead of an ID.

The output looks like this:

Address Joined Error

10.1.2.3 false failed to connect to "10.1.2.3:8300": dial tcp 10.1.2.3:8300: i/o timeout

10.1.2.4 true (none)

10.1.2.5 true (none)

The Error field will have a human-readable error message if Consul was unable to join the given address.

The return code will indicate success or failure.

## [»](https://www.consul.io/docs/commands/operator/area.html#list)list

This command lists all network areas.

Usage: consul operator area list [options]

#### [»](https://www.consul.io/docs/commands/operator/area.html" \l "api-options-3)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/operator/area.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/operator/area.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/operator/area.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/operator/area.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/operator/area.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/operator/area.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/operator/area.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

The output looks like this:

Area PeerDC RetryJoin

6a52a0af-62e2-dad4-da60-e66acc37096c dc2 10.1.2.3,10.1.2.4,10.1.2.5

96e33424-f5ce-9fcd-ecab-27974e36678f other (none)

Area is the ID of the network area.

PeerDC is the peer datacenter for the area.

RetryJoin is the list of servers to join, defined when the area was created.

The return code will indicate success or failure.

## [»](https://www.consul.io/docs/commands/operator/area.html#members)members

This command displays Consul server nodes present in a network area, or all areas if no area is specified.

Usage: consul operator area members [options]

#### [»](https://www.consul.io/docs/commands/operator/area.html" \l "api-options-4)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/operator/area.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/operator/area.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/operator/area.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/operator/area.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/operator/area.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/operator/area.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/operator/area.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/operator/area.html" \l "command-options-3)Command Options

* [-id=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "id-lt-value-gt--2) - Looks up the area to operate on by its ID. This can be given instead of a peer datacenter.

* [-peer-datacenter=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "peer-datacenter-lt-value-gt--3) - Looks up the area to operate on by its peer datacenter. This can be given instead of an ID.

The output looks like this:

Area Node Address Status Build Protocol DC RTT

6a52a0af-62e2-dad4-da60-e66acc37096c node-1.dc1 127.0.0.1:8300 alive 0.8.0 2 dc1 0s

6a52a0af-62e2-dad4-da60-e66acc37096c node-2.dc1 127.0.0.2:8300 alive 0.8.0 2 dc1 594.191µs

96e33424-f5ce-9fcd-ecab-27974e36678f node-1.dc1 127.0.0.1:8300 alive 0.8.0 2 dc1 0s

96e33424-f5ce-9fcd-ecab-27974e36678f node-2.dc1 127.0.0.2:8300 alive 0.8.0 2 dc1 634.109µs

Area is the ID of the network area.

Node is the name of the node.

Address is the IP and server RPC port for the node.

Status is the current health status of the node, as determined by the network area distributed failure detector. This will be "alive", "leaving", "left", or "failed". A "failed" status means that other servers are not able to probe this server over its server RPC interface.

Build has the Consul version running on the node.

Protocol is the [protocol version](https://www.consul.io/docs/upgrading.html#protocol-versions) being spoken by the node.

DC is the node's Consul datacenter.

RTT is an estimated network round trip time from the server answering the query to the given server, in a human-readable format. This is computed using [network coordinates](https://www.consul.io/docs/internals/coordinates.html).

The return code will indicate success or failure.

## [»](https://www.consul.io/docs/commands/operator/area.html#update)update

This command updates the configuration of network area.

Usage: consul operator area update [options]

#### [»](https://www.consul.io/docs/commands/operator/area.html" \l "api-options-5)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/operator/area.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/operator/area.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/operator/area.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/operator/area.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/operator/area.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/operator/area.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/operator/area.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/operator/area.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/operator/area.html" \l "command-options-4)Command Options

* [-id=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "id-lt-value-gt--3) - Looks up the area to operate on by its ID. This can be given instead of a peer datacenter.

* [-peer-datacenter=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "peer-datacenter-lt-value-gt--4) - Declares the peer Consul datacenter that will make up the other side of this network area. Network areas always involve a pair of datacenters: the datacenter where the area was created, and the peer datacenter. This is required.

* [-use-tls=<value>](https://www.consul.io/docs/commands/operator/area.html" \l "use-tls-lt-value-gt--1) Specifies whether gossip over this area should be encrypted with TLS if possible. Must be either true or false.

The output looks like this:

Updated area "d2872ec5-68ea-b862-b75d-0bee99aca100"

The return code will indicate success or failure.

# Consul Operator Autopilot

Command: consul operator autopilot

The Autopilot operator command is used to interact with Consul's Autopilot subsystem. The command can be used to view or modify the current Autopilot configuration. See the [Autopilot Guide](https://www.consul.io/docs/guides/autopilot.html) for more information about Autopilot.

Usage: consul operator autopilot <subcommand> [options]

The Autopilot operator command is used to interact with Consul's Autopilot

subsystem. The command can be used to view or modify the current configuration.

Subcommands:

get-config Display the current Autopilot configuration

set-config Modify the current Autopilot configuration

## [»](https://www.consul.io/docs/commands/operator/autopilot.html#get-config)get-config

This command displays the current Raft peer configuration.

Usage: consul operator autopilot get-config [options]

#### [»](https://www.consul.io/docs/commands/operator/autopilot.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/operator/autopilot.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/operator/autopilot.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/operator/autopilot.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

The output looks like this:

CleanupDeadServers = true

LastContactThreshold = 200ms

MaxTrailingLogs = 250

ServerStabilizationTime = 10s

RedundancyZoneTag = ""

DisableUpgradeMigration = false

UpgradeMigrationTag = ""

## [»](https://www.consul.io/docs/commands/operator/autopilot.html#set-config)set-config

Modifies the current Autopilot configuration.

Usage: consul operator autopilot set-config [options]

#### [»](https://www.consul.io/docs/commands/operator/autopilot.html#api-options-1)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/operator/autopilot.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/operator/autopilot.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/operator/autopilot.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/operator/autopilot.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/operator/autopilot.html#command-options)Command Options

* [-cleanup-dead-servers](https://www.consul.io/docs/commands/operator/autopilot.html" \l "cleanup-dead-servers) - Specifies whether to enable automatic removal of dead servers upon the successful joining of new servers to the cluster. Must be one of [true|false].

* [-last-contact-threshold](https://www.consul.io/docs/commands/operator/autopilot.html" \l "last-contact-threshold) - Controls the maximum amount of time a server can go without contact from the leader before being considered unhealthy. Must be a duration value such as 200ms.

* [-max-trailing-logs](https://www.consul.io/docs/commands/operator/autopilot.html" \l "max-trailing-logs) - Controls the maximum number of log entries that a server can trail the leader by before being considered unhealthy.

* [-server-stabilization-time](https://www.consul.io/docs/commands/operator/autopilot.html" \l "server-stabilization-time) - Controls the minimum amount of time a server must be stable in the 'healthy' state before being added to the cluster. Only takes effect if all servers are running Raft protocol version 3 or higher. Must be a duration value such as 10s.

* [-disable-upgrade-migration](https://www.consul.io/docs/commands/operator/autopilot.html" \l "disable-upgrade-migration) - (Enterprise-only) Controls whether Consul will avoid promoting new servers until it can perform a migration. Must be one of [true|false].

* [-redundancy-zone-tag](https://www.consul.io/docs/commands/operator/autopilot.html" \l "redundancy-zone-tag)- (Enterprise-only) Controls the [-node-meta](https://www.consul.io/docs/agent/options.html#_node_meta) key name used for separating servers into different redundancy zones.

* [-upgrade-version-tag](https://www.consul.io/docs/commands/operator/autopilot.html" \l "upgrade-version-tag) - (Enterprise-only) Controls the [-node-meta](https://www.consul.io/docs/agent/options.html#_node_meta) tag to use for version info when performing upgrade migrations. If left blank, the Consul version will be used.

The output looks like this:

Configuration updated!

The return code will indicate success or failure.

# Consul Operator Raft

Command: consul operator raft

The Raft operator command is used to interact with Consul's Raft subsystem. The command can be used to verify Raft peers or in rare cases to recover quorum by removing invalid peers.

Usage: consul operator raft <subcommand> [options]

The Raft operator command is used to interact with Consul's Raft subsystem. The

command can be used to verify Raft peers or in rare cases to recover quorum by

removing invalid peers.

Subcommands:

list-peers Display the current Raft peer configuration

remove-peer Remove a Consul server from the Raft configuration

## [»](https://www.consul.io/docs/commands/operator/raft.html" \l "list-peers)list-peers

This command displays the current Raft peer configuration.

Usage: consul operator raft list-peers -stale=[true|false]

* [-stale](https://www.consul.io/docs/commands/operator/raft.html#stale) - Optional and defaults to "false" which means the leader provides the result. If the cluster is in an outage state without a leader, you may need to set this to "true" to get the configuration from a non-leader server.

The output looks like this:

Node ID Address State Voter RaftProtocol

alice 127.0.0.1:8300 127.0.0.1:8300 follower true 2

bob 127.0.0.2:8300 127.0.0.2:8300 leader true 3

carol 127.0.0.3:8300 127.0.0.3:8300 follower true 2

Node is the node name of the server, as known to Consul, or "(unknown)" if the node is stale and not known.

ID is the ID of the server. This is the same as the Address in Consul 0.7 but may be upgraded to a GUID in a future version of Consul.

Address is the IP:port for the server.

State is either "follower" or "leader" depending on the server's role in the Raft configuration.

Voter is "true" or "false", indicating if the server has a vote in the Raft configuration. Future versions of Consul may add support for non-voting servers.

## [»](https://www.consul.io/docs/commands/operator/raft.html" \l "remove-peer)remove-peer

This command removes the Consul server with given address from the Raft configuration.

There are rare cases where a peer may be left behind in the Raft configuration even though the server is no longer present and known to the cluster. This command can be used to remove the failed server so that it is no longer affects the Raft quorum. If the server still shows in the output of the [consul members](https://www.consul.io/docs/commands/members.html) command, it is preferable to clean up by simply running [consul force-leave](https://www.consul.io/docs/commands/force-leave.html) instead of this command.

Usage: consul operator raft remove-peer -address="IP:port"

* [-address](https://www.consul.io/docs/commands/operator/raft.html" \l "address) - "IP:port" for the server to remove. The port number is usually 8300, unless configured otherwise.

* [-id](https://www.consul.io/docs/commands/operator/raft.html" \l "id) - ID of the server to remove.

The return code will indicate success or failure.

# Consul Reload

Command: consul reload

The reload command triggers a reload of configuration files for the agent.

The SIGHUP signal is usually used to trigger a reload of configurations, but in some cases it may be more convenient to trigger the CLI instead.

This command operates the same as the signal, meaning that it will trigger a reload, but does not wait for the reload to complete. Any errors with the reload will be present in the agent logs and not in the output of this command.

**NOTE**

Not all configuration options are reloadable. See the [Reloadable Configuration](https://www.consul.io/docs/agent/options.html#reloadable-configuration) section on the agent options page for details on which options are supported.

## [»](https://www.consul.io/docs/commands/reload.html#usage)Usage

Usage: consul reload

#### [»](https://www.consul.io/docs/commands/reload.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/reload.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/reload.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/reload.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/reload.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/reload.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/reload.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/reload.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

# Consul RTT

Command: consul rtt

The rtt command estimates the network round trip time between two nodes using Consul's network coordinate model of the cluster.

See the [Network Coordinates](https://www.consul.io/docs/internals/coordinates.html) internals guide for more information on how these coordinates are computed.

## [»](https://www.consul.io/docs/commands/rtt.html#usage)Usage

Usage: consul rtt [options] node1 [node2]

At least one node name is required. If the second node name isn't given, it is set to the agent's node name. These are the node names as known to Consul as the consul members command would show, not IP addresses.

#### [»](https://www.consul.io/docs/commands/rtt.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/rtt.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/rtt.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/rtt.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/rtt.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/rtt.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/rtt.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/rtt.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/rtt.html#command-options)Command Options

* [-wan](https://www.consul.io/docs/commands/rtt.html#wan) - Instructs the command to use WAN coordinates instead of LAN coordinates. By default, the two nodes are assumed to be nodes in the local datacenter and the LAN coordinates are used. If the -wan option is given, then the WAN coordinates are used, and the node names must be suffixed by a period and the datacenter (eg. "myserver.dc1"). It is not possible to measure between LAN coordinates and WAN coordinates, so both nodes must be in the same area.

The following environment variables control accessing the HTTP server via SSL:

* [CONSUL\_HTTP\_SSL](https://www.consul.io/docs/commands/rtt.html#consul_http_ssl) Set this to enable SSL
* [CONSUL\_HTTP\_SSL\_VERIFY](https://www.consul.io/docs/commands/rtt.html#consul_http_ssl_verify) Set this to disable certificate checking (not recommended)

## [»](https://www.consul.io/docs/commands/rtt.html#output)Output

If coordinates are available, the command will print the estimated round trip time between the given nodes:

$ consul rtt n1 n2

Estimated n1 <-> n2 rtt: 0.610 ms (using LAN coordinates)

$ consul rtt n2 # Running from n1

Estimated n1 <-> n2 rtt: 0.610 ms (using LAN coordinates)

$ consul rtt -wan n1.dc1 n2.dc2

Estimated n1.dc1 <-> n2.dc2 rtt: 1.275 ms (using WAN coordinates)

# Consul Agent Services

Command: consul services

The services command has subcommands for interacting with Consul services registered with the [local agent](https://www.consul.io/docs/agent/basics.html). These provide useful commands such as register and deregister for easily registering services in scripts, dev mode, etc. To view all services in the catalog, instead of only agent-local services, see the [catalog services](https://www.consul.io/docs/commands/catalog/services.html) command.

## [»](https://www.consul.io/docs/commands/services.html#usage)Usage

Usage: consul services <subcommand>

For the exact documentation for your Consul version, run consul services -h to view the complete list of subcommands.

Usage: consul services <subcommand> [options] [args]

...

Subcommands:

deregister Deregister services with the local agent

register Register services with the local agent

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar.

## [»](https://www.consul.io/docs/commands/services.html#basic-examples)Basic Examples

To create a simple service:

$ consul services register -name=web

To create a service from a configuration file:

$ cat web.json

{

"Service": {

"Name": "web"

}

}

$ consul services register web.json

To deregister a service:

*# Either style works:*

$ consul services deregister web.json

$ consul services deregister -id web

# Consul Agent Service Registration

Command: consul services register

The services register command registers a service with the local agent. This command returns after registration and must be paired with explicit service deregistration. This command simplifies service registration from scripts, in dev mode, etc.

This is just one method of service registration. Services can also be registered by placing a [service definition](https://www.consul.io/docs/agent/services.html) in the Consul agent configuration directory and issuing a [reload](https://www.consul.io/docs/commands/reload.html). This approach is easiest for configuration management systems that other systems that have access to the configuration directory. Clients may also use the [HTTP API](https://www.consul.io/api/agent/service.html) directly.

## [»](https://www.consul.io/docs/commands/services/register.html#usage)Usage

Usage: consul services register [options] [FILE...]

This command can register either a single service using flags documented below, or one or more services using service definition files in HCL or JSON format. The service is registered against the specified Consul agent (defaults to the local agent). This agent will execute all registered health checks.

This command returns after registration succeeds. It must be paired with a deregistration command or API call to remove the service. To ensure that services are properly deregistered, it is **highly recommended** that a check is created with the[DeregisterCriticalServiceAfter](https://www.consul.io/api/agent/check.html#deregistercriticalserviceafter) configuration set. This will ensure that even if deregistration failed for any reason, the agent will automatically deregister the service instance after it is unhealthy for the specified period of time.

Registered services are persisted in the agent state directory. If the state directory remains unmodified, registered services will persist across restarts.

**Warning for Consul operators:** The Consul agent persists registered services in the local state directory. If this state directory is deleted or lost, services registered with this command will need to be reregistered.

#### [»](https://www.consul.io/docs/commands/services/register.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/services/register.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/services/register.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/services/register.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/services/register.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/services/register.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/services/register.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/services/register.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/services/register.html" \l "service-registration-flags)Service Registration Flags

The flags below should only be set if no arguments are given. If no arguments are given, the flags below can be used to register a single service.

Note that the behavior of each of the fields below is exactly the same as when constructing a standard [service definition](https://www.consul.io/docs/agent/services.html). Please refer to that documentation for full details.

* [-id](https://www.consul.io/docs/commands/services/register.html#id) - The ID of the service. This will default to -name if not set.
* [-name](https://www.consul.io/docs/commands/services/register.html#name) - The name of the service to register.
* [-address](https://www.consul.io/docs/commands/services/register.html#address) - The address of the service. If this isn't specified, it will default to the address registered with the local agent.

* [-port](https://www.consul.io/docs/commands/services/register.html" \l "port) - The port of the service.
* [-meta key=value](https://www.consul.io/docs/commands/services/register.html#meta-key-value) - Specify arbitrary KV metadata to associate with the service instance. This can be specified multiple times.

* [-tag value](https://www.consul.io/docs/commands/services/register.html" \l "tag-value) - Associate a tag with the service instance. This flag can be specified multiples times.

## [»](https://www.consul.io/docs/commands/services/register.html#examples)Examples

To create a simple service:

$ consul services register -name=web

To create a service from a configuration file:

$ cat web.json

{

"Service": {

"Name": "web"

}

}

$ consul services register web.json

# Consul Agent Service Deregistration

Command: consul services deregister

The services deregister command deregisters a service with the local agent. Note that this command can only deregister services that were registered with the agent specified (defaults to the local agent) and is meant to be paired with services register.

This is just one method for service deregistration. If the service was registered with a configuration file, then deleting that file and [reloading](https://www.consul.io/docs/commands/reload.html) Consul is the correct method to deregister. See [Service Definition](https://www.consul.io/docs/agent/services.html) for more information about registering services generally.

## [»](https://www.consul.io/docs/commands/services/deregister.html#usage)Usage

Usage: consul services deregister [options] [FILE...]

This command can deregister either a single service using the -id flag documented below, or one or more services using service definition files in HCL or JSON format. This flexibility makes it easy to pair the command with the services register command since the argument syntax is the same.

#### [»](https://www.consul.io/docs/commands/services/deregister.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/services/deregister.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/services/deregister.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/services/deregister.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/services/deregister.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/services/deregister.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/services/deregister.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/services/deregister.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/services/deregister.html" \l "service-deregistration-flags)Service Deregistration Flags

The flags below should only be set if no arguments are given. If no arguments are given, the flags below can be used to deregister a single service.

* [-id](https://www.consul.io/docs/commands/services/deregister.html#id) - The ID of the service.

## [»](https://www.consul.io/docs/commands/services/deregister.html#examples)Examples

To deregister by ID:

$ consul services deregister -id=web

To deregister from a configuration file:

$ cat web.json

{

"Service": {

"Name": "web"

}

}

$ consul services deregister web.json

# Consul Snapshot

Command: consul snapshot

The snapshot command has subcommands for saving, restoring, and inspecting the state of the Consul servers for disaster recovery. These are atomic, point-in-time snapshots which include key/value entries, service catalog, prepared queries, sessions, and ACLs. This command is available in Consul 0.7.1 and later.

Snapshots are also accessible via the [HTTP API](https://www.consul.io/api/snapshot.html).

## [»](https://www.consul.io/docs/commands/snapshot.html#usage)Usage

Usage: consul snapshot <subcommand>

For the exact documentation for your Consul version, run consul snapshot -h to view the complete list of subcommands.

Usage: consul snapshot <subcommand> [options] [args]

# ...

Subcommands:

agent Periodically saves snapshots of Consul server state

inspect Displays information about a Consul snapshot file

restore Restores snapshot of Consul server state

save Saves snapshot of Consul server state

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar or one of the links below:

* [agent](https://www.consul.io/docs/commands/snapshot/agent.html) (Consul Enterprise only)
* [inspect](https://www.consul.io/docs/commands/snapshot/inspect.html)
* [restore](https://www.consul.io/docs/commands/snapshot/restore.html)
* [save](https://www.consul.io/docs/commands/snapshot/save.html)

## [»](https://www.consul.io/docs/commands/snapshot.html#basic-examples)Basic Examples

To create a snapshot and save it as a file called "backup.snap":

$ consul snapshot save backup.snap

Saved and verified snapshot to index 8419

To restore a snapshot from a file called "backup.snap":

$ consul snapshot restore backup.snap

Restored snapshot

To inspect a snapshot from the file "backup.snap":

$ consul snapshot inspect backup.snap

ID 2-5-1477944140022

Size 667

Index 5

Term 2

Version 1

To run a daemon process that periodically saves snapshots (Consul Enterprise only):

$ consul snapshot agent

For more examples, ask for subcommand help or view the subcommand documentation by clicking on one of the links in the sidebar.

# Consul Snapshot Agent

Command: consul snapshot agent

The [agent](https://www.consul.io/docs/commands/snapshot/agent.html) subcommand described here is available only in [Consul Enterprise](https://www.hashicorp.com/products/consul/) version 0.7.1 and later. All other [snapshot subcommands](https://www.consul.io/docs/commands/snapshot.html) are available in the open source version of Consul.

The snapshot agent subcommand starts a process that takes snapshots of the state of the Consul servers and saves them locally, or pushes them to an optional remote storage service. This subcommand is only available in Consul Enterprise 0.7.1 and later.

The agent can be run as a long-running daemon process or in a one-shot mode from a batch job, based on the [-interval](https://www.consul.io/docs/commands/snapshot/agent.html#interval)argument. Snapshotting a remote datacenter is only available in one-shot mode.

As a long-running daemon, the agent will perform a leader election so multiple processes can be run in a highly available fashion with automatic failover. The agent will also register itself with Consul as a service, along with health checks that show the agent is alive ("Consul Snapshot Agent Alive") and able to take snapshots ("Consul Snapshot Agent Saving Snapshots"). The latter check is only added on agents who have become a leader, so it's possible for operators to tell which instances are alive and on standby and which instance has become leader and starting saving snapshots.

As snapshots are saved, they will be reported in the log produced by the agent:

2016/11/16 21:21:13 [INFO] Snapshot agent running

2016/11/16 21:21:13 [INFO] Waiting to obtain leadership...

2016/11/16 21:21:13 [INFO] Obtained leadership

2016/11/16 21:21:13 [INFO] Saved snapshot 1479360073448728784

The number shown with the saved snapshot is its ID, which is based on a UNIX timestamp with nanosecond resolution, so collisions are unlikely and IDs are monotonically increasing with time. This makes it easy to locate the latest snapshot, even if the log data isn't available. The snapshot ID always appears in the file name when using local storage, or in the object key when using remote storage.

Snapshots can be restored using the [consul snapshot restore](https://www.consul.io/docs/commands/snapshot/restore.html) command, or the [HTTP API](https://www.consul.io/api/snapshot.html).

If ACLs are enabled, a management token must be supplied in order to perform snapshot operations.

## [»](https://www.consul.io/docs/commands/snapshot/agent.html#usage)Usage

Usage: consul snapshot agent [options]

#### [»](https://www.consul.io/docs/commands/snapshot/agent.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/snapshot/agent.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/snapshot/agent.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/snapshot/agent.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/snapshot/agent.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/snapshot/agent.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/snapshot/agent.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/snapshot/agent.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.

#### [»](https://www.consul.io/docs/commands/snapshot/agent.html" \l "config-file-options-)Config File Options:

* [-config-dir](https://www.consul.io/docs/commands/snapshot/agent.html" \l "config-dir) - Directory to look for JSON config files. Files will be read in alphabetical order and must end with the extension ".json". This won't recursively descend directories. This can be specified multiple times on the command line.
* [-config-file](https://www.consul.io/docs/commands/snapshot/agent.html#config-file) - File to read JSON configuration from. Files must end with the extension ".json". This can be specified multiple times on the command line.

Config files referenced using -config-dir and -config-file have the following format (shown populated with default values):

{

"snapshot\_agent": {

"http\_addr": "127.0.0.1:8500",

"token": "",

"datacenter": "",

"ca\_file": "",

"ca\_path": "",

"cert\_file": "",

"key\_file": "",

"tls\_server\_name": "",

"log": {

"level": "INFO",

"enable\_syslog": **false**,

"syslog\_facility": "LOCAL0"

},

"snapshot": {

"interval": "1h",

"retain": 30,

"stale": **false**,

"service": "consul-snapshot",

"deregister\_after": "72h",

"lock\_key": "consul-snapshot/lock",

"max\_failures": 3

},

"local\_storage": {

"path": "."

},

"aws\_storage": {

"access\_key\_id": "",

"secret\_access\_key": "",

"s3\_region": "",

"s3\_bucket": "",

"s3\_key\_prefix": "consul-snapshot",

"s3\_server\_side\_encryption":**false**,

"s3\_static\_snapshot\_name":""

}

}

}

All fields are optional, and config files without a snapshot\_agent object will be ignored. At least one config file needs to have a snapshot\_agent object, or the snapshot agent will fail to start. The Consul agent is set up to ignore anysnapshot\_agent object, so it's safe to use common config directories for both agents if desired.

#### [»](https://www.consul.io/docs/commands/snapshot/agent.html" \l "snapshot-options)Snapshot Options

* -interval - Interval at which to perform snapshots as a time with a unit suffix, which can be "s", "m", "h" for seconds, minutes, or hours. If 0 is provided, the agent will take a single snapshot and then exit, which is useful for running snapshots via batch jobs. Defaults to "1h"

* [-lock-key](https://www.consul.io/docs/commands/snapshot/agent.html" \l "lock-key) - A prefix in Consul's KV store used to coordinate between different instances of the snapshot agent order to only have one active instance at a time. For highly available operation of the snapshot agent, simply run multiple instances. All instances must be configured with the same lock key in order to properly coordinate. Defaults to "consul-snapshot/lock".

* [-max-failures](https://www.consul.io/docs/commands/snapshot/agent.html" \l "max-failures) - Number of snapshot failures after which the snapshot agent will give up leadership. In a highly available operation with multiple snapshot agents available, this gives another agent a chance to take over if an agent is experiencing issues, such as running out of disk space for snapshots. Defaults to 3.

* [-retain](https://www.consul.io/docs/commands/snapshot/agent.html" \l "retain) - Number of snapshots to retain. After each snapshot is taken, the oldest snapshots will start to be deleted in order to retain at most this many snapshots. If this is set to 0, the agent will not perform this and snapshots will accumulate forever. Defaults to 30.

#### [»](https://www.consul.io/docs/commands/snapshot/agent.html" \l "agent-options)Agent Options

* [-deregister-after](https://www.consul.io/docs/commands/snapshot/agent.html" \l "deregister-after) - An interval, after which if the agent is unhealthy it will be automatically deregistered from Consul service discovery. This is a time with a unit suffix, which can be "s", "m", "h" for seconds, minutes, or hours. If 0 is provided, this will be disabled. Defaults to "72h".
* [-log-level](https://www.consul.io/docs/commands/snapshot/agent.html#log-level) - Controls verbosity of snapshot agent logs. Valid options are "TRACE", "DEBUG", "INFO", "WARN", "ERR". Defaults to "INFO".
* [-service](https://www.consul.io/docs/commands/snapshot/agent.html#service) - The service name to used when registering the agent with Consul. Registering helps monitor running agents and the leader registers an additional health check to monitor that snapshots are taking place. Defaults to "consul-snapshot".

* [-syslog](https://www.consul.io/docs/commands/snapshot/agent.html" \l "syslog) - This enables forwarding logs to syslog. Defaults to false.

* [-syslog-facility](https://www.consul.io/docs/commands/snapshot/agent.html" \l "syslog-facility) - Sets the facility to use for forwarding logs to syslog. Defaults to "LOCAL0".

#### [»](https://www.consul.io/docs/commands/snapshot/agent.html" \l "local-storage-options)Local Storage Options

* [-local-path](https://www.consul.io/docs/commands/snapshot/agent.html" \l "local-path) - Location to store snapshots locally. The default behavior of the snapshot agent is to store snapshots locally in this directory. Defaults to "." to use the current working directory. If an alternate storage option is configured, then local storage will be disabled and this option will be ignored.

#### [»](https://www.consul.io/docs/commands/snapshot/agent.html" \l "s3-storage-options)S3 Storage Options

Note that despite the AWS references, any S3-compatible endpoint can be specified with -aws-s3-endpoint.

* [-aws-access-key-id](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-access-key-id) and -aws-secret-access-key - These arguments supply authentication information for connecting to S3. These may also be supplied using the following alternative methods:

* + [AWS\_ACCESS\_KEY\_ID](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws_access_key_id) and AWS\_SECRET\_ACCESS\_KEY environment variables
  + A credentials file (~/.aws/credentials or the file at the path specified by the AWS\_SHARED\_CREDENTIALS\_FILEenvironment variable)
  + ECS task role metadata (container-specific)
  + EC2 instance role metadata

* [-aws-s3-bucket](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-s3-bucket) - S3 bucket to use. Required for S3 storage, and setting this disables local storage. This should be only the bucket name without any part of the key prefix.

* [-aws-s3-key-prefix](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-s3-key-prefix) - Prefix to use for snapshot files in S3. Defaults to "consul-snapshot".

* [-aws-s3-region](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-s3-region) - S3 region to use. Required for S3 storage.

* [-aws-s3-endpoint](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-s3-endpoint) - Optional S3 endpoint to use. Can also be specified using the AWS\_S3\_ENDPOINT environment variable.

* [-aws-s3-server-side-encryption](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-s3-server-side-encryption) - Enables saving snapshots to S3 using server side encryption with [Amazon S3-Managed Encryption Keys](http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingServerSideEncryption.html)

* [-aws-s3-static-snapshot-name](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-s3-static-snapshot-name) - If this is given, all snapshots are saved with the same file name. The agent will not rotate or versionize snapshots, and will save them with the same name each time. Use this if you want to rely on [S3's versioning capabilities](http://docs.aws.amazon.com/AmazonS3/latest/dev/Versioning.html) instead of the agent handling it for you.

* [-aws-s3-enable-kms](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-s3-enable-kms) - Enables using [Amazon KMS](https://aws.amazon.com/kms/) for encrypting snapshots.

* [-aws-s3-kms-key](https://www.consul.io/docs/commands/snapshot/agent.html" \l "aws-s3-kms-key) - Optional Amazon KMS key to use, if this is not set the default KMS master key will be used. Set this if you want to manage key rotation yourself.

#### [»](https://www.consul.io/docs/commands/snapshot/agent.html" \l "s3-required-permissions)S3 Required Permissions

Different S3 permissions are required depending on the configuration of the snapshot agent. In particular extra permissions are required when snapshot rotation is enabled. S3 storage snapshot rotation is enabled when the retainconfiguration is greater than 0 and when there is no aws-s3-static-snapshot-name configured.

| **Permission** | **Resource** | **When you need it** |
| --- | --- | --- |
| PutObject | arn:aws:s3:::<bucket name>/<key> | Required for all operations. |
| DeleteObject | arn:aws:s3:::<bucket name>/<key> | Required only when snapshot rotation is enabled |
| ListBucket | arn:aws:s3:::<bucket name> | Required only when snapshot rotation is enabled |
| ListBucketVersions | arn:aws:s3:::<bucket name> | Required only when snapshot rotation is enabled |

Within the table <key> refers to the the key used to store the snapshot. When aws-s3-static-snapshot-name is configured the <key> is simply the value of that configuration. Otherwise the <key> will be the <aws-s3-key-prefix configuration>/consul-\*.snap.

The following example IAM policy document assumes that the aws-s3-bucket is consul-data with defaults for aws-s3-key-prefix, aws-s3-static-snapshot-name and retain:

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "",

"Effect": "Allow",

"Action": [

"s3:PutObject",

"s3:DeleteObject"

],

"Resource": "arn:aws:s3:::consul-data/consul-snapshots/consul-\*.snap"

},

{

"Sid": "",

"Effect": "Allow",

"Action": [

"s3:ListBucketVersions",

"s3:ListBucket"

],

"Resource": "arn:aws:s3:::consul-data"

}

]

}

## [»](https://www.consul.io/docs/commands/snapshot/agent.html#examples)Examples

Running the agent with no arguments will run a long-running daemon process that will perform leader election for highly available operation, register itself with Consul service discovery with health checks, take snapshots every hour, retain the last 30 snapshots, and save snapshots into the current working directory:

$ consul snapshot agent

To run a one-shot backup, set the backup interval to 0. This will run a single snapshot and delete any old snapshots based on the retain settings, but it will not perform any leader election or service registration:

$ consul snapshot agent -interval=0

Please see the [HTTP API](https://www.consul.io/api/snapshot.html) documentation for more details about snapshot internals.

# Consul Snapshot Inspect

Command: consul snapshot inspect

The snapshot inspect command is used to inspect an atomic, point-in-time snapshot of the state of the Consul servers which includes key/value entries, service catalog, prepared queries, sessions, and ACLs. The snapshot is read from the given file.

The following fields are displayed when inspecting a snapshot:

* [ID](https://www.consul.io/docs/commands/snapshot/inspect.html#id) - A unique ID for the snapshot, only used for differentiation purposes.

* [Size](https://www.consul.io/docs/commands/snapshot/inspect.html" \l "size) - The size of the snapshot, in bytes.

* [Index](https://www.consul.io/docs/commands/snapshot/inspect.html" \l "index) - The Raft index of the latest log entry in the snapshot.

* [Term](https://www.consul.io/docs/commands/snapshot/inspect.html" \l "term) - The Raft term of the latest log entry in the snapshot.

* [Version](https://www.consul.io/docs/commands/snapshot/inspect.html" \l "version) - The snapshot format version. This only refers to the structure of the snapshot, not the data contained within.

## [»](https://www.consul.io/docs/commands/snapshot/inspect.html#usage)Usage

Usage: consul snapshot inspect [options] FILE

## [»](https://www.consul.io/docs/commands/snapshot/inspect.html#examples)Examples

To inspect a snapshot from the file "backup.snap":

$ consul snapshot inspect backup.snap

ID 2-5-1477944140022

Size 667

Index 5

Term 2

Version 1

Please see the [HTTP API](https://www.consul.io/api/snapshot.html) documentation for more details about snapshot internals.

# Consul Snapshot Restore

Command: consul snapshot restore

The snapshot restore command is used to restore an atomic, point-in-time snapshot of the state of the Consul servers which includes key/value entries, service catalog, prepared queries, sessions, and ACLs. The snapshot is read from the given file.

Restores involve a potentially dangerous low-level Raft operation that is not designed to handle server failures during a restore. This command is primarily intended to be used when recovering from a disaster, restoring into a fresh cluster of Consul servers.

If ACLs are enabled, a management token must be supplied in order to perform a snapshot restore.

## [»](https://www.consul.io/docs/commands/snapshot/restore.html#usage)Usage

Usage: consul snapshot restore [options] FILE

#### [»](https://www.consul.io/docs/commands/snapshot/restore.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/snapshot/restore.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/snapshot/restore.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/snapshot/restore.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/snapshot/restore.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/snapshot/restore.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/snapshot/restore.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/snapshot/restore.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/snapshot/restore.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/snapshot/restore.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

## [»](https://www.consul.io/docs/commands/snapshot/restore.html#examples)Examples

To restore a snapshot from the file "backup.snap":

$ consul snapshot restore backup.snap

Restored snapshot

Please see the [HTTP API](https://www.consul.io/api/snapshot.html) documentation for more details about snapshot internals.

# Consul Snapshot Save

Command: consul snapshot save

The snapshot save command is used to retrieve an atomic, point-in-time snapshot of the state of the Consul servers which includes key/value entries, service catalog, prepared queries, sessions, and ACLs. The snapshot is saved to the given file.

If ACLs are enabled, a management token must be supplied in order to perform a snapshot save.

## [»](https://www.consul.io/docs/commands/snapshot/save.html#usage)Usage

Usage: consul snapshot save [options] FILE

#### [»](https://www.consul.io/docs/commands/snapshot/save.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/snapshot/save.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/snapshot/save.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/snapshot/save.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/snapshot/save.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/snapshot/save.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/snapshot/save.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/snapshot/save.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/snapshot/save.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/snapshot/save.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

## [»](https://www.consul.io/docs/commands/snapshot/save.html#examples)Examples

To create a snapshot from the leader server and save it to "backup.snap":

$ consul snapshot save backup.snap

Saved and verified snapshot to index 8419

By default, snapshots are taken using a consistent mode that forwards requests to the leader and the leader verifies it is still in power before taking the snapshot.

After the snapshot is written to the given file it is read back and verified for integrity.

To create a potentially stale snapshot from any available server, use the stale consistency mode:

$ consul snapshot save -stale backup.snap

# ...

This is useful for situations where a cluster is in a degraded state and no leader is available. To target a specific server for a snapshot, you can run the consul snapshot save command on that specific server.

Please see the [HTTP API](https://www.consul.io/api/snapshot.html) documentation for more details about snapshot internals.

# Consul TLS

Command: consul tls

The tls command is used to help with setting up a CA and certificates for Consul TLS.

## [»](https://www.consul.io/docs/commands/tls.html#basic-examples)Basic Examples

Create a CA:

$ consul tls ca create

==> Saved consul-agent-ca.pem

==> Saved consul-agent-ca-key.pem

Create a client certificate:

$ consul tls cert create -client

==> Using consul-agent-ca.pem and consul-agent-ca-key.pem

==> Saved consul-client-dc1-0.pem

==> Saved consul-client-dc1-0-key.pem

For more examples, ask for subcommand help or view the subcommand documentation by clicking on one of the links in the sidebar.

## [»](https://www.consul.io/docs/commands/tls.html#usage)Usage

Usage: consul tls <subcommand> <subcommand> [options]

For the exact documentation for your Consul version, run consul tls -h to view the complete list of subcommands.

Usage: consul tls <subcommand> <subcommand> [options]

# ...

Subcommands:

ca Helpers for CAs

cert Helpers for certificates

For more information, examples, and usage about a subcommand, click on the name of the subcommand in the sidebar or one of the links below:

# Consul TLS CA Create

Command: consul tls ca create

This command create a self signed CA to be used for Consul TLS setup.

## [»](https://www.consul.io/docs/commands/tls/ca.html" \l "example)Example

Create CA:

$ consul tls ca create

**==>** Saved consul-ca.pem

**==>** Saved consul-ca-key.pem

## [»](https://www.consul.io/docs/commands/tls/ca.html#usage)Usage

Usage: consul tls ca create [filename-prefix] [options]

#### [»](https://www.consul.io/docs/commands/tls/ca.html" \l "tls-ca-create-options)TLS CA Create Options

* [-days=<int>](https://www.consul.io/docs/commands/tls/ca.html" \l "days-lt-int-gt-) - Provide number of days the CA is valid for from now on, defaults to 5 years.

# Consul TLS Cert Create

Command: consul tls cert create

The tls cert create command is used to create certificates for your Consul TLS setup.

## [»](https://www.consul.io/docs/commands/tls/cert.html#examples)Examples

Create a certificate for servers:

$ consul tls cert create -server

**==>** WARNING: Server Certificates grants authority to become a

server and access all state **in** the cluster including root keys

and all ACL tokens. Do not distribute them to production hosts

that are not server nodes. Store them as securely as CA keys.

**==>** Using consul-ca.pem and consul-ca-key.pem

**==>** Saved consul-server-dc1-0.pem

**==>** Saved consul-server-dc1-0-key.pem

Create a certificate for clients:

$ consul tls cert create -client

**==>** Using consul-ca.pem and consul-ca-key.pem

**==>** Saved consul-client-0.pem

**==>** Saved consul-client-0-key.pem

Create a certificate for cli:

$ consul tls cert create -cli

**==>** Using consul-ca.pem and consul-ca-key.pem

**==>** Saved consul-cli-0.pem

**==>** Saved consul-cli-0-key.pem

## [»](https://www.consul.io/docs/commands/tls/cert.html#usage)Usage

Usage: consul tls cert create [filename-prefix] [options]

#### [»](https://www.consul.io/docs/commands/tls/cert.html" \l "tls-cert-create-options)TLS Cert Create Options

* [-additional-dnsname=<string>](https://www.consul.io/docs/commands/tls/cert.html" \l "additional-dnsname-lt-string-gt-) - Provide additional dnsname for Subject Alternative Names.

* [-ca=<string>](https://www.consul.io/docs/commands/tls/cert.html" \l "ca-lt-string-gt-) - Provide path to the ca

* [-cli](https://www.consul.io/docs/commands/tls/cert.html" \l "cli) - Generate cli certificate

* [-client](https://www.consul.io/docs/commands/tls/cert.html" \l "client) - Generate client certificate
* [-days=<int>](https://www.consul.io/docs/commands/tls/cert.html#days-lt-int-gt-) - Provide number of days the certificate is valid for from now on.

* [-dc=<string>](https://www.consul.io/docs/commands/tls/cert.html" \l "dc-lt-string-gt-) - Provide the datacenter. Matters only for -server certificates

* [-domain=<string>](https://www.consul.io/docs/commands/tls/cert.html" \l "domain-lt-string-gt-) - Provide the domain. Matters only for -server certificates

* [-key=<string>](https://www.consul.io/docs/commands/tls/cert.html" \l "key-lt-string-gt-) - Provide path to the key

* [-server](https://www.consul.io/docs/commands/tls/cert.html" \l "server) - Generate server certificate

# Consul Validate

The consul validate command performs a thorough sanity test on Consul configuration files. For each file or directory given, the command will attempt to parse the contents just as the consul agent command would, and catch any errors.

This is useful to do a test of the configuration only, without actually starting the agent. This performs all of the validation the agent would, so this should be given the complete set of configuration files that are going to be loaded by the agent. This command cannot operate on partial configuration fragments since those won't pass the full agent validation.

For more information on the format of Consul's configuration files, read the consul agent [Configuration Files](https://www.consul.io/docs/agent/options.html#configuration-files) section.

## [»](https://www.consul.io/docs/commands/validate.html#usage)Usage

Usage: consul validate [options] FILE\_OR\_DIRECTORY...

Returns 0 if the configuration is valid, or 1 if there are problems.

$ consul validate /etc/consul.d

Configuration is valid!

Consul Version

Command: consul version

The version command prints the version of Consul and the protocol versions it understands for speaking to other agents.

$ consul version

Consul v0.7.4

Protocol 2 spoken by default, understands 2 to 3 (agent will automatically use protocol >2 when speaking to compatible agents)

# Consul Watch

Command: consul watch

The watch command provides a mechanism to watch for changes in a particular data view (list of nodes, service members, key value, etc) and to invoke a process with the latest values of the view. If no process is specified, the current values are dumped to STDOUT which can be a useful way to inspect data in Consul.

There is more [documentation on watches here](https://www.consul.io/docs/agent/watches.html).

## [»](https://www.consul.io/docs/commands/watch.html#usage)Usage

Usage: consul watch [options] [child...]

The only required option is -type which specifies the particular data view. Depending on the type, various options may be required or optionally provided. There is more documentation on watch [specifications here](https://www.consul.io/docs/agent/watches.html).

#### [»](https://www.consul.io/docs/commands/watch.html#api-options)API Options

* [-ca-file=<value>](https://www.consul.io/docs/commands/watch.html#ca-file-lt-value-gt-) - Path to a CA file to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CACERT environment variable.
* [-ca-path=<value>](https://www.consul.io/docs/commands/watch.html#ca-path-lt-value-gt-) - Path to a directory of CA certificates to use for TLS when communicating with Consul. This can also be specified via the CONSUL\_CAPATH environment variable.
* [-client-cert=<value>](https://www.consul.io/docs/commands/watch.html#client-cert-lt-value-gt-) - Path to a client cert file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_CERT environment variable.
* [-client-key=<value>](https://www.consul.io/docs/commands/watch.html#client-key-lt-value-gt-) - Path to a client key file to use for TLS when verify\_incoming is enabled. This can also be specified via the CONSUL\_CLIENT\_KEY environment variable.
* [-http-addr=<addr>](https://www.consul.io/docs/commands/watch.html#http-addr-lt-addr-gt-) - Address of the Consul agent with the port. This can be an IP address or DNS address, but it must include the port. This can also be specified via the CONSUL\_HTTP\_ADDR environment variable. In Consul 0.8 and later, the default value is [http://127.0.0.1:8500](http://127.0.0.1:8500/), and https can optionally be used instead. The scheme can also be set to HTTPS by setting the environment variable CONSUL\_HTTP\_SSL=true. This may be a unix domain socket usingunix:///path/to/socket if the [agent is configured to listen](https://www.consul.io/docs/agent/options.html#addresses) that way.
* [-tls-server-name=<value>](https://www.consul.io/docs/commands/watch.html#tls-server-name-lt-value-gt-) - The server name to use as the SNI host when connecting via TLS. This can also be specified via the CONSUL\_TLS\_SERVER\_NAME environment variable.
* [-token=<value>](https://www.consul.io/docs/commands/watch.html#token-lt-value-gt-) - ACL token to use in the request. This can also be specified via the CONSUL\_HTTP\_TOKENenvironment variable. If unspecified, the query will default to the token of the Consul agent at the HTTP address.
* [-datacenter=<name>](https://www.consul.io/docs/commands/watch.html#datacenter-lt-name-gt-) - Name of the datacenter to query. If unspecified, the query will default to the datacenter of the Consul agent at the HTTP address.
* [-stale](https://www.consul.io/docs/commands/watch.html#stale) - Permit any Consul server (non-leader) to respond to this request. This allows for lower latency and higher throughput, but can result in stale data. This option has no effect on non-read operations. The default value is false.

#### [»](https://www.consul.io/docs/commands/watch.html#command-options)Command Options

* [-key](https://www.consul.io/docs/commands/watch.html" \l "key) - Key to watch. Only for key type.
* [-name](https://www.consul.io/docs/commands/watch.html#name)- Event name to watch. Only for event type.

* [-passingonly=[true|false]](https://www.consul.io/docs/commands/watch.html" \l "passingonly-true-false-) - Should only passing entries be returned. Defaults to false and only applies for service type.
* [-prefix](https://www.consul.io/docs/commands/watch.html#prefix) - Key prefix to watch. Only for keyprefix type.
* [-service](https://www.consul.io/docs/commands/watch.html#service) - Service to watch. Required for service type, optional for checks type.
* [-shell](https://www.consul.io/docs/commands/watch.html#shell) - Optional, use a shell to run the command (can set a custom shell via the SHELL environment variable). The default value is true.

* [-state](https://www.consul.io/docs/commands/watch.html" \l "state) - Check state to filter on. Optional for checks type.
* [-tag](https://www.consul.io/docs/commands/watch.html#tag) - Service tag to filter on. Optional for service type.

* [-type](https://www.consul.io/docs/commands/watch.html" \l "type) - Watch type. Required, one of "key, keyprefix, services, nodes, service, checks, or event.

# Consul Agent

The Consul agent is the core process of Consul. The agent maintains membership information, registers services, runs checks, responds to queries, and more. The agent must run on every node that is part of a Consul cluster.

Any agent may run in one of two modes: client or server. A server node takes on the additional responsibility of being part of the [consensus quorum](https://www.consul.io/docs/internals/consensus.html). These nodes take part in Raft and provide strong consistency and availability in the case of failure. The higher burden on the server nodes means that usually they should be run on dedicated instances -- they are more resource intensive than a client node. Client nodes make up the majority of the cluster, and they are very lightweight as they interface with the server nodes for most operations and maintain very little state of their own.

## [»](https://www.consul.io/docs/agent/basics.html" \l "running-an-agent)Running an Agent

The agent is started with the [consul agent](https://www.consul.io/docs/commands/agent.html) command. This command blocks, running forever or until told to quit. The agent command takes a variety of [configuration options](https://www.consul.io/docs/agent/options.html#command-line-options), but most have sane defaults.

When running [consul agent](https://www.consul.io/docs/commands/agent.html), you should see output similar to this:

$ consul agent -data-dir=/tmp/consul

==> Starting Consul agent...

==> Consul agent running!

Node name: 'Armons-MacBook-Air'

Datacenter: 'dc1'

Server: false (bootstrap: false)

Client Addr: 127.0.0.1 (HTTP: 8500, DNS: 8600)

Cluster Addr: 192.168.1.43 (LAN: 8301, WAN: 8302)

==> Log data will now stream in as it occurs:

[INFO] serf: EventMemberJoin: Armons-MacBook-Air.local 192.168.1.43

...

There are several important messages that [consul agent](https://www.consul.io/docs/commands/agent.html) outputs:

* **Node name**: This is a unique name for the agent. By default, this is the hostname of the machine, but you may customize it using the [-node](https://www.consul.io/docs/agent/options.html#_node) flag.
* **Datacenter**: This is the datacenter in which the agent is configured to run. Consul has first-class support for multiple datacenters; however, to work efficiently, each node must be configured to report its datacenter. The [-datacenter](https://www.consul.io/docs/agent/options.html#_datacenter) flag can be used to set the datacenter. For single-DC configurations, the agent will default to "dc1".
* **Server**: This indicates whether the agent is running in server or client mode. Server nodes have the extra burden of participating in the consensus quorum, storing cluster state, and handling queries. Additionally, a server may be in ["bootstrap"](https://www.consul.io/docs/agent/options.html#_bootstrap_expect) mode. Multiple servers cannot be in bootstrap mode as that would put the cluster in an inconsistent state.
* **Client Addr**: This is the address used for client interfaces to the agent. This includes the ports for the HTTP and DNS interfaces. By default, this binds only to localhost. If you change this address or port, you'll have to specify a -http-addr whenever you run commands such as [consul members](https://www.consul.io/docs/commands/members.html) to indicate how to reach the agent. Other applications can also use the HTTP address and port [to control Consul](https://www.consul.io/api/index.html).
* **Cluster Addr**: This is the address and set of ports used for communication between Consul agents in a cluster. Not all Consul agents in a cluster have to use the same port, but this address **MUST** be reachable by all other nodes.

When running under systemd on Linux, Consul notifies systemd by sending READY=1 to the $NOTIFY\_SOCKET when a LAN join has completed. For this either the join or retry\_join option has to be set and the service definition file has to have Type=notify set.

## [»](https://www.consul.io/docs/agent/basics.html" \l "stopping-an-agent)Stopping an Agent

An agent can be stopped in two ways: gracefully or forcefully. To gracefully halt an agent, send the process an interrupt signal (usually Ctrl-C from a terminal or running kill -INT consul\_pid ). When gracefully exiting, the agent first notifies the cluster it intends to leave the cluster. This way, other cluster members notify the cluster that the node has left.

Alternatively, you can force kill the agent by sending it a kill signal. When force killed, the agent ends immediately. The rest of the cluster will eventually (usually within seconds) detect that the node has died and notify the cluster that the node has failed.

It is especially important that a server node be allowed to leave gracefully so that there will be a minimal impact on availability as the server leaves the consensus quorum.

For client agents, the difference between a node failing and a node leaving may not be important for your use case. For example, for a web server and load balancer setup, both result in the same outcome: the web node is removed from the load balancer pool.

## [»](https://www.consul.io/docs/agent/basics.html" \l "lifecycle)Lifecycle

Every agent in the Consul cluster goes through a lifecycle. Understanding this lifecycle is useful for building a mental model of an agent's interactions with a cluster and how the cluster treats a node.

When an agent is first started, it does not know about any other node in the cluster. To discover its peers, it must join the cluster. This is done with the [join](https://www.consul.io/docs/commands/join.html) command or by providing the proper configuration to auto-join on start. Once a node joins, this information is gossiped to the entire cluster, meaning all nodes will eventually be aware of each other. If the agent is a server, existing servers will begin replicating to the new node.

In the case of a network failure, some nodes may be unreachable by other nodes. In this case, unreachable nodes are marked as failed. It is impossible to distinguish between a network failure and an agent crash, so both cases are handled the same. Once a node is marked as failed, this information is updated in the service catalog.

**Note:** There is some nuance here since this update is only possible if the servers can still [form a quorum](https://www.consul.io/docs/internals/consensus.html). Once the network recovers or a crashed agent restarts the cluster will repair itself and unmark a node as failed. The health check in the catalog will also be updated to reflect this.

When a node leaves, it specifies its intent to do so, and the cluster marks that node as having left. Unlike the failed case, all of the services provided by a node are immediately deregistered. If the agent was a server, replication to it will stop.

To prevent an accumulation of dead nodes (nodes in either failed or left states), Consul will automatically remove dead nodes out of the catalog. This process is called reaping. This is currently done on a configurable interval of 72 hours (changing the reap interval is not recommended due to its consequences during outage situations). Reaping is similar to leaving, causing all associated services to be deregistered.

**1.4.0 and later:** This guide only applies in Consul versions 1.4.0 and later. The documentation for the legacy ACL system is [here](https://www.consul.io/docs/guides/acl-legacy.html)

# ACL System

Consul provides an optional Access Control List (ACL) system which can be used to control access to data and APIs. The ACL is [Capability-based](https://en.wikipedia.org/wiki/Capability-based_security), relying on tokens which are associated with policies to determine which fine grained rules can be applied. Consul's capability based ACL system is very similar to the design of [AWS IAM](https://aws.amazon.com/iam/).

## [»](https://www.consul.io/docs/agent/acl-system.html" \l "acl-system-overview)ACL System Overview

The ACL system is designed to be easy to use and fast to enforce while providing administrative insight. At the highest level, there are two major components to the ACL system:

* **ACL Policies** - Policies allow the grouping of a set of rules into a logical unit that can be reused and linked with many tokens.
* **ACL Tokens** - Requests to Consul are authorized by using bearer token. Each ACL token has a public Accessor ID which is used to name a token, and a Secret ID which is used as the bearer token used to make requests to Consul.

ACL tokens and policies are managed by Consul operators via Consul's [ACL API](https://www.consul.io/api/acl/acl.html), [ACL CLI](https://www.consul.io/docs/commands/acl.html), or systems like [HashiCorp's Vault](https://www.vaultproject.io/docs/secrets/consul/index.html).

### [»](https://www.consul.io/docs/agent/acl-system.html" \l "acl-policies)ACL Policies

An ACL policy is a named set of rules and is composed of the following elements:

* **ID** - The policies auto-generated public identifier.
* **Name** - A unique meaningful name for the policy.
* **Rules** - Set of rules granting or denying permissions. See the [Rule Specification](https://www.consul.io/docs/agent/acl-rules.html#rule-specification) documentation for more details.
* **Datacenters** - A list of datacenters the policy is valid within.

#### [»](https://www.consul.io/docs/agent/acl-system.html" \l "builtin-policies)Builtin Policies

* **Global Management** - Grants unrestricted privileges to any token that uses it. When created it will be named global-management and will be assigned the reserved ID of 00000000-0000-0000-0000-000000000001. This policy can be renamed but modification of anything else including the rule set and datacenter scoping will be prevented by Consul.

### [»](https://www.consul.io/docs/agent/acl-system.html" \l "acl-tokens)ACL Tokens

ACL tokens are used to determine if the caller is authorized to perform an action. An ACL token is composed of the following elements:

* **Accessor ID** - The token's public identifier.
* **Secret ID** -The bearer token used when making requests to Consul.
* **Description** - A human readable description of the token. (Optional)
* **Policy Set** - The list of policies that are applicable for the token.
* **Locality** - Indicates whether the token should be local to the datacenter it was created within or created in the primary datacenter and globally replicated.

#### [»](https://www.consul.io/docs/agent/acl-system.html" \l "builtin-tokens)Builtin Tokens

During cluster bootstrapping when ACLs are enabled both the special anonymous and the master token will be injected.

* **Anonymous Token** - The anonymous token is used when a request is made to Consul without specifying a bearer token. The anonymous token's description and policies may be updated but Consul will prevent this tokens deletion. When created, it will be assigned 00000000-0000-0000-0000-000000000002 for its Accessor ID and anonymous for its Secret ID.
* **Master Token** - When a master token is present within the Consul configuration, it is created and will be linked With the builtin Global Management policy giving it unrestricted privileges. The master token is created with the Secret ID set to the value of the configuration entry.

#### [»](https://www.consul.io/docs/agent/acl-system.html" \l "authorization)Authorization

The token Secret ID is passed along with each RPC request to the servers. Consul's [HTTP endpoints](https://www.consul.io/api/index.html) can accept tokens via the token query string parameter, the X-Consul-Token request header, or an [RFC6750](https://tools.ietf.org/html/rfc6750) authorization bearer token. Consul's [CLI commands](https://www.consul.io/docs/commands/index.html) can accept tokens via the token argument, or the CONSUL\_HTTP\_TOKEN environment variable.

If no token is provided for an HTTP request then Consul will use the default ACL token if it has been configured. If no default ACL token was configured then the anonymous token will be used.

#### [»](https://www.consul.io/docs/agent/acl-system.html" \l "acl-rules-and-scope)ACL Rules and Scope

The rules from all policies linked with a token are combined to form that token's effective rule set. Policy rules can be defined in either a whitelist or blacklist mode depending on the configuration of [acl\_default\_policy](https://www.consul.io/docs/agent/options.html#acl_default_policy). If the default policy is to "deny" access to all resources, then policy rules can be set to whitelist access to specific resources. Conversely, if the default policy is “allow” then policy rules can be used to explicitly deny access to resources.

The following table summarizes the ACL resources that are available for constructing rules:

| **Resource** | **Scope** |
| --- | --- |
| [acl](https://www.consul.io/docs/agent/acl-system.html#acl-rules) | Operations for managing the ACL system [ACL API](https://www.consul.io/api/acl/acl.html) |
| [agent](https://www.consul.io/docs/agent/acl-system.html#agent-rules) | Utility operations in the [Agent API](https://www.consul.io/api/agent.html), other than service and check registration |
| [event](https://www.consul.io/docs/agent/acl-system.html#event-rules) | Listing and firing events in the [Event API](https://www.consul.io/api/event.html) |
| [key](https://www.consul.io/docs/agent/acl-system.html#key-value-rules) | Key/value store operations in the [KV Store API](https://www.consul.io/api/kv.html) |
| [keyring](https://www.consul.io/docs/agent/acl-system.html#keyring-rules) | Keyring operations in the [Keyring API](https://www.consul.io/api/operator/keyring.html) |
| [node](https://www.consul.io/docs/agent/acl-system.html#node-rules) | Node-level catalog operations in the [Catalog API](https://www.consul.io/api/catalog.html), [Health API](https://www.consul.io/api/health.html), [Prepared Query API](https://www.consul.io/api/query.html), [Network Coordinate API](https://www.consul.io/api/coordinate.html), and [Agent API](https://www.consul.io/api/agent.html) |
| [operator](https://www.consul.io/docs/agent/acl-system.html#operator-rules) | Cluster-level operations in the [Operator API](https://www.consul.io/api/operator.html), other than the [Keyring API](https://www.consul.io/api/operator/keyring.html) |
| [query](https://www.consul.io/docs/agent/acl-system.html#prepared-query-rules) | Prepared query operations in the [Prepared Query API](https://www.consul.io/api/query.html) |
| [service](https://www.consul.io/docs/agent/acl-system.html#service-rules) | Service-level catalog operations in the [Catalog API](https://www.consul.io/api/catalog.html), [Health API](https://www.consul.io/api/health.html), [Prepared Query API](https://www.consul.io/api/query.html), and [Agent API](https://www.consul.io/api/agent.html) |
| [session](https://www.consul.io/docs/agent/acl-system.html#session-rules) | Session operations in the [Session API](https://www.consul.io/api/session.html) |

Since Consul snapshots actually contain ACL tokens, the [Snapshot API](https://www.consul.io/api/snapshot.html) requires a token with "write" privileges for the ACL system.

The following resources are not covered by ACL policies:

1. The [Status API](https://www.consul.io/api/status.html) is used by servers when bootstrapping and exposes basic IP and port information about the servers, and does not allow modification of any state.
2. The datacenter listing operation of the [Catalog API](https://www.consul.io/api/catalog.html#list-datacenters) similarly exposes the names of known Consul datacenters, and does not allow modification of any state.
3. The [connect CA roots endpoint](https://www.consul.io/api/connect/ca.html#list-ca-root-certificates) exposes just the public TLS certificate which other systems can use to verify the TLS connection with Consul.

Constructing rules from these policies is covered in detail on the [ACL Rules](https://www.consul.io/docs/agent/acl-rules.html) page.

## [»](https://www.consul.io/docs/agent/acl-system.html" \l "configuring-acls)Configuring ACLs

ACLs are configured using several different configuration options. These are marked as to whether they are set on servers, clients, or both.

| **Configuration Option** | **Servers** | **Clients** | **Purpose** |
| --- | --- | --- | --- |
| [acl.enabled](https://www.consul.io/docs/agent/options.html#acl_enabled) | REQUIRED | REQUIRED | Controls whether ACLs are enabled |
| [acl.default\_policy](https://www.consul.io/docs/agent/options.html#acl_default_policy) | OPTIONAL | N/A | Determines whitelist or blacklist mode |
| [acl.down\_policy](https://www.consul.io/docs/agent/options.html#acl_down_policy) | OPTIONAL | OPTIONAL | Determines what to do when the remote token or policy resolution fails |
| [acl.policy\_ttl](https://www.consul.io/docs/agent/options.html#acl_policy_ttl) | OPTIONAL | OPTIONAL | Determines time-to-live for cached ACL Policies |
| [acl.token\_ttl](https://www.consul.io/docs/agent/options.html#acl_token_ttl) | OPTIONAL | OPTIONAL | Determines time-to-live for cached ACL Tokens |

A number of special tokens can also be configured which allow for bootstrapping the ACL system, or accessing Consul in special situations:

| **Special Token** | **Servers** | **Clients** | **Purpose** |
| --- | --- | --- | --- |
| [acl.tokens.agent\_master](https://www.consul.io/docs/agent/options.html#acl_tokens_agent_master) | OPTIONAL | OPTIONAL | Special token that can be used to access [Agent API](https://www.consul.io/api/agent.html) when remote bearer token resolution fails; used for setting up the cluster such as doing initial join operations, see the [ACL Agent Master Token](https://www.consul.io/docs/agent/acl-system.html#acl-agent-master-token)section for more details |
| [acl.tokens.agent](https://www.consul.io/docs/agent/options.html#acl_tokens_agent) | OPTIONAL | OPTIONAL | Special token that is used for an agent's internal operations, see the [ACL Agent Token](https://www.consul.io/docs/agent/acl-system.html#acl-agent-token) section for more details |
| [acl.tokens.master](https://www.consul.io/docs/agent/options.html#acl_tokens_master) | OPTIONAL | N/A | Special token used to bootstrap the ACL system, see the [Bootstrapping ACLs](https://learn.hashicorp.com/consul/advanced/day-1-operations/acl-guide) guide for more details |
| [acl.tokens.default](https://www.consul.io/docs/agent/options.html#acl_tokens_default) | OPTIONAL | OPTIONAL | Default token to use for client requests where no token is supplied; this is often configured with read-only access to services to enable DNS service discovery on agents |

All of these tokens except the master token can all be introduced or updated via the [/v1/agent/token API](https://www.consul.io/api/agent.html#update-acl-tokens).

#### [»](https://www.consul.io/docs/agent/acl-system.html" \l "acl-agent-master-token)ACL Agent Master Token

Since the [acl.tokens.agent\_master](https://www.consul.io/docs/agent/options.html#acl_tokens_agent_master) is designed to be used when the Consul servers are not available, its policy is managed locally on the agent and does not need to have a token defined on the Consul servers via the ACL API. Once set, it implicitly has the following policy associated with it

agent "<node name of agent>" {

policy = "write"

}

node\_prefix "" {

policy = "read"

}

#### [»](https://www.consul.io/docs/agent/acl-system.html" \l "acl-agent-token)ACL Agent Token

The [acl.tokens.agent](https://www.consul.io/docs/agent/options.html#acl_tokens_agent) is a special token that is used for an agent's internal operations. It isn't used directly for any user-initiated operations like the [acl.tokens.default](https://www.consul.io/docs/agent/options.html#acl_tokens_default), though if the acl.tokens.agent\_token isn't configured the acl.tokens.default will be used. The ACL agent token is used for the following operations by the agent:

1. Updating the agent's node entry using the [Catalog API](https://www.consul.io/api/catalog.html), including updating its node metadata, tagged addresses, and network coordinates
2. Performing [anti-entropy](https://www.consul.io/docs/internals/anti-entropy.html) syncing, in particular reading the node metadata and services registered with the catalog
3. Reading and writing the special \_rexec section of the KV store when executing [consul exec](https://www.consul.io/docs/commands/exec.html) commands

Here's an example policy sufficient to accomplish the above for a node called mynode:

node "mynode" {

policy = "write"

}

service\_prefix "" {

policy = "read"

}

key\_prefix "\_rexec" {

policy = "write"

}

The service\_prefix policy needs read access for any services that can be registered on the agent. If [remote exec is disabled](https://www.consul.io/docs/agent/options.html#disable_remote_exec), the default, then the key\_prefix policy can be omitted.

## [»](https://www.consul.io/docs/agent/acl-system.html" \l "next-steps)Next Steps

Setup ACLs with the [Boostrapping guide](https://www.consul.io/docs/guides/acl.html) or continue reading about [ACL rules](https://www.consul.io/docs/agent/acl-rules.html).

**1.4.0 and later:** This guide only applies in Consul versions 1.4.0 and later. The documentation for the legacy ACL system is [here](https://www.consul.io/docs/guides/acl-legacy.html)

# ACL Rules

Consul provides an optional Access Control List (ACL) system which can be used to control access to data and APIs. To learn more about Consul's ACL review the [ACL system documentation](https://www.consul.io/docs/agent/acl-system.html)

A core part of the ACL system is the rule language, which is used to describe the policy that must be enforced. There are two types of rules: prefix based rules and exact matching rules.

## [»](https://www.consul.io/docs/agent/acl-rules.html" \l "rule-specification)Rule Specification

Rules are composed of a resource, a segment (for some resource areas) and a policy disposition. The general structure of a rule is:

<resource> "<segment>" {

policy = "<policy disposition>"

}

Segmented resource areas allow operators to more finely control access to those resources. Note that not all resource areas are segmented such as the keyring, operator, and acl resources. For those rules they would look like:

<resource> = "<policy disposition>"

Policies can have several control levels:

* [read](https://www.consul.io/docs/agent/acl-rules.html#read): allow the resource to be read but not modified.

* [write](https://www.consul.io/docs/agent/acl-rules.html" \l "write): allow the resource to be read and modified.
* [deny](https://www.consul.io/docs/agent/acl-rules.html#deny): do not allow the resource to be read or modified.
* [list](https://www.consul.io/docs/agent/acl-rules.html#list): allows access to all the keys under a segment in the Consul KV. Note, this policy can only be used with the key\_prefix resource and [acl.enabled\_key\_list\_policy](https://www.consul.io/docs/guides/acl.html#list-policy-for-keys) must be set to true.

When using prefix-based rules, the most specific prefix match determines the action. This allows for flexible rules like an empty prefix to allow read-only access to all resources, along with some specific prefixes that allow write access or that are denied all access. Exact matching rules will only apply to the exact resource specified. The order of precedence for matching rules are, DENY has priority over WRITE or READ and WRITE has priority over READ.

We make use of the [HashiCorp Configuration Language (HCL)](https://github.com/hashicorp/hcl/) to specify rules. This language is human readable and interoperable with JSON making it easy to machine-generate. Rules can make use of one or more policies.

Specification in the HCL format looks like:

# These control access to the key/value store.

key\_prefix "" {

policy = "read"

}

key\_prefix "foo/" {

policy = "write"

}

key\_prefix "foo/private/" {

policy = "deny"

}

# Or for exact key matches

key "foo/bar/secret" {

policy = "deny"

}

# This controls access to cluster-wide Consul operator information.

operator = "read"

This is equivalent to the following JSON input:

{

"key\_prefix": {

"": {

"policy": "read"

},

"foo/": {

"policy": "write"

},

"foo/private/": {

"policy": "deny"

}

},

"key" : {

"foo/bar/secret" : {

"policy" : "deny"

}

},

"operator": "read"

}

The [ACL API](https://www.consul.io/api/acl/acl.html) allows either HCL or JSON to be used to define the content of the rules section of a policy.

Here's a sample request using the HCL form:

$ curl \

--request PUT \

--data \

'{

"Name": "my-app-policy",

"Rules": "key \"\" { policy = \"read\" } key \"foo/\" { policy = \"write\" } key \"foo/private/\" { policy = \"deny\" } operator = \"read\""

}' http://127.0.0.1:8500/v1/acl/policy?token=<token with ACL "write">

Here's an equivalent request using the JSON form:

$ curl \

--request PUT \

--data \

'{

"Name": "my-app-policy",

"Rules": "{\"key\":{\"\":{\"policy\":\"read\"},\"foo/\":{\"policy\":\"write\"},\"foo/private\":{\"policy\":\"deny\"}},\"operator\":\"read\"}"

}' http://127.0.0.1:8500/v1/acl/policy?token=<management token>

On success, the Policy is returned:

{

"CreateIndex": 7,

"Hash": "UMG6QEbV40Gs7Cgi6l/ZjYWUwRS0pIxxusFKyKOt8qI=",

"ID": "5f423562-aca1-53c3-e121-cb0eb2ea1cd3",

"ModifyIndex": 7,

"Name": "my-app-policy",

"Rules": "key \"\" { policy = \"read\" } key \"foo/\" { policy = \"write\" } key \"foo/private/\" { policy = \"deny\" } operator = \"read\""

}

The created policy can now be specified either by name or by ID when [creating a token](https://www.consul.io/docs/guides/acl.html#step-4-create-an-agent-token). This will grant the rules provided to the [bearer of that token](https://www.consul.io/api/index.html#authentication).

Below is a breakdown of each rule type.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "acl-resource-rules)ACL Resource Rules

The acl resource controls access to ACL operations in the [ACL API](https://www.consul.io/api/acl/acl.html).

ACL rules look like this:

acl = "write"

There is only one acl rule allowed per policy and its value is set to one of the [policy dispositions](https://www.consul.io/docs/guides/acl.html#rule-specification). In the example above ACLs may be read or written including discovering any token's secret ID. Snapshotting also requires acl = "write"permissions due to the fact that all the token secrets are contained within the snapshot.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "agent-rules)Agent Rules

The agent and agent\_prefix resources control access to the utility operations in the [Agent API](https://www.consul.io/api/agent.html), such as join and leave. All of the catalog-related operations are covered by the [node or node\_prefix](https://www.consul.io/docs/agent/acl-rules.html#node-rules) and [service or service\_prefix](https://www.consul.io/docs/agent/acl-rules.html#service-rules) policies instead.

Agent rules look like this:

agent\_prefix "" {

policy = "read"

}

agent "foo" {

policy = "write"

}

agent\_prefix "bar" {

policy = "deny"

}

Agent rules are keyed by the node name they apply to. In the example above the rules allow read-only access to any node name by using the empty prefix, read-write access to the node with the exact name foo, and denies all access to any noe name that starts with bar.

Since [Agent API](https://www.consul.io/api/agent.html) utility operations may be reqired before an agent is joined to a cluster, or during an outage of the Consul servers or ACL datacenter, a special token may be configured with [acl.tokens.agent\_master](https://www.consul.io/docs/agent/options.html#acl_tokens_agent_master) to allow write access to these operations even if no ACL resolution capability is available.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "event-rules)Event Rules

The event and event\_prefix resources control access to event operations in the [Event API](https://www.consul.io/api/event.html), such as firing events and listing events.

Event rules look like this:

event\_prefix "" {

policy = "read"

}

event "deploy" {

policy = "write"

}

Event rules are segmented by the event name they apply to. In the example above, the rules allow read-only access to any event, and firing of the "deploy" event.

The [consul exec](https://www.consul.io/docs/commands/exec.html) command uses events with the "\_rexec" prefix during operation, so to enable this feature in a Consul environment with ACLs enabled, you will need to give agents a token with access to this event prefix, in addition to configuring [disable\_remote\_exec](https://www.consul.io/docs/agent/options.html#disable_remote_exec) to false.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "key-value-rules)Key/Value Rules

The key and key\_prefix resources control access to key/value store operations in the [KV API](https://www.consul.io/api/kv.html). Key rules look like this:

key\_prefix "" {

policy = "read"

}

key "foo" {

policy = "write"

}

key "bar" {

policy = "deny"

}

Key rules are segmented by the key name they apply to. In the example above, the rules allow read-only access to any key name with the empty prefix rule, allow read-write access to the "foo" key, and deny access to the "bar" key.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "list-policy-for-keys)List Policy for Keys

Consul 1.0 introduces a new list policy for keys that is only enforced when opted in via the boolean config param "acl.enable\_key\_list\_policy". list controls access to recursively list entries and keys, and enables more fine grained policies. With "acl.enable\_key\_list\_policy", recursive reads via [the KV API](https://www.consul.io/api/kv.html#recurse) with an invalid token result in a 403. Example:

key\_prefix "" {

policy = "deny"

}

key\_prefix "bar" {

policy = "list"

}

key\_prefix "baz" {

policy = "read"

}

In the example above, the rules allow reading the key "baz", and only allow recursive reads on the prefix "bar".

A token with write access on a prefix also has list access. A token with list access on a prefix also has read access on all its suffixes.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "sentinel-integration)Sentinel Integration

Consul Enterprise supports additional optional fields for key write policies for [Sentinel](https://docs.hashicorp.com/sentinel/app/consul/) integration. An example key rule with a Sentinel code policy looks like this:

key "foo" {

policy = "write"

sentinel {

code = <<EOF

import "strings"

main = rule { strings.has\_suffix(value, "bar") }

EOF

enforcementlevel = "hard-mandatory"

}

}

For more detailed information, see the [Consul Sentinel documentation](https://www.consul.io/docs/agent/sentinel.html).

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "keyring-rules)Keyring Rules

The keyring resource controls access to keyring operations in the [Keyring API](https://www.consul.io/api/operator/keyring.html).

Keyring rules look like this:

keyring = "write"

There's only one keyring policy allowed per rule set, and its value is set to one of the policy dispositions. In the example above, the keyring may be read and updated.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "node-rules)Node Rules

The node and node\_prefix resources controls node-level registration and read access to the [Catalog API](https://www.consul.io/api/catalog.html), service discovery with the [Health API](https://www.consul.io/api/health.html), and filters results in [Agent API](https://www.consul.io/api/agent.html) operations like fetching the list of cluster members.

Node rules look like this:

node\_prefix "" {

policy = "read"

}

node "app" {

policy = "write"

}

node "admin" {

policy = "deny"

}

Node rules are segmented by the node name they apply to. In the example above, the rules allow read-only access to any node name with the empty prefix, allow read-write access to the "app" node, and deny all access to the "admin" node.

Agents need to be configured with an [acl.tokens.agent](https://www.consul.io/docs/agent/options.html#acl_tokens_agent) with at least "write" privileges to their own node name in order to register their information with the catalog, such as node metadata and tagged addresses. If this is configured incorrectly, the agent will print an error to the console when it tries to sync its state with the catalog.

Consul's DNS interface is also affected by restrictions on node rules. If the [acl.token.default](https://www.consul.io/docs/agent/options.html#acl_tokens_default) used by the agent does not have "read" access to a given node, then the DNS interface will return no records when queried for it.

When reading from the catalog or retrieving information from the health endpoints, node rules are used to filter the results of the query. This allows for configurations where a token has access to a given service name, but only on an allowed subset of node names.

Node rules come into play when using the [Agent API](https://www.consul.io/api/agent.html) to register node-level checks. The agent will check tokens locally as a check is registered, and Consul also performs periodic [anti-entropy](https://www.consul.io/docs/internals/anti-entropy.html) syncs, which may require an ACL token to complete. To accommodate this, Consul provides two methods of configuring ACL tokens to use for registration events:

1. Using the [acl.tokens.default](https://www.consul.io/docs/agent/options.html#acl_tokens_default) configuration directive. This allows a single token to be configured globally and used during all check registration operations.
2. Providing an ACL token with service and check definitions at registration time. This allows for greater flexibility and enables the use of multiple tokens on the same agent. Examples of what this looks like are available for both [services](https://www.consul.io/docs/agent/services.html) and [checks](https://www.consul.io/docs/agent/checks.html). Tokens may also be passed to the [HTTP API](https://www.consul.io/api/index.html) for operations that require them.

In addition to ACLs, in Consul 0.9.0 and later, the agent must be configured with [enable\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_script_checks) set to true in order to enable script checks.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "operator-rules)Operator Rules

The operator resource controls access to cluster-level operations in the [Operator API](https://www.consul.io/api/operator.html), other than the [Keyring API](https://www.consul.io/api/operator/keyring.html).

Operator rules look like this:

operator = "read"

There's only one operator rule allowed per rule set, and its value is set to one of the policy dispositions. In the example above, the token could be used to query the operator endpoints for diagnostic purposes but not make any changes.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "prepared-query-rules)Prepared Query Rules

The query and query\_prefix resources control access to create, update, and delete prepared queries in the [Prepared Query API](https://www.consul.io/api/query.html). Executing queries is subject to node/node\_prefix and service/service\_prefix policies, as will be explained below.

Query rules look like this:

query\_prefix "" {

policy = "read"

}

query "foo" {

policy = "write"

}

Query rules are segmented by the query name they apply to. In the example above, the rules allow read-only access to any query name with the empty prefix, and allow read-write access to the query named "foo". This allows control of the query namespace to be delegated based on ACLs.

There are a few variations when using ACLs with prepared queries, each of which uses ACLs in one of two ways: open, protected by unguessable IDs or closed, managed by ACL policies. These variations are covered here, with examples:

* Static queries with no Name defined are not controlled by any ACL policies. These types of queries are meant to be ephemeral and not shared to untrusted clients, and they are only reachable if the prepared query ID is known. Since these IDs are generated using the same random ID scheme as ACL Tokens, it is infeasible to guess them. When listing all prepared queries, only a management token will be able to see these types, though clients can read instances for which they have an ID. An example use for this type is a query built by a startup script, tied to a session, and written to a configuration file for a process to use via DNS.
* Static queries with a Name defined are controlled by the query and query\_prefix ACL resources. Clients are required to have an ACL token with permissions on to access that query name. Clients can list or read queries for which they have "read" access based on their prefix, and similar they can update any queries for which they have "write" access. An example use for this type is a query with a well-known name (eg. prod-master-customer-db) that is used and known by many clients to provide geo-failover behavior for a database.
* [Template queries](https://www.consul.io/api/query.html#templates) queries work like static queries with a Name defined, except that a catch-all template with an empty Name requires an ACL token that can write to any query prefix.

When prepared queries are executed via DNS lookups or HTTP requests, the ACL checks are run against the service being queried, similar to how ACLs work with other service lookups. There are several ways the ACL token is selected for this check:

* If an ACL Token was captured when the prepared query was defined, it will be used to perform the service lookup. This allows queries to be executed by clients with lesser or even no ACL Token, so this should be used with care.
* If no ACL Token was captured, then the client's ACL Token will be used to perform the service lookup.
* If no ACL Token was captured and the client has no ACL Token, then the anonymous token will be used to perform the service lookup.

In the common case, the ACL Token of the invoker is used to test the ability to look up a service. If a Token was specified when the prepared query was created, the behavior changes and now the captured ACL Token set by the definer of the query is used when looking up a service.

Capturing ACL Tokens is analogous to [PostgreSQL’s](http://www.postgresql.org/docs/current/static/sql-createfunction.html) SECURITY DEFINER attribute which can be set on functions, and using the client's ACL Token is similar to the complementary SECURITY INVOKER attribute.

Prepared queries were originally introduced in Consul 0.6.0, and ACL behavior remained unchanged through version 0.6.3, but was then changed to allow better management of the prepared query namespace.

These differences are outlined in the table below:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Version <= 0.6.3** | **Version > 0.6.3** |
| Create static query without Name | The ACL Token used to create the prepared query is checked to make sure it can access the service being queried. This token is captured as the Token to use when executing the prepared query. | No ACL policies are used as long as no Name is defined. No Token is captured by default unless specifically supplied by the client when creating the query. |
| Create static query with Name | The ACL Token used to create the prepared query is checked to make sure it can access the service being queried. This token is captured as the Token to use when executing the prepared query. | The client token's query ACL policy is used to determine if the client is allowed to register a query for the given Name. No Token is captured by default unless specifically supplied by the client when creating the query. |
| Manage static query without Name | The ACL Token used to create the query or a token with management privileges must be supplied in order to perform these operations. | Any client with the ID of the query can perform these operations. |
| Manage static query with a Name | The ACL token used to create the query or a token with management privileges must be supplied in order to perform these operations. | Similar to create, the client token's query ACL policy is used to determine if these operations are allowed. |
| List queries | A token with management privileges is required to list any queries. | The client token's query ACL policy is used to determine which queries they can see. Only tokens with management privileges can see prepared queries without Name. |
| Execute query | Since a Token is always captured when a query is created, that is used to check access to the service being queried. Any token supplied by the client is ignored. | The captured token, client's token, or anonymous token is used to filter the results, as described above. |

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "service-rules)Service Rules

The service and service\_prefix resources control service-level registration and read access to the [Catalog API](https://www.consul.io/api/catalog.html) and service discovery with the [Health API](https://www.consul.io/api/health.html).

Service rules look like this:

service\_prefix "" {

policy = "read"

}

service "app" {

policy = "write"

}

service "admin" {

policy = "deny"

}

Service rules are segmented by the service name they apply to. In the example above, the rules allow read-only access to any service name with the empty prefix, allow read-write access to the "app" service, and deny all access to the "admin" service.

Consul's DNS interface is affected by restrictions on service rules. If the [acl.tokens.default](https://www.consul.io/docs/agent/options.html#acl_tokens_default) used by the agent does not have "read" access to a given service, then the DNS interface will return no records when queried for it.

When reading from the catalog or retrieving information from the health endpoints, service rules are used to filter the results of the query.

Service rules come into play when using the [Agent API](https://www.consul.io/api/agent.html) to register services or checks. The agent will check tokens locally as a service or check is registered, and Consul also performs periodic [anti-entropy](https://www.consul.io/docs/internals/anti-entropy.html) syncs, which may require an ACL token to complete. To accommodate this, Consul provides two methods of configuring ACL tokens to use for registration events:

1. Using the [acl.tokens.default](https://www.consul.io/docs/agent/options.html#acl_tokens_default) configuration directive. This allows a single token to be configured globally and used during all service and check registration operations.
2. Providing an ACL token with service and check definitions at registration time. This allows for greater flexibility and enables the use of multiple tokens on the same agent. Examples of what this looks like are available for both [services](https://www.consul.io/docs/agent/services.html) and [checks](https://www.consul.io/docs/agent/checks.html). Tokens may also be passed to the [HTTP API](https://www.consul.io/api/index.html) for operations that require them. **Note:** all tokens passed to an agent are persisted on local disk to allow recovery from restarts. See [-data-dir flag documentation](https://www.consul.io/docs/agent/options.html#acl_token)for notes on securing access.

In addition to ACLs, in Consul 0.9.0 and later, the agent must be configured with [enable\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_script_checks) or[enable\_local\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_local_script_checks) set to true in order to enable script checks.

#### [»](https://www.consul.io/docs/agent/acl-rules.html" \l "session-rules)Session Rules

The session and session\_prefix resources controls access to [Session API](https://www.consul.io/api/session.html) operations.

Session rules look like this:

session\_prefix "" {

policy = "read"

}

session "app" {

policy = "write"

}

session "admin" {

policy = "deny"

}

Session rules are segmented by the node name they apply to. In the example above, the rules allow read-only access to sessions on node name with the empty prefix, allow creating sessions on the node named "app", and deny all access to any sessions on the "admin" node.

# 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 redisservice, 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](https://www.consul.io/docs/agent/options.html#client_addr),[ports.dns](https://www.consul.io/docs/agent/options.html#dns_port), [recursors](https://www.consul.io/docs/agent/options.html#recursors), [domain](https://www.consul.io/docs/agent/options.html#domain), and [dns\_config](https://www.consul.io/docs/agent/options.html#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](https://www.consul.io/docs/agent/options.html), 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](https://www.consul.io/docs/agent/options.html#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.

## [»](https://www.consul.io/docs/agent/dns.html" \l "node-lookups)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 and AAAA records containing the IP address, and TXT records containing the node\_meta values 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

foo.node.consul. 0 IN TXT "meta\_key=meta\_value"

foo.node.consul. 0 IN TXT "value only"

;; AUTHORITY SECTION:

consul. 0 IN SOA ns.consul. postmaster.consul. 1392836399 3600 600 86400 0

By default the TXT records value will match the node's metadata key-value pairs according to [RFC1464](https://www.ietf.org/rfc/rfc1464.txt). Alternatively, the TXT record will only include the node's metadata value when the node's metadata key starts with rfc1035-.

## [»](https://www.consul.io/docs/agent/dns.html" \l "service-lookups)Service Lookups

A service lookup is used to query for service providers. Service queries support two lookup methods: standard and strict [RFC 2782](https://tools.ietf.org/html/rfc2782).

By default, SRV weights are all set at 1, but changing weights is supported using the Weights attribute of the [service definition](https://www.consul.io/docs/agent/services.html).

Note that DNS is limited in size per request, even when performing DNS TCP queries.

For services having many instances (more than 500), it might not be possible to retrieve the complete list of instances for the service.

When DNS SRV response are sent, order is randomized, but weights are not taken into account. In the case of truncation different clients using weighted SRV responses will have partial and inconsistent views of instances weights so the request distribution could be skewed from the intended weights. In that case, it is recommended to use the HTTP API to retrieve the list of nodes.

### [»](https://www.consul.io/docs/agent/dns.html" \l "standard-lookup)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

### [»](https://www.consul.io/docs/agent/dns.html" \l "rfc-2782-lookup)RFC 2782 Lookup

The format for RFC 2782 SRV lookups is:

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

Per [RFC 2782](https://tools.ietf.org/html/rfc2782), 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.consul ANY

; (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.dc1.consul.

;; 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.

### [»](https://www.consul.io/docs/agent/dns.html" \l "prepared-query-lookups)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](https://www.consul.io/api/query.html). 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](https://www.consul.io/api/query.html#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.

### [»](https://www.consul.io/docs/agent/dns.html" \l "connect-capable-service-lookups)Connect-Capable Service Lookups

To find Connect-capable services:

<service>.connect.<domain>

This will find all [Connect-capable](https://www.consul.io/docs/connect/index.html) endpoints for the given service. A Connect-capable endpoint may be both a proxy for a service or a natively integrated Connect application. The DNS interface does not differentiate the two.

Most services will use a [proxy](https://www.consul.io/docs/connect/proxies.html) that handles service discovery automatically and therefore won't use this DNS format. This DNS format is primarily useful for [Connect-native](https://www.consul.io/docs/connect/native.html) applications.

This endpoint currently only finds services within the same datacenter and doesn't support tags. This DNS interface will be expanded over time. If you need more complex behavior, please use the [catalog API](https://www.consul.io/api/catalog.html).

### [»](https://www.consul.io/docs/agent/dns.html" \l "udp-based-dns-queries)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.

## [»](https://www.consul.io/docs/agent/dns.html" \l "caching)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](https://www.consul.io/docs/guides/dns-cache.html).

## [»](https://www.consul.io/docs/agent/dns.html" \l "wan-address-translation)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](https://www.consul.io/docs/agent/options.html#_advertise-wan) and [translate\_wan\_addrs](https://www.consul.io/docs/agent/options.html#translate_wan_addrs) configuration options.

# Configuration

The agent has various configuration options that can be specified via the command-line or via configuration files. All of the configuration options are completely optional. Defaults are specified with their descriptions.

Configuration precedence is evaluated in the following order:

1. Command line arguments
2. Environment Variables
3. Configuration files

When loading configuration, Consul loads the configuration from files and directories in lexical order. For example, configuration file basic\_config.json will be processed before extra\_config.json. Configuration can be in either [HCL](https://github.com/hashicorp/hcl#syntax) or JSON format. Available in Consul 1.0 and later, the HCL support now requires an .hcl or .json extension on all configuration files in order to specify their format.

Configuration specified later will be merged into configuration specified earlier. In most cases, "merge" means that the later version will override the earlier. In some cases, such as event handlers, merging appends the handlers to the existing configuration. The exact merging behavior is specified for each option below.

Consul also supports reloading configuration when it receives the SIGHUP signal. Not all changes are respected, but those that are documented below in the [Reloadable Configuration](https://www.consul.io/docs/agent/options.html#reloadable-configuration) section. The [reload command](https://www.consul.io/docs/commands/reload.html) can also be used to trigger a configuration reload.

## [»](https://www.consul.io/docs/agent/options.html" \l "command-line-options)Command-line Options

The options below are all specified on the command-line.

* [-advertise](https://www.consul.io/docs/agent/options.html" \l "_advertise) - The advertise address is used to change the address that we advertise to other nodes in the cluster. By default, the [-bind](https://www.consul.io/docs/agent/options.html#_bind) address is advertised. However, in some cases, there may be a routable address that cannot be bound. This flag enables gossiping a different address to support this. If this address is not routable, the node will be in a constant flapping state as other nodes will treat the non-routability as a failure. In Consul 1.0 and later this can be set to a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template.

* [-advertise-wan](https://www.consul.io/docs/agent/options.html" \l "_advertise-wan) - The advertise WAN address is used to change the address that we advertise to server nodes joining through the WAN. This can also be set on client agents when used in combination with the [translate\_wan\_addrs](https://www.consul.io/docs/agent/options.html#translate_wan_addrs) configuration option. By default, the [-advertise](https://www.consul.io/docs/agent/options.html#_advertise) address is advertised. However, in some cases all members of all datacenters cannot be on the same physical or virtual network, especially on hybrid setups mixing cloud and private datacenters. This flag enables server nodes gossiping through the public network for the WAN while using private VLANs for gossiping to each other and their client agents, and it allows client agents to be reached at this address when being accessed from a remote datacenter if the remote datacenter is configured with [translate\_wan\_addrs](https://www.consul.io/docs/agent/options.html#translate_wan_addrs). In Consul 1.0 and later this can be set to a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template

* [-bootstrap](https://www.consul.io/docs/agent/options.html" \l "_bootstrap) - This flag is used to control if a server is in "bootstrap" mode. It is important that no more than one server per datacenter be running in this mode. Technically, a server in bootstrap mode is allowed to self-elect as the Raft leader. It is important that only a single node is in this mode; otherwise, consistency cannot be guaranteed as multiple nodes are able to self-elect. It is not recommended to use this flag after a cluster has been bootstrapped.

* [-bootstrap-expect](https://www.consul.io/docs/agent/options.html" \l "_bootstrap_expect) - This flag provides the number of expected servers in the datacenter. Either this value should not be provided or the value must agree with other servers in the cluster. When provided, Consul waits until the specified number of servers are available and then bootstraps the cluster. This allows an initial leader to be elected automatically. This cannot be used in conjunction with the legacy [-bootstrap](https://www.consul.io/docs/agent/options.html#_bootstrap) flag. This flag requires [-server](https://www.consul.io/docs/agent/options.html#_server) mode.

* [-bind](https://www.consul.io/docs/agent/options.html" \l "_bind) - The address that should be bound to for internal cluster communications. This is an IP address that should be reachable by all other nodes in the cluster. By default, this is "0.0.0.0", meaning Consul will bind to all addresses on the local machine and will [advertise](https://www.consul.io/docs/agent/options.html#_advertise) the first available private IPv4 address to the rest of the cluster. If there are **multiple private IPv4 addresses** available, Consul will exit with an error at startup. If you specify "[::]", Consul will[advertise](https://www.consul.io/docs/agent/options.html#_advertise) the first available public IPv6 address. If there are **multiple public IPv6 addresses** available, Consul will exit with an error at startup. Consul uses both TCP and UDP and the same port for both. If you have any firewalls, be sure to allow both protocols. **In Consul 1.0 and later this can be set to a**[**go-sockaddr**](https://godoc.org/github.com/hashicorp/go-sockaddr/template)**template that needs to resolve to a single address.**

* [-serf-wan-bind](https://www.consul.io/docs/agent/options.html" \l "_serf_wan_bind) - The address that should be bound to for Serf WAN gossip communications. By default, the value follows the same rules as [-bind command-line flag](https://www.consul.io/docs/agent/options.html#_bind), and if this is not specified, the -bind option is used. This is available in Consul 0.7.1 and later. In Consul 1.0 and later this can be set to a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template

* [-serf-lan-bind](https://www.consul.io/docs/agent/options.html" \l "_serf_lan_bind) - The address that should be bound to for Serf LAN gossip communications. This is an IP address that should be reachable by all other LAN nodes in the cluster. By default, the value follows the same rules as [-bindcommand-line flag](https://www.consul.io/docs/agent/options.html#_bind), and if this is not specified, the -bind option is used. This is available in Consul 0.7.1 and later. In Consul 1.0 and later this can be set to a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template

* [-client](https://www.consul.io/docs/agent/options.html" \l "_client) - The address to which Consul will bind client interfaces, including the HTTP and DNS servers. By default, this is "127.0.0.1", allowing only loopback connections. In Consul 1.0 and later this can be set to a space-separated list of addresses to bind to, or a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template that can potentially resolve to multiple addresses.

* [-config-file](https://www.consul.io/docs/agent/options.html" \l "_config_file) - A configuration file to load. For more information on the format of this file, read the [Configuration Files](https://www.consul.io/docs/agent/options.html#configuration_files) section. This option can be specified multiple times to load multiple configuration files. If it is specified multiple times, configuration files loaded later will merge with configuration files loaded earlier. During a config merge, single-value keys (string, int, bool) will simply have their values replaced while list types will be appended together.

* [-config-dir](https://www.consul.io/docs/agent/options.html" \l "_config_dir) - A directory of configuration files to load. Consul will load all files in this directory with the suffix ".json" or ".hcl". The load order is alphabetical, and the the same merge routine is used as with the [config-file](https://www.consul.io/docs/agent/options.html#_config_file)option above. This option can be specified multiple times to load multiple directories. Sub-directories of the config directory are not loaded. For more information on the format of the configuration files, see the [Configuration Files](https://www.consul.io/docs/agent/options.html#configuration_files)section.

* [-config-format](https://www.consul.io/docs/agent/options.html" \l "_config_format) - The format of the configuration files to load. Normally, Consul detects the format of the config files from the ".json" or ".hcl" extension. Setting this option to either "json" or "hcl" forces Consul to interpret any file with or without extension to be interpreted in that format.

* [-data-dir](https://www.consul.io/docs/agent/options.html" \l "_data_dir) - This flag provides a data directory for the agent to store state. This is required for all agents. The directory should be durable across reboots. This is especially critical for agents that are running in server mode as they must be able to persist cluster state. Additionally, the directory must support the use of filesystem locking, meaning some types of mounted folders (e.g. VirtualBox shared folders) may not be suitable. **Note:** both server and non-server agents may store ACL tokens in the state in this directory so read access may grant access to any tokens on servers and to any tokens used during service registration on non-servers. On Unix-based platforms the files are written with 0600 permissions so you should ensure only trusted processes can execute as the same user as Consul. On Windows, you should ensure the directory has suitable permissions configured as these will be inherited.

* [-datacenter](https://www.consul.io/docs/agent/options.html" \l "_datacenter) - This flag controls the datacenter in which the agent is running. If not provided, it defaults to "dc1". Consul has first-class support for multiple datacenters, but it relies on proper configuration. Nodes in the same datacenter should be on a single LAN.

* [-dev](https://www.consul.io/docs/agent/options.html" \l "_dev) - Enable development server mode. This is useful for quickly starting a Consul agent with all persistence options turned off, enabling an in-memory server which can be used for rapid prototyping or developing against the API. In this mode, [Connect is enabled](https://www.consul.io/docs/connect/configuration.html) and will by default create a new root CA certificate on startup. This mode is **not** intended for production use as it does not write any data to disk. The gRPC port is also defaulted to 8502 in this mode.

* [-disable-host-node-id](https://www.consul.io/docs/agent/options.html" \l "_disable_host_node_id) - Setting this to true will prevent Consul from using information from the host to generate a deterministic node ID, and will instead generate a random node ID which will be persisted in the data directory. This is useful when running multiple Consul agents on the same host for testing. This defaults to false in Consul prior to version 0.8.5 and in 0.8.5 and later defaults to true, so you must opt-in for host-based IDs. Host-based IDs are generated using <https://github.com/shirou/gopsutil/tree/master/host>, which is shared with HashiCorp's [Nomad](https://www.nomadproject.io/), so if you opt-in to host-based IDs then Consul and Nomad will use information on the host to automatically assign the same ID in both systems.

* [-disable-keyring-file](https://www.consul.io/docs/agent/options.html" \l "_disable_keyring_file) - If set, the keyring will not be persisted to a file. Any installed keys will be lost on shutdown, and only the given -encrypt key will be available on startup. This defaults to false.

* [-dns-port](https://www.consul.io/docs/agent/options.html" \l "_dns_port) - the DNS port to listen on. This overrides the default port 8600. This is available in Consul 0.7 and later.

* [-domain](https://www.consul.io/docs/agent/options.html" \l "_domain) - By default, Consul responds to DNS queries in the "consul." domain. This flag can be used to change that domain. All queries in this domain are assumed to be handled by Consul and will not be recursively resolved.

* [-enable-script-checks](https://www.consul.io/docs/agent/options.html" \l "_enable_script_checks) This controls whether [health checks that execute scripts](https://www.consul.io/docs/agent/checks.html) are enabled on this agent, and defaults to false so operators must opt-in to allowing these. This was added in Consul 0.9.0.

**Security Warning:** Enabling script checks in some configurations may introduce a remote execution vulnerability which is known to be targeted by malware. We strongly recommend -enable-local-script-checks instead. See [this blog post](https://www.hashicorp.com/blog/protecting-consul-from-rce-risk-in-specific-configurations) for more details.

* [-enable-local-script-checks](https://www.consul.io/docs/agent/options.html" \l "_enable_local_script_checks) Like [enable\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_script_checks), but only enable them when they are defined in the local configuration files. Script checks defined in HTTP API registrations will still not be allowed.

* [-encrypt](https://www.consul.io/docs/agent/options.html" \l "_encrypt) - Specifies the secret key to use for encryption of Consul network traffic. This key must be 16-bytes that are Base64-encoded. The easiest way to create an encryption key is to use [consul keygen](https://www.consul.io/docs/commands/keygen.html). All nodes within a cluster must share the same encryption key to communicate. The provided key is automatically persisted to the data directory and loaded automatically whenever the agent is restarted. This means that to encrypt Consul's gossip protocol, this option only needs to be provided once on each agent's initial startup sequence. If it is provided after Consul has been initialized with an encryption key, then the provided key is ignored and a warning will be displayed.

* [-grpc-port](https://www.consul.io/docs/agent/options.html" \l "_grpc_port) - the gRPC API port to listen on. Default -1 (gRPC disabled). See [ports](https://www.consul.io/docs/agent/options.html#ports) documentation for more detail.

* [-hcl](https://www.consul.io/docs/agent/options.html" \l "_hcl) - A HCL configuration fragment. This HCL configuration fragment is appended to the configuration and allows to specify the full range of options of a config file on the command line. This option can be specified multiple times. This was added in Consul 1.0.

* [-http-port](https://www.consul.io/docs/agent/options.html" \l "_http_port) - the HTTP API port to listen on. This overrides the default port 8500. This option is very useful when deploying Consul to an environment which communicates the HTTP port through the environment e.g. PaaS like CloudFoundry, allowing you to set the port directly via a Procfile.

* [-log-file](https://www.consul.io/docs/agent/options.html" \l "_log_file) - to redirect all the Consul agent log messages to a file. This can be specified with the complete path along with the name of the log. In case the path doesn't have the filename, the filename defaults to consul-{timestamp}.log. Can be combined with [-log-rotate-bytes](https://www.consul.io/docs/agent/options.html#_log_rotate_bytes) and [-log-rotate-duration](https://www.consul.io/docs/agent/options.html#_log_rotate_duration)for a fine-grained log rotation experience.

* [-log-rotate-bytes](https://www.consul.io/docs/agent/options.html" \l "_log_rotate_bytes) - to specify the number of bytes that should be written to a log before it needs to be rotated. Unless specified, there is no limit to the number of bytes that can be written to a log file.

* [-log-rotate-duration](https://www.consul.io/docs/agent/options.html" \l "_log_rotate_duration) - to specify the maximum duration a log should be written to before it needs to be rotated. Unless specified, logs are rotated on a daily basis (24 hrs).

* [-join](https://www.consul.io/docs/agent/options.html" \l "_join) - Address of another agent to join upon starting up. This can be specified multiple times to specify multiple agents to join. If Consul is unable to join with any of the specified addresses, agent startup will fail. By default, the agent won't join any nodes when it starts up. Note that using [retry\_join](https://www.consul.io/docs/agent/options.html#retry_join) could be more appropriate to help mitigate node startup race conditions when automating a Consul cluster deployment.

In Consul 1.1.0 and later this can be set to a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template

* [-retry-join](https://www.consul.io/docs/agent/options.html" \l "retry-join) - Similar to [-join](https://www.consul.io/docs/agent/options.html#_join) but allows retrying a join if the first attempt fails. This is useful for cases where you know the address will eventually be available. The list can contain IPv4, IPv6, or DNS addresses. In Consul 1.1.0 and later this can be set to a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template. If Consul is running on the non-default Serf LAN port, this must be specified as well. IPv6 must use the "bracketed" syntax. If multiple values are given, they are tried and retried in the order listed until the first succeeds. Here are some examples:
* *# Using a DNS entry*
* $ consul agent -retry-join "consul.domain.internal"
* *# Using IPv4*
* $ consul agent -retry-join "10.0.4.67"
* *# Using IPv6*
* $ consul agent -retry-join "[::1]:8301"

### [»](https://www.consul.io/docs/agent/options.html" \l "cloud-auto-joining)Cloud Auto-Joining

As of Consul 0.9.1, retry-join accepts a unified interface using the [go-discover](https://github.com/hashicorp/go-discover) library for doing automatic cluster joining using cloud metadata. For more information, see the [Cloud Auto-join page](https://www.consul.io/docs/agent/cloud-auto-join.html).

*# Using Cloud Auto-Joining*

$ consul agent -retry-join "provider=aws tag\_key=..."

* [-retry-interval](https://www.consul.io/docs/agent/options.html" \l "_retry_interval) - Time to wait between join attempts. Defaults to 30s.

* [-retry-max](https://www.consul.io/docs/agent/options.html" \l "_retry_max) - The maximum number of [-join](https://www.consul.io/docs/agent/options.html#_join) attempts to be made before exiting with return code 1. By default, this is set to 0 which is interpreted as infinite retries.

* [-join-wan](https://www.consul.io/docs/agent/options.html" \l "_join_wan) - Address of another wan agent to join upon starting up. This can be specified multiple times to specify multiple WAN agents to join. If Consul is unable to join with any of the specified addresses, agent startup will fail. By default, the agent won't [-join-wan](https://www.consul.io/docs/agent/options.html#_join_wan) any nodes when it starts up.

In Consul 1.1.0 and later this can be set to a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template.

* [-retry-join-wan](https://www.consul.io/docs/agent/options.html" \l "_retry_join_wan) - Similar to [retry-join](https://www.consul.io/docs/agent/options.html#_retry_join) but allows retrying a wan join if the first attempt fails. This is useful for cases where we know the address will become available eventually. As of Consul 0.9.3 [Cloud Auto-Joining](https://www.consul.io/docs/agent/options.html#cloud-auto-joining) is supported as well.

In Consul 1.1.0 and later this can be set to a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template

* [-retry-interval-wan](https://www.consul.io/docs/agent/options.html" \l "_retry_interval_wan) - Time to wait between [-join-wan](https://www.consul.io/docs/agent/options.html#_join_wan) attempts. Defaults to 30s.

* [-retry-max-wan](https://www.consul.io/docs/agent/options.html" \l "_retry_max_wan) - The maximum number of [-join-wan](https://www.consul.io/docs/agent/options.html#_join_wan) attempts to be made before exiting with return code 1. By default, this is set to 0 which is interpreted as infinite retries.

* [-log-level](https://www.consul.io/docs/agent/options.html" \l "_log_level) - The level of logging to show after the Consul agent has started. This defaults to "info". The available log levels are "trace", "debug", "info", "warn", and "err". You can always connect to an agent via [consul monitor](https://www.consul.io/docs/commands/monitor.html) and use any log level. Also, the log level can be changed during a config reload.

* [-node](https://www.consul.io/docs/agent/options.html" \l "_node) - The name of this node in the cluster. This must be unique within the cluster. By default this is the hostname of the machine.

* [-node-id](https://www.consul.io/docs/agent/options.html" \l "_node_id) - Available in Consul 0.7.3 and later, this is a unique identifier for this node across all time, even if the name of the node or address changes. This must be in the form of a hex string, 36 characters long, such asadf4238a-882b-9ddc-4a9d-5b6758e4159e. If this isn't supplied, which is the most common case, then the agent will generate an identifier at startup and persist it in the [data directory](https://www.consul.io/docs/agent/options.html#_data_dir) so that it will remain the same across agent restarts. Information from the host will be used to generate a deterministic node ID if possible, unless [-disable-host-node-id](https://www.consul.io/docs/agent/options.html#_disable_host_node_id) is set to true.

* [-node-meta](https://www.consul.io/docs/agent/options.html" \l "_node_meta) - Available in Consul 0.7.3 and later, this specifies an arbitrary metadata key/value pair to associate with the node, of the form key:value. This can be specified multiple times. Node metadata pairs have the following restrictions:
  + A maximum of 64 key/value pairs can be registered per node.
  + Metadata keys must be between 1 and 128 characters (inclusive) in length
  + Metadata keys must contain only alphanumeric, -, and \_ characters.
  + Metadata keys must not begin with the consul- prefix; that is reserved for internal use by Consul.
  + Metadata values must be between 0 and 512 (inclusive) characters in length.
  + Metadata values for keys beginning with rfc1035- are encoded verbatim in DNS TXT requests, otherwise the metadata kv-pair is encoded according [RFC1464](https://www.ietf.org/rfc/rfc1464.txt).

* [-pid-file](https://www.consul.io/docs/agent/options.html" \l "_pid_file) - This flag provides the file path for the agent to store its PID. This is useful for sending signals (for example, SIGINT to close the agent or SIGHUP to update check definite

* [-protocol](https://www.consul.io/docs/agent/options.html" \l "_protocol) - The Consul protocol version to use. This defaults to the latest version. This should be set only when [upgrading](https://www.consul.io/docs/upgrading.html). You can view the protocol versions supported by Consul by running consul -v.

* [-raft-protocol](https://www.consul.io/docs/agent/options.html" \l "_raft_protocol) - This controls the internal version of the Raft consensus protocol used for server communications. This must be set to 3 in order to gain access to Autopilot features, with the exception of [cleanup\_dead\_servers](https://www.consul.io/docs/agent/options.html#cleanup_dead_servers). Defaults to 3 in Consul 1.0.0 and later (defaulted to 2 previously). See [Raft Protocol Version Compatibility](https://www.consul.io/docs/upgrade-specific.html#raft-protocol-version-compatibility) for more details.

* [-raft-snapshot-threshold](https://www.consul.io/docs/agent/options.html" \l "_raft_snapshot_threshold) - This controls the minimum number of raft commit entries between snapshots that are saved to disk. This is a low-level parameter that should rarely need to be changed. Very busy clusters experiencing excessive disk IO may increase this value to reduce disk IO, and minimize the chances of all servers taking snapshots at the same time. Increasing this trades off disk IO for disk space since the log will grow much larger and the space in the raft.db file can't be reclaimed till the next snapshot. Servers may take longer to recover from crashes or failover if this is increased significantly as more logs will need to be replayed. In Consul 1.1.0 and later this defaults to 16384, and in prior versions it was set to 8192.

* [-raft-snapshot-interval](https://www.consul.io/docs/agent/options.html" \l "_raft_snapshot_interval) - This controls how often servers check if they need to save a snapshot to disk. his is a low-level parameter that should rarely need to be changed. Very busy clusters experiencing excessive disk IO may increase this value to reduce disk IO, and minimize the chances of all servers taking snapshots at the same time. Increasing this trades off disk IO for disk space since the log will grow much larger and the space in the raft.db file can't be reclaimed till the next snapshot. Servers may take longer to recover from crashes or failover if this is increased significantly as more logs will need to be replayed. In Consul 1.1.0 and later this defaults to 30s, and in prior versions it was set to 5s.

* [-recursor](https://www.consul.io/docs/agent/options.html" \l "_recursor) - Specifies the address of an upstream DNS server. This option may be provided multiple times, and is functionally equivalent to the [recursors configuration option](https://www.consul.io/docs/agent/options.html#recursors).

* [-rejoin](https://www.consul.io/docs/agent/options.html" \l "_rejoin) - When provided, Consul will ignore a previous leave and attempt to rejoin the cluster when starting. By default, Consul treats leave as a permanent intent and does not attempt to join the cluster again when starting. This flag allows the previous state to be used to rejoin the cluster.

* [-segment](https://www.consul.io/docs/agent/options.html" \l "_segment) - (Enterprise-only) This flag is used to set the name of the network segment the agent belongs to. An agent can only join and communicate with other agents within its network segment. See the [Network Segments Guide](https://www.consul.io/docs/guides/segments.html) for more details. By default, this is an empty string, which is the default network segment.

* [-serf-lan-port](https://www.consul.io/docs/agent/options.html" \l "_serf_lan_port) - the Serf LAN port to listen on. This overrides the default Serf LAN port 8301. This is available in Consul 1.2.2 and later.

* [-serf-wan-port](https://www.consul.io/docs/agent/options.html" \l "_serf_wan_port) - the Serf WAN port to listen on. This overrides the default Serf WAN port 8302. This is available in Consul 1.2.2 and later.

* [-server](https://www.consul.io/docs/agent/options.html" \l "_server) - This flag is used to control if an agent is in server or client mode. When provided, an agent will act as a Consul server. Each Consul cluster must have at least one server and ideally no more than 5 per datacenter. All servers participate in the Raft consensus algorithm to ensure that transactions occur in a consistent, linearizable manner. Transactions modify cluster state, which is maintained on all server nodes to ensure availability in the case of node failure. Server nodes also participate in a WAN gossip pool with server nodes in other datacenters. Servers act as gateways to other datacenters and forward traffic as appropriate.

* [-server-port](https://www.consul.io/docs/agent/options.html" \l "_server_port) - the server RPC port to listen on. This overrides the default server RPC port 8300. This is available in Consul 1.2.2 and later.

* [-non-voting-server](https://www.consul.io/docs/agent/options.html" \l "_non_voting_server) - (Enterprise-only) This flag is used to make the server not participate in the Raft quorum, and have it only receive the data replication stream. This can be used to add read scalability to a cluster in cases where a high volume of reads to servers are needed.

* [-syslog](https://www.consul.io/docs/agent/options.html" \l "_syslog) - This flag enables logging to syslog. This is only supported on Linux and OSX. It will result in an error if provided on Windows.

* [-ui](https://www.consul.io/docs/agent/options.html" \l "_ui) - Enables the built-in web UI server and the required HTTP routes. This eliminates the need to maintain the Consul web UI files separately from the binary.

* [-ui-dir](https://www.consul.io/docs/agent/options.html" \l "_ui_dir) - This flag provides the directory containing the Web UI resources for Consul. This will automatically enable the Web UI. The directory must be readable to the agent. Starting with Consul version 0.7.0 and later, the Web UI assets are included in the binary so this flag is no longer necessary; specifying only the -ui flag is enough to enable the Web UI. Specifying both the '-ui' and '-ui-dir' flags will result in an error.

## [»](https://www.consul.io/docs/agent/options.html" \l "configuration-files)Configuration Files

In addition to the command-line options, configuration can be put into files. This may be easier in certain situations, for example when Consul is being configured using a configuration management system.

The configuration files are JSON formatted, making them easily readable and editable by both humans and computers. The configuration is formatted as a single JSON object with configuration within it.

Configuration files are used for more than just setting up the agent, they are also used to provide check and service definitions. These are used to announce the availability of system servers to the rest of the cluster. They are documented separately under [check configuration](https://www.consul.io/docs/agent/checks.html) and [service configuration](https://www.consul.io/docs/agent/services.html) respectively. The service and check definitions support being updated during a reload.

#### [»](https://www.consul.io/docs/agent/options.html" \l "example-configuration-file)Example Configuration File

{

"datacenter": "east-aws",

"data\_dir": "/opt/consul",

"log\_level": "INFO",

"node\_name": "foobar",

"server": **true**,

"watches": [

{

"type": "checks",

"handler": "/usr/bin/health-check-handler.sh"

}

],

"telemetry": {

"statsite\_address": "127.0.0.1:2180"

}

}

#### [»](https://www.consul.io/docs/agent/options.html" \l "example-configuration-file-with-tls)Example Configuration File, with TLS

{

"datacenter": "east-aws",

"data\_dir": "/opt/consul",

"log\_level": "INFO",

"node\_name": "foobar",

"server": **true**,

"addresses": {

"https": "0.0.0.0"

},

"ports": {

"https": 8501

},

"key\_file": "/etc/pki/tls/private/my.key",

"cert\_file": "/etc/pki/tls/certs/my.crt",

"ca\_file": "/etc/pki/tls/certs/ca-bundle.crt"

}

See, especially, the use of the ports setting:

"ports": {

"https": 8501

}

Consul will not enable TLS for the HTTP API unless the https port has been assigned a port number > 0. We recommend using 8501 for https as this default will automatically work with some tooling.

#### [»](https://www.consul.io/docs/agent/options.html" \l "configuration-key-reference)Configuration Key Reference

* [acl](https://www.consul.io/docs/agent/options.html" \l "acl) - This object allows a number of sub-keys to be set which controls the ACL system. Configuring the ACL system within the ACL stanza was added in Consul 1.4.0

The following sub-keys are available:

* + [enabled](https://www.consul.io/docs/agent/options.html" \l "acl_enabled) - Enables ACLs.

* + [policy\_ttl](https://www.consul.io/docs/agent/options.html" \l "acl_policy_ttl) - Used to control Time-To-Live caching of ACL policies. By default, this is 30 seconds. This setting has a major performance impact: reducing it will cause more frequent refreshes while increasing it reduces the number of refreshes. However, because the caches are not actively invalidated, ACL policy may be stale up to the TTL value.

* + [token\_ttl](https://www.consul.io/docs/agent/options.html" \l "acl_token_ttl) - Used to control Time-To-Live caching of ACL tokens. By default, this is 30 seconds. This setting has a major performance impact: reducing it will cause more frequent refreshes while increasing it reduces the number of refreshes. However, because the caches are not actively invalidated, ACL token may be stale up to the TTL value.

* + [down\_policy](https://www.consul.io/docs/agent/options.html" \l "acl_down_policy) - Either "allow", "deny", "extend-cache" or "async-cache"; "extend-cache" is the default. In the case that a policy or token cannot be read from the [primary\_datacenter](https://www.consul.io/docs/agent/options.html#primary_datacenter) or leader node, the down policy is applied. In "allow" mode, all actions are permitted, "deny" restricts all operations, and "extend-cache" allows any cached objects to be used, ignoring their TTL values. If a non-cached ACL is used, "extend-cache" acts like "deny". The value "async-cache" acts the same way as "extend-cache" but performs updates asynchronously when ACL is present but its TTL is expired, thus, if latency is bad between the primary and secondary datacenters, latency of operations is not impacted.

* + [default\_policy](https://www.consul.io/docs/agent/options.html" \l "acl_default_policy) - Either "allow" or "deny"; defaults to "allow" but this will be changed in a future major release. The default policy controls the behavior of a token when there is no matching rule. In "allow" mode, ACLs are a blacklist: any operation not specifically prohibited is allowed. In "deny" mode, ACLs are a whitelist: any operation not specifically allowed is blocked. Note: this will not take effect until you've enabled ACLs.

* + [enable\_key\_list](https://www.consul.io/docs/agent/options.html" \l "acl_enable_key_list) - Either "enabled" or "disabled", defaults to "disabled". When enabled, the list permission will be required on the prefix being recursively read from the KV store. Regardless of being enabled, the full set of KV entries under the prefix will be filtered to remove any entries that the request's ACL token does not grant at least read persmissions. This option is only available in Consul 1.0 and newer.

* + [enable\_token\_replication](https://www.consul.io/docs/agent/options.html" \l "acl_enable_token_replication) - By default secondary Consul datacenters will perform replication of only ACL policies. Setting this configuration will also enable ACL token replication.

* + [enable\_token\_persistence](https://www.consul.io/docs/agent/options.html" \l "acl_enable_token_persistence) - Either true or false. When true tokens set using the API will be persisted to disk and reloaded when an agent restarts.

* + [tokens](https://www.consul.io/docs/agent/options.html" \l "acl_tokens) - This object holds all of the configured ACL tokens for the agents usage.

* + - [master](https://www.consul.io/docs/agent/options.html" \l "acl_tokens_master) - Only used for servers in the [primary\_datacenter](https://www.consul.io/docs/agent/options.html#primary_datacenter). This token will be created with management-level permissions if it does not exist. It allows operators to bootstrap the ACL system with a token Secret ID that is well-known.   
        
      The master token is only installed when a server acquires cluster leadership. If you would like to install or change the acl\_master\_token, set the new value for master in the configuration for all servers. Once this is done, restart the current leader to force a leader election. If the master token is not supplied, then the servers do not create a master token. When you provide a value, it should be a UUID. To maintain backwards compatibility and an upgrade path this restriction is not currently enforced but will be in a future major Consul release.

* + - [default](https://www.consul.io/docs/agent/options.html" \l "acl_tokens_default) - When provided, the agent will use this token when making requests to the Consul servers. Clients can override this token on a per-request basis by providing the "?token" query parameter. When not provided, the empty token, which maps to the 'anonymous' ACL token, is used.

* + - [agent](https://www.consul.io/docs/agent/options.html" \l "acl_tokens_agent) - Used for clients and servers to perform internal operations. If this isn't specified, then the[default](https://www.consul.io/docs/agent/options.html#acl_tokens_default) will be used.   
        
      This token must at least have write access to the node name it will register as in order to set any of the node-level information in the catalog such as metadata, or the node's tagged addresses. There are other places this token is used, please see [ACL Agent Token](https://www.consul.io/docs/guides/acl.html#acl-agent-token) for more details.

* + - [agent\_master](https://www.consul.io/docs/agent/options.html" \l "acl_tokens_agent_master) - Used to access [agent endpoints](https://www.consul.io/api/agent.html) that require agent read or write privileges, or node read privileges, even if Consul servers aren't present to validate any tokens. This should only be used by operators during outages, regular ACL tokens should normally be used by applications.

* + - [replication](https://www.consul.io/docs/agent/options.html" \l "acl_tokens_replication) - The ACL token used to authorize secondary datacenters with the primary datacenter for replication operations. This token is required for servers outside the [primary\_datacenter](https://www.consul.io/docs/agent/options.html#primary_datacenter) when ACLs are enabled. This token may be provided later using the [agent token API](https://www.consul.io/api/agent.html#update-acl-tokens) on each server. This token must have at least "read" permissions on ACL data but if ACL token replication is enabled then it must have "write" permissions. This also enables Connect replication in Consul Enterprise, for which the token will require both operator "write" and intention "read" permissions for replicating CA and Intention data.

* [acl\_datacenter](https://www.consul.io/docs/agent/options.html" \l "acl_datacenter) - **This field is deprecated in Consul 1.4.0. See the**[**primary\_datacenter**](https://www.consul.io/docs/agent/options.html#primary_datacenter)**field instead.**

This designates the datacenter which is authoritative for ACL information. It must be provided to enable ACLs. All servers and datacenters must agree on the ACL datacenter. Setting it on the servers is all you need for cluster-level enforcement, but for the APIs to forward properly from the clients, it must be set on them too. In Consul 0.8 and later, this also enables agent-level enforcement of ACLs. Please see the [ACL Guide](https://www.consul.io/docs/guides/acl.html) for more details.

* [acl\_default\_policy](https://www.consul.io/docs/agent/options.html" \l "acl_default_policy_legacy) - **Deprecated in Consul 1.4.0. See the**[**acl.default\_policy**](https://www.consul.io/docs/agent/options.html#acl_default_policy)**field instead.** Either "allow" or "deny"; defaults to "allow". The default policy controls the behavior of a token when there is no matching rule. In "allow" mode, ACLs are a blacklist: any operation not specifically prohibited is allowed. In "deny" mode, ACLs are a whitelist: any operation not specifically allowed is blocked. Note: this will not take effect until you've set primary\_datacenter to enable ACL support.

* [acl\_down\_policy](https://www.consul.io/docs/agent/options.html" \l "acl_down_policy_legacy) - **Deprecated in Consul 1.4.0. See the**[**acl.down\_policy**](https://www.consul.io/docs/agent/options.html#acl_down_policy)**field instead.**Either "allow", "deny", "extend-cache" or "async-cache"; "extend-cache" is the default. In the case that the policy for a token cannot be read from the [primary\_datacenter](https://www.consul.io/docs/agent/options.html#primary_datacenter) or leader node, the down policy is applied. In "allow" mode, all actions are permitted, "deny" restricts all operations, and "extend-cache" allows any cached ACLs to be used, ignoring their TTL values. If a non-cached ACL is used, "extend-cache" acts like "deny". The value "async-cache" acts the same way as "extend-cache" but performs updates asynchronously when ACL is present but its TTL is expired, thus, if latency is bad between ACL authoritative and other datacenters, latency of operations is not impacted.

* [acl\_agent\_master\_token](https://www.consul.io/docs/agent/options.html" \l "acl_agent_master_token_legacy) - **Deprecated in Consul 1.4.0. See the**[**acl.tokens.agent\_master**](https://www.consul.io/docs/agent/options.html#acl_tokens_agent_master)**field instead.** Used to access [agent endpoints](https://www.consul.io/api/agent.html) that require agent read or write privileges, or node read privileges, even if Consul servers aren't present to validate any tokens. This should only be used by operators during outages, regular ACL tokens should normally be used by applications. This was added in Consul 0.7.2 and is only used when[acl\_enforce\_version\_8](https://www.consul.io/docs/agent/options.html#acl_enforce_version_8) is set to true. Please see [ACL Agent Master Token](https://www.consul.io/docs/guides/acl.html#acl-agent-master-token) for more details.

* [acl\_agent\_token](https://www.consul.io/docs/agent/options.html" \l "acl_agent_token_legacy) - **Deprecated in Consul 1.4.0. See the**[**acl.tokens.agent**](https://www.consul.io/docs/agent/options.html#acl_tokens_agent)**field instead.** Used for clients and servers to perform internal operations. If this isn't specified, then the [acl\_token](https://www.consul.io/docs/agent/options.html#acl_token) will be used. This was added in Consul 0.7.2.

This token must at least have write access to the node name it will register as in order to set any of the node-level information in the catalog such as metadata, or the node's tagged addresses. There are other places this token is used, please see [ACL Agent Token](https://www.consul.io/docs/guides/acl.html#acl-agent-token) for more details.

* [acl\_enforce\_version\_8](https://www.consul.io/docs/agent/options.html" \l "acl_enforce_version_8) - **Deprecated in Consul 1.4.0** Used for clients and servers to determine if enforcement should occur for new ACL policies being previewed before Consul 0.8. Added in Consul 0.7.2, this defaults to false in versions of Consul prior to 0.8, and defaults to true in Consul 0.8 and later. This helps ease the transition to the new ACL features by allowing policies to be in place before enforcement begins. Please see the [ACL Guide](https://www.consul.io/docs/guides/acl.html#version_8_acls) for more details.

* [acl\_master\_token](https://www.consul.io/docs/agent/options.html" \l "acl_master_token_legacy) - **Deprecated in Consul 1.4.0. See the**[**acl.tokens.master**](https://www.consul.io/docs/agent/options.html#acl_tokens_master)**field instead.** Only used for servers in the [primary\_datacenter](https://www.consul.io/docs/agent/options.html#primary_datacenter). This token will be created with management-level permissions if it does not exist. It allows operators to bootstrap the ACL system with a token ID that is well-known.

The acl\_master\_token is only installed when a server acquires cluster leadership. If you would like to install or change the acl\_master\_token, set the new value for acl\_master\_token in the configuration for all servers. Once this is done, restart the current leader to force a leader election. If the acl\_master\_token is not supplied, then the servers do not create a master token. When you provide a value, it can be any string value. Using a UUID would ensure that it looks the same as the other tokens, but isn't strictly necessary.

* [acl\_replication\_token](https://www.consul.io/docs/agent/options.html" \l "acl_replication_token_legacy) - **Deprecated in Consul 1.4.0. See the**[**acl.tokens.replication**](https://www.consul.io/docs/agent/options.html#acl_tokens_replication)**field instead.** Only used for servers outside the [primary\_datacenter](https://www.consul.io/docs/agent/options.html#primary_datacenter) running Consul 0.7 or later. When provided, this will enable [ACL replication](https://www.consul.io/docs/guides/acl.html#replication) using this token to retrieve and replicate the ACLs to the non-authoritative local datacenter. In Consul 0.9.1 and later you can enable ACL replication using [enable\_acl\_replication](https://www.consul.io/docs/agent/options.html#enable_acl_replication) and then set the token later using the [agent token API](https://www.consul.io/api/agent.html#update-acl-tokens) on each server. If the acl\_replication\_token is set in the config, it will automatically set[enable\_acl\_replication](https://www.consul.io/docs/agent/options.html#enable_acl_replication) to true for backward compatibility.

If there's a partition or other outage affecting the authoritative datacenter, and the [acl\_down\_policy](https://www.consul.io/docs/agent/options.html#acl_down_policy) is set to "extend-cache", tokens not in the cache can be resolved during the outage using the replicated set of ACLs. Please see the [ACL Guide](https://www.consul.io/docs/guides/acl.html#replication) replication section for more details.

* [acl\_token](https://www.consul.io/docs/agent/options.html" \l "acl_token_legacy) - **Deprecated in Consul 1.4.0. See the**[**acl.tokens.default**](https://www.consul.io/docs/agent/options.html#acl_tokens_default)**field instead.** When provided, the agent will use this token when making requests to the Consul servers. Clients can override this token on a per-request basis by providing the "?token" query parameter. When not provided, the empty token, which maps to the 'anonymous' ACL policy, is used.

* [acl\_ttl](https://www.consul.io/docs/agent/options.html" \l "acl_ttl_legacy) - **Deprecated in Consul 1.4.0. See the**[**acl.token\_ttl**](https://www.consul.io/docs/agent/options.html#acl_token_ttl)**field instead.**Used to control Time-To-Live caching of ACLs. By default, this is 30 seconds. This setting has a major performance impact: reducing it will cause more frequent refreshes while increasing it reduces the number of refreshes. However, because the caches are not actively invalidated, ACL policy may be stale up to the TTL value.

* [addresses](https://www.consul.io/docs/agent/options.html" \l "addresses) - This is a nested object that allows setting bind addresses. In Consul 1.0 and later these can be set to a space-separated list of addresses to bind to, or a [go-sockaddr](https://godoc.org/github.com/hashicorp/go-sockaddr/template) template that can potentially resolve to multiple addresses.

http, https and grpc all support binding to a Unix domain socket. A socket can be specified in the form unix:///path/to/socket. A new domain socket will be created at the given path. If the specified file path already exists, Consul will attempt to clear the file and create the domain socket in its place. The permissions of the socket file are tunable via the [unix\_sockets config construct](https://www.consul.io/docs/agent/options.html#unix_sockets).

When running Consul agent commands against Unix socket interfaces, use the -http-addr argument to specify the path to the socket. You can also place the desired values in the CONSUL\_HTTP\_ADDR environment variable.

For TCP addresses, the environment variable value should be an IP address with the port. For example: 10.0.0.1:8500 and not 10.0.0.1. However, ports are set separately in the [ports](https://www.consul.io/docs/agent/options.html#ports) structure when defining them in a configuration file.

The following keys are valid:

* + [dns](https://www.consul.io/docs/agent/options.html" \l "dns) - The DNS server. Defaults to client\_addr

* + [http](https://www.consul.io/docs/agent/options.html" \l "http) - The HTTP API. Defaults to client\_addr

* + [https](https://www.consul.io/docs/agent/options.html" \l "https) - The HTTPS API. Defaults to client\_addr

* + [grpc](https://www.consul.io/docs/agent/options.html" \l "grpc) - The gRPC API. Defaults to client\_addr

* [advertise\_addr](https://www.consul.io/docs/agent/options.html" \l "advertise_addr) Equivalent to the [-advertise command-line flag](https://www.consul.io/docs/agent/options.html#_advertise).

* [serf\_wan](https://www.consul.io/docs/agent/options.html" \l "serf_wan_bind) Equivalent to the [-serf-wan-bind command-line flag](https://www.consul.io/docs/agent/options.html#_serf_wan_bind).

* [serf\_lan](https://www.consul.io/docs/agent/options.html" \l "serf_lan_bind) Equivalent to the [-serf-lan-bind command-line flag](https://www.consul.io/docs/agent/options.html#_serf_lan_bind).

* [advertise\_addr\_wan](https://www.consul.io/docs/agent/options.html" \l "advertise_addr_wan) Equivalent to the [-advertise-wan command-line flag](https://www.consul.io/docs/agent/options.html#_advertise-wan).

* [autopilot](https://www.consul.io/docs/agent/options.html" \l "autopilot) Added in Consul 0.8, this object allows a number of sub-keys to be set which can configure operator-friendly settings for Consul servers. For more information about Autopilot, see the [Autopilot Guide](https://www.consul.io/docs/guides/autopilot.html).

The following sub-keys are available:

* + [cleanup\_dead\_servers](https://www.consul.io/docs/agent/options.html" \l "cleanup_dead_servers) - This controls the automatic removal of dead server nodes periodically and whenever a new server is added to the cluster. Defaults to true.

* + [last\_contact\_threshold](https://www.consul.io/docs/agent/options.html" \l "last_contact_threshold) - Controls the maximum amount of time a server can go without contact from the leader before being considered unhealthy. Must be a duration value such as 10s. Defaults to 200ms.

* + [max\_trailing\_logs](https://www.consul.io/docs/agent/options.html" \l "max_trailing_logs) - Controls the maximum number of log entries that a server can trail the leader by before being considered unhealthy. Defaults to 250.

* + [server\_stabilization\_time](https://www.consul.io/docs/agent/options.html" \l "server_stabilization_time) - Controls the minimum amount of time a server must be stable in the 'healthy' state before being added to the cluster. Only takes effect if all servers are running Raft protocol version 3 or higher. Must be a duration value such as 30s. Defaults to 10s.

* + [redundancy\_zone\_tag](https://www.consul.io/docs/agent/options.html" \l "redundancy_zone_tag) - (Enterprise-only) This controls the [-node-meta](https://www.consul.io/docs/agent/options.html#_node_meta) key to use when Autopilot is separating servers into zones for redundancy. Only one server in each zone can be a voting member at one time. If left blank (the default), this feature will be disabled.

* + [disable\_upgrade\_migration](https://www.consul.io/docs/agent/options.html" \l "disable_upgrade_migration) - (Enterprise-only) If set to true, this setting will disable Autopilot's upgrade migration strategy in Consul Enterprise of waiting until enough newer-versioned servers have been added to the cluster before promoting any of them to voters. Defaults to false.
* [bootstrap](https://www.consul.io/docs/agent/options.html#bootstrap) Equivalent to the [-bootstrap command-line flag](https://www.consul.io/docs/agent/options.html#_bootstrap).

* [bootstrap\_expect](https://www.consul.io/docs/agent/options.html" \l "bootstrap_expect) Equivalent to the [-bootstrap-expect command-line flag](https://www.consul.io/docs/agent/options.html#_bootstrap_expect).

* [bind\_addr](https://www.consul.io/docs/agent/options.html" \l "bind_addr) Equivalent to the [-bind command-line flag](https://www.consul.io/docs/agent/options.html#_bind).

* [ca\_file](https://www.consul.io/docs/agent/options.html" \l "ca_file) This provides a file path to a PEM-encoded certificate authority. The certificate authority is used to check the authenticity of client and server connections with the appropriate [verify\_incoming](https://www.consul.io/docs/agent/options.html#verify_incoming) or [verify\_outgoing](https://www.consul.io/docs/agent/options.html#verify_outgoing) flags.

* [ca\_path](https://www.consul.io/docs/agent/options.html" \l "ca_path) This provides a path to a directory of PEM-encoded certificate authority files. These certificate authorities are used to check the authenticity of client and server connections with the appropriate [verify\_incoming](https://www.consul.io/docs/agent/options.html#verify_incoming) or[verify\_outgoing](https://www.consul.io/docs/agent/options.html#verify_outgoing) flags.

* [cert\_file](https://www.consul.io/docs/agent/options.html" \l "cert_file) This provides a file path to a PEM-encoded certificate. The certificate is provided to clients or servers to verify the agent's authenticity. It must be provided along with [key\_file](https://www.consul.io/docs/agent/options.html#key_file).

* [check\_update\_interval](https://www.consul.io/docs/agent/options.html" \l "check_update_interval) This interval controls how often check output from checks in a steady state is synchronized with the server. By default, this is set to 5 minutes ("5m"). Many checks which are in a steady state produce slightly different output per run (timestamps, etc) which cause constant writes. This configuration allows deferring the sync of check output for a given interval to reduce write pressure. If a check ever changes state, the new state and associated output is synchronized immediately. To disable this behavior, set the value to "0s".

* [client\_addr](https://www.consul.io/docs/agent/options.html" \l "client_addr) Equivalent to the [-client command-line flag](https://www.consul.io/docs/agent/options.html#_client).

* [connect](https://www.consul.io/docs/agent/options.html" \l "connect) This object allows setting options for the Connect feature.

The following sub-keys are available:

* + [enabled](https://www.consul.io/docs/agent/options.html" \l "connect_enabled) Controls whether Connect features are enabled on this agent. Should be enabled on all clients and servers in the cluster in order for Connect to function properly. Defaults to false.

* + [ca\_provider](https://www.consul.io/docs/agent/options.html" \l "connect_ca_provider) Controls which CA provider to use for Connect's CA. Currently only the consul and vaultproviders are supported. This is only used when initially bootstrapping the cluster. For an existing cluster, use the [Update CA Configuration Endpoint](https://www.consul.io/api/connect/ca.html#update-ca-configuration).

* + [ca\_config](https://www.consul.io/docs/agent/options.html" \l "connect_ca_config) An object which allows setting different config options based on the CA provider chosen. This is only used when initially bootstrapping the cluster. For an existing cluster, use the [Update CA Configuration Endpoint](https://www.consul.io/api/connect/ca.html#update-ca-configuration).

The following providers are supported:

#### [»](https://www.consul.io/docs/agent/options.html" \l "consul-ca-provider-ca_provider-quot-consul-quot-)Consul CA Provider (ca\_provider = "consul")

* + - [private\_key](https://www.consul.io/docs/agent/options.html" \l "consul_ca_private_key) The PEM contents of the private key to use for the CA.

* + - [root\_cert](https://www.consul.io/docs/agent/options.html" \l "consul_ca_root_cert) The PEM contents of the root certificate to use for the CA.

#### [»](https://www.consul.io/docs/agent/options.html" \l "vault-ca-provider-ca_provider-quot-vault-quot-)Vault CA Provider (ca\_provider = "vault")

* + - [address](https://www.consul.io/docs/agent/options.html" \l "vault_ca_address) The address of the Vault server to connect to.

* + - [token](https://www.consul.io/docs/agent/options.html" \l "vault_ca_token) The Vault token to use.

* + - [root\_pki\_path](https://www.consul.io/docs/agent/options.html" \l "vault_ca_root_pki) The path to use for the root CA pki backend in Vault. This can be an existing backend with a CA already configured, or a blank/unmounted backend in which case Connect will automatically mount/generate the CA. The Vault token given above must have sudo access to this backend, as well as permission to mount the backend at this path if it is not already mounted.

* + - [intermediate\_pki\_path](https://www.consul.io/docs/agent/options.html" \l "vault_ca_intermediate_pki) The path to use for the temporary intermediate CA pki backend in Vault. Connect will overwrite any data at this path in order to generate a temporary intermediate CA. The Vault token given above must have write access to this backend, as well as permission to mount the backend at this path if it is not already mounted.

#### [»](https://www.consul.io/docs/agent/options.html" \l "common-ca-config-options)Common CA Config Options

There are also a number of common configuration options supported by all providers:

* + - [leaf\_cert\_ttl](https://www.consul.io/docs/agent/options.html" \l "ca_leaf_cert_ttl) The upper bound on the lease duration of a leaf certificate issued for a service. In most cases a new leaf certificate will be requested by a proxy before this limit is reached. This is also the effective limit on how long a server outage can last (with no leader) before network connections will start being rejected, and as a result the defaults is 72h to last through a weekend without intervention. This value cannot be lower than 1 hour or higher than 1 year.

This value is also used when rotating out old root certificates from the cluster. When a root certificate has been inactive (rotated out) for more than twice the current leaf\_cert\_ttl, it will be removed from the trusted list.

* + - [csr\_max\_per\_second](https://www.consul.io/docs/agent/options.html" \l "ca_csr_max_per_second) Sets a rate limit on the maximum number of Certificate Signing Requests (CSRs) the servers will accept. This is used to prevent CA rotation from causing unbounded CPU usage on servers. It defaults to 50 which is conservative - a 2017 Macbook can process about 100 per second using only ~40% of one CPU core - but sufficient for deployments up to ~1500 service instances before the time it takes to rotate is impacted. For larger deployments we recommend increasing this based on the expected number of server instances and server resources, or use csr\_max\_concurrent instead if servers have more than one core. Setting this to zero disables rate limiting. Added in 1.4.1.

* + - [csr\_max\_concurrent](https://www.consul.io/docs/agent/options.html" \l "ca_csr_max_concurrent) Sets a limit on how many Certificate Signing Requests will be processed concurrently. Defaults to 0 (disabled). This is useful when you have more than one or two cores available to the server. For example on an 8 core server, setting this to 1 will ensure that even during a CA rotation no more than one server core on the leader will be consumed at a time with generating new certificates. Setting this is recommended instead of csr\_max\_per\_second where you know there are multiple cores available since it is simpler to reason about limiting CSR resources this way without artificially slowing down rotations. Added in 1.4.1.

* + - [proxy](https://www.consul.io/docs/agent/options.html" \l "connect_proxy) [**Deprecated**](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) This object allows setting options for the Connect proxies. The following sub-keys are available:

* + - * [allow\_managed\_api\_registration](https://www.consul.io/docs/agent/options.html" \l "connect_proxy_allow_managed_registration) [**Deprecated**](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) Allows managed proxies to be configured with services that are registered via the Agent HTTP API. Enabling this would allow anyone with permission to register a service to define a command to execute for the proxy. By default, this is false to protect against arbitrary process execution.

* + - * [allow\_managed\_root](https://www.consul.io/docs/agent/options.html" \l "connect_proxy_allow_managed_root) [**Deprecated**](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) Allows Consul to start managed proxies if Consul is running as root (EUID of the process is zero). We recommend running Consul as a non-root user. By default, this is false to protect inadvertently running external processes as root.

* + - [proxy\_defaults](https://www.consul.io/docs/agent/options.html" \l "connect_proxy_defaults) [**Deprecated**](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) This object configures the default proxy settings for service definitions with [managed proxies](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) (now deprecated). It accepts the fields exec\_mode, daemon\_command, and config. These are used as default values for the respective fields in the service definition.

* [datacenter](https://www.consul.io/docs/agent/options.html" \l "datacenter) Equivalent to the [-datacenter command-line flag](https://www.consul.io/docs/agent/options.html#_datacenter).

* [data\_dir](https://www.consul.io/docs/agent/options.html" \l "data_dir) Equivalent to the [-data-dir command-line flag](https://www.consul.io/docs/agent/options.html#_data_dir).

* [disable\_anonymous\_signature](https://www.consul.io/docs/agent/options.html" \l "disable_anonymous_signature) Disables providing an anonymous signature for de-duplication with the update check. See [disable\_update\_check](https://www.consul.io/docs/agent/options.html#disable_update_check).

* [disable\_host\_node\_id](https://www.consul.io/docs/agent/options.html" \l "disable_host_node_id) Equivalent to the [-disable-host-node-id command-line flag](https://www.consul.io/docs/agent/options.html#_disable_host_node_id).

* [disable\_http\_unprintable\_char\_filter](https://www.consul.io/docs/agent/options.html" \l "disable_http_unprintable_char_filter) Defaults to false. Consul 1.0.3 fixed a potential security vulnerability where malicious users could craft KV keys with unprintable chars that would confuse operators using the CLI or UI into taking wrong actions. Users who had data written in older versions of Consul that did not have this restriction will be unable to delete those values by default in 1.0.3 or later. This setting enables those users to temporarilydisable the filter such that delete operations can work on those keys again to get back to a healthy state. It is strongly recommended that this filter is not disabled permanently as it exposes the original security vulnerability.

* [disable\_remote\_exec](https://www.consul.io/docs/agent/options.html" \l "disable_remote_exec) Disables support for remote execution. When set to true, the agent will ignore any incoming remote exec requests. In versions of Consul prior to 0.8, this defaulted to false. In Consul 0.8 the default was changed to true, to make remote exec opt-in instead of opt-out.

* [disable\_update\_check](https://www.consul.io/docs/agent/options.html" \l "disable_update_check) Disables automatic checking for security bulletins and new version releases. This is disabled in Consul Enterprise.

* [discard\_check\_output](https://www.consul.io/docs/agent/options.html" \l "discard_check_output) Discards the output of health checks before storing them. This reduces the number of writes to the Consul raft log in environments where health checks have volatile output like timestamps, process ids, ...

* [discovery\_max\_stale](https://www.consul.io/docs/agent/options.html" \l "discovery_max_stale) - Enables stale requests for all service discovery HTTP endpoints. This is equivalent to the[max\_stale](https://www.consul.io/docs/agent/options.html#max_stale) configuration for DNS requests. If this value is zero (default), all service discovery HTTP endpoints are forwarded to the leader. If this value is greater than zero, any Consul server can handle the service discovery request. If a Consul server is behind the leader by more than discovery\_max\_stale, the query will be re-evaluated on the leader to get more up-to-date results. Consul agents also add a new X-Consul-Effective-Consistencyresponse header which indicates if the agent did a stale read. discover-max-stale was introduced in Consul 1.0.7 as a way for Consul operators to force stale requests from clients at the agent level, and defaults to zero which matches default consistency behavior in earlier Consul versions.

* [dns\_config](https://www.consul.io/docs/agent/options.html" \l "dns_config) This object allows a number of sub-keys to be set which can tune how DNS queries are serviced. See this guide on [DNS caching](https://www.consul.io/docs/guides/dns-cache.html) for more detail.

The following sub-keys are available:

* + [allow\_stale](https://www.consul.io/docs/agent/options.html" \l "allow_stale) - Enables a stale query for DNS information. This allows any Consul server, rather than only the leader, to service the request. The advantage of this is you get linear read scalability with Consul servers. In versions of Consul prior to 0.7, this defaulted to false, meaning all requests are serviced by the leader, providing stronger consistency but less throughput and higher latency. In Consul 0.7 and later, this defaults to true for better utilization of available servers.

* + [max\_stale](https://www.consul.io/docs/agent/options.html" \l "max_stale) - When [allow\_stale](https://www.consul.io/docs/agent/options.html#allow_stale) is specified, this is used to limit how stale results are allowed to be. If a Consul server is behind the leader by more than max\_stale, the query will be re-evaluated on the leader to get more up-to-date results. Prior to Consul 0.7.1 this defaulted to 5 seconds; in Consul 0.7.1 and later this defaults to 10 years ("87600h") which effectively allows DNS queries to be answered by any server, no matter how stale. In practice, servers are usually only milliseconds behind the leader, so this lets Consul continue serving requests in long outage scenarios where no leader can be elected.

* + [node\_ttl](https://www.consul.io/docs/agent/options.html" \l "node_ttl) - By default, this is "0s", so all node lookups are served with a 0 TTL value. DNS caching for node lookups can be enabled by setting this value. This should be specified with the "s" suffix for second or "m" for minute.

* + [service\_ttl](https://www.consul.io/docs/agent/options.html" \l "service_ttl) - This is a sub-object which allows for setting a TTL on service lookups with a per-service policy. The "\*" wildcard service can be used when there is no specific policy available for a service. By default, all services are served with a 0 TTL value. DNS caching for service lookups can be enabled by setting this value.

* + [enable\_truncate](https://www.consul.io/docs/agent/options.html" \l "enable_truncate) - If set to true, a UDP DNS query that would return more than 3 records, or more than would fit into a valid UDP response, will set the truncated flag, indicating to clients that they should re-query using TCP to get the full set of records.

* + [only\_passing](https://www.consul.io/docs/agent/options.html" \l "only_passing) - If set to true, any nodes whose health checks are warning or critical will be excluded from DNS results. If false, the default, only nodes whose healthchecks are failing as critical will be excluded. For service lookups, the health checks of the node itself, as well as the service-specific checks are considered. For example, if a node has a health check that is critical then all services on that node will be excluded because they are also considered critical.

* + [recursor\_timeout](https://www.consul.io/docs/agent/options.html" \l "recursor_timeout) - Timeout used by Consul when recursively querying an upstream DNS server. See [recursors](https://www.consul.io/docs/agent/options.html#recursors) for more details. Default is 2s. This is available in Consul 0.7 and later.

* + [disable\_compression](https://www.consul.io/docs/agent/options.html" \l "disable_compression) - If set to true, DNS responses will not be compressed. Compression was added and enabled by default in Consul 0.7.

* + [udp\_answer\_limit](https://www.consul.io/docs/agent/options.html" \l "udp_answer_limit) - Limit the number of resource records contained in the answer section of a UDP-based DNS response. This parameter applies only to UDP DNS queries that are less than 512 bytes. This setting is deprecated and replaced in Consul 1.0.7 by [a\_record\_limit](https://www.consul.io/docs/agent/options.html#a_record_limit).

* + [a\_record\_limit](https://www.consul.io/docs/agent/options.html" \l "a_record_limit) - Limit the number of resource records contained in the answer section of a A, AAAA or ANY DNS response (both TCP and UDP). When answering a question, Consul will use the complete list of matching hosts, shuffle the list randomly, and then limit the number of answers to a\_record\_limit (default: no limit). This limit does not apply to SRV records.

In environments where [RFC 3484 Section 6](https://tools.ietf.org/html/rfc3484#section-6) Rule 9 is implemented and enforced (i.e. DNS answers are always sorted and therefore never random), clients may need to set this value to 1 to preserve the expected randomized distribution behavior (note: [RFC 3484](https://tools.ietf.org/html/rfc3484) has been obsoleted by [RFC 6724](https://tools.ietf.org/html/rfc6724) and as a result it should be increasingly uncommon to need to change this value with modern resolvers).

* + [enable\_additional\_node\_meta\_txt](https://www.consul.io/docs/agent/options.html" \l "enable_additional_node_meta_txt) - When set to true, Consul will add TXT records for Node metadata into the Additional section of the DNS responses for several query types such as SRV queries. When set to false those records are not emitted. This does not impact the behavior of those same TXT records when they would be added to the Answer section of the response like when querying with type TXT or ANY. This defaults to true.

* + [soa](https://www.consul.io/docs/agent/options.html" \l "soa) Allow to tune the setting set up in SOA. Non specified values fallback to their default values, all values are integers and expressed as seconds.   
      
    The following settings are available:

* + - [expire](https://www.consul.io/docs/agent/options.html" \l "soa_expire) - Configure SOA Expire duration in seconds, default value is 86400, ie: 24 hours.

* + - [min\_ttl](https://www.consul.io/docs/agent/options.html" \l "soa_min_ttl) - Configure SOA DNS minimum TTL. As explained in [RFC-2308](https://tools.ietf.org/html/rfc2308) this also controls negative cache TTL in most implementations. Default value is 0, ie: no minimum delay or negative TTL.

* + - [refresh](https://www.consul.io/docs/agent/options.html" \l "soa_refresh) - Configure SOA Refresh duration in seconds, default value is 3600, ie: 1 hour.

* + - [retry](https://www.consul.io/docs/agent/options.html" \l "soa_retry) - Configures the Retry duration expressed in seconds, default value is 600, ie: 10 minutes.

* + [use\_cache](https://www.consul.io/docs/agent/options.html" \l "dns_use_cache) - When set to true, DNS resolution will use the agent cache described in [agent caching](https://www.consul.io/api/index.html#agent-caching). This setting affects all service and prepared queries DNS requests. Implies [allow\_stale](https://www.consul.io/docs/agent/options.html#allow_stale)

* + [cache\_max\_age](https://www.consul.io/docs/agent/options.html" \l "dns_cache_max_age) - When [use\_cache](https://www.consul.io/docs/agent/options.html#dns_use_cache) is enabled, the agent will attempt to re-fetch the result from the servers if the cached value is older than this duration. See: [agent caching](https://www.consul.io/api/index.html#agent-caching).

* [domain](https://www.consul.io/docs/agent/options.html" \l "domain) Equivalent to the [-domain command-line flag](https://www.consul.io/docs/agent/options.html#_domain).

* [enable\_acl\_replication](https://www.consul.io/docs/agent/options.html" \l "enable_acl_replication) When set on a Consul server, enables [ACL replication](https://www.consul.io/docs/guides/acl.html#replication) without having to set the replication token via [acl\_replication\_token](https://www.consul.io/docs/agent/options.html#acl_replication_token). Instead, enable ACL replication and then introduce the token using the [agent token API](https://www.consul.io/api/agent.html#update-acl-tokens) on each server. See [acl\_replication\_token](https://www.consul.io/docs/agent/options.html#acl_replication_token) for more details.

* [enable\_agent\_tls\_for\_checks](https://www.consul.io/docs/agent/options.html" \l "enable_agent_tls_for_checks) When set, uses a subset of the agent's TLS configuration (key\_file, cert\_file, ca\_file, ca\_path, and server\_name) to set up the client for HTTP or gRPC health checks. This allows services requiring 2-way TLS to be checked using the agent's credentials. This was added in Consul 1.0.1 and defaults to false.

* [enable\_debug](https://www.consul.io/docs/agent/options.html" \l "enable_debug) When set, enables some additional debugging features. Currently, this is only used to access runtime profiling HTTP endpoints, which are available with an operator:read ACL regardles of the value of enable\_debug.

* [enable\_script\_checks](https://www.consul.io/docs/agent/options.html" \l "enable_script_checks) Equivalent to the [-enable-script-checks command-line flag](https://www.consul.io/docs/agent/options.html#_enable_script_checks).

**Security Warning:** Enabling script checks in some configurations may introduce a remote execution vulnerability which is known to be targeted by malware. We strongly recommend enable\_local\_script\_checks instead. See [this blog post](https://www.hashicorp.com/blog/protecting-consul-from-rce-risk-in-specific-configurations) for more details.

* [enable\_local\_script\_checks](https://www.consul.io/docs/agent/options.html" \l "enable_local_script_checks) Equivalent to the [-enable-local-script-checks command-line flag](https://www.consul.io/docs/agent/options.html#_enable_local_script_checks).

* [enable\_syslog](https://www.consul.io/docs/agent/options.html" \l "enable_syslog) Equivalent to the [-syslog command-line flag](https://www.consul.io/docs/agent/options.html#_syslog).

* [encrypt](https://www.consul.io/docs/agent/options.html" \l "encrypt) Equivalent to the [-encrypt command-line flag](https://www.consul.io/docs/agent/options.html#_encrypt).

* [encrypt\_verify\_incoming](https://www.consul.io/docs/agent/options.html" \l "encrypt_verify_incoming) - This is an optional parameter that can be used to disable enforcing encryption for incoming gossip in order to upshift from unencrypted to encrypted gossip on a running cluster. See [this section](https://www.consul.io/docs/agent/encryption.html#configuring-gossip-encryption-on-an-existing-cluster) for more information. Defaults to true.

* [encrypt\_verify\_outgoing](https://www.consul.io/docs/agent/options.html" \l "encrypt_verify_outgoing) - This is an optional parameter that can be used to disable enforcing encryption for outgoing gossip in order to upshift from unencrypted to encrypted gossip on a running cluster. See [this section](https://www.consul.io/docs/agent/encryption.html#configuring-gossip-encryption-on-an-existing-cluster) for more information. Defaults to true.

* [disable\_keyring\_file](https://www.consul.io/docs/agent/options.html" \l "disable_keyring_file) - Equivalent to the [-disable-keyring-file command-line flag](https://www.consul.io/docs/agent/options.html#_disable_keyring_file).

* [gossip\_lan](https://www.consul.io/docs/agent/options.html" \l "gossip_lan) - **(Advanced)** This object contains a number of sub-keys which can be set to tune the LAN gossip communications. These are only provided for users running especially large clusters that need fine tuning and are prepared to spend significant effort correctly tuning them for their environment and workload. **Tuning these improperly can cause Consul to fail in unexpected ways**. The default values are appropriate in almost all deployments.

* + [gossip\_nodes](https://www.consul.io/docs/agent/options.html" \l "gossip_nodes) - The number of random nodes to send gossip messages to per gossip\_interval. Increasing this number causes the gossip messages to propagate across the cluster more quickly at the expense of increased bandwidth. The default is 3.

* + [gossip\_interval](https://www.consul.io/docs/agent/options.html" \l "gossip_interval) - The interval between sending messages that need to be gossiped that haven't been able to piggyback on probing messages. If this is set to zero, non-piggyback gossip is disabled. By lowering this value (more frequent) gossip messages are propagated across the cluster more quickly at the expense of increased bandwidth. The default is 200ms.

* + [probe\_interval](https://www.consul.io/docs/agent/options.html" \l "probe_interval) - The interval between random node probes. Setting this lower (more frequent) will cause the cluster to detect failed nodes more quickly at the expense of increased bandwidth usage. The default is 1s.

* + [probe\_timeout](https://www.consul.io/docs/agent/options.html" \l "probe_timeout) - The timeout to wait for an ack from a probed node before assuming it is unhealthy. This should be at least the 99-percentile of RTT (round-trip time) on your network. The default is 500ms and is a conservative value suitable for almost all realistic deployments.

* + [retransmit\_mult](https://www.consul.io/docs/agent/options.html" \l "retransmit_mult) - The multiplier for the number of retransmissions that are attempted for messages broadcasted over gossip. The number of retransmits is scaled using this multiplier and the cluster size. The higher the multiplier, the more likely a failed broadcast is to converge at the expense of increased bandwidth. The default is 4.

* + [suspicion\_mult](https://www.consul.io/docs/agent/options.html" \l "suspicion_mult) - The multiplier for determining the time an inaccessible node is considered suspect before declaring it dead. The timeout is scaled with the cluster size and the probe\_interval. This allows the timeout to scale properly with expected propagation delay with a larger cluster size. The higher the multiplier, the longer an inaccessible node is considered part of the cluster before declaring it dead, giving that suspect node more time to refute if it is indeed still alive. The default is 4.

* [gossip\_wan](https://www.consul.io/docs/agent/options.html" \l "gossip_wan) - **(Advanced)** This object contains a number of sub-keys which can be set to tune the WAN gossip communications. These are only provided for users running especially large clusters that need fine tuning and are prepared to spend significant effort correctly tuning them for their environment and workload. **Tuning these improperly can cause Consul to fail in unexpected ways**. The default values are appropriate in almost all deployments.
  + [gossip\_nodes](https://www.consul.io/docs/agent/options.html" \l "gossip_nodes) - The number of random nodes to send gossip messages to per gossip\_interval. Increasing this number causes the gossip messages to propagate across the cluster more quickly at the expense of increased bandwidth. The default is 3.
  + [gossip\_interval](https://www.consul.io/docs/agent/options.html" \l "gossip_interval) - The interval between sending messages that need to be gossiped that haven't been able to piggyback on probing messages. If this is set to zero, non-piggyback gossip is disabled. By lowering this value (more frequent) gossip messages are propagated across the cluster more quickly at the expense of increased bandwidth. The default is 200ms.
  + [probe\_interval](https://www.consul.io/docs/agent/options.html" \l "probe_interval) - The interval between random node probes. Setting this lower (more frequent) will cause the cluster to detect failed nodes more quickly at the expense of increased bandwidth usage. The default is 1s.
  + [probe\_timeout](https://www.consul.io/docs/agent/options.html" \l "probe_timeout) - The timeout to wait for an ack from a probed node before assuming it is unhealthy. This should be at least the 99-percentile of RTT (round-trip time) on your network. The default is 500ms and is a conservative value suitable for almost all realistic deployments.
  + [retransmit\_mult](https://www.consul.io/docs/agent/options.html" \l "retransmit_mult) - The multiplier for the number of retransmissions that are attempted for messages broadcasted over gossip. The number of retransmits is scaled using this multiplier and the cluster size. The higher the multiplier, the more likely a failed broadcast is to converge at the expense of increased bandwidth. The default is 4.
  + [suspicion\_mult](https://www.consul.io/docs/agent/options.html" \l "suspicion_mult) - The multiplier for determining the time an inaccessible node is considered suspect before declaring it dead. The timeout is scaled with the cluster size and the probe\_interval. This allows the timeout to scale properly with expected propagation delay with a larger cluster size. The higher the multiplier, the longer an inaccessible node is considered part of the cluster before declaring it dead, giving that suspect node more time to refute if it is indeed still alive. The default is 4.

* [key\_file](https://www.consul.io/docs/agent/options.html" \l "key_file) This provides a the file path to a PEM-encoded private key. The key is used with the certificate to verify the agent's authenticity. This must be provided along with [cert\_file](https://www.consul.io/docs/agent/options.html#cert_file).

* [http\_config](https://www.consul.io/docs/agent/options.html" \l "http_config) This object allows setting options for the HTTP API.

The following sub-keys are available:

* + [block\_endpoints](https://www.consul.io/docs/agent/options.html" \l "block_endpoints) This object is a list of HTTP API endpoint prefixes to block on the agent, and defaults to an empty list, meaning all endpoints are enabled. Any endpoint that has a common prefix with one of the entries on this list will be blocked and will return a 403 response code when accessed. For example, to block all of the V1 ACL endpoints, set this to ["/v1/acl"], which will block /v1/acl/create, /v1/acl/update, and the other ACL endpoints that begin with /v1/acl. This only works with API endpoints, not /ui or /debug, those must be disabled with their respective configuration options. Any CLI commands that use disabled endpoints will no longer function as well. For more general access control, Consul's [ACL system](https://www.consul.io/docs/guides/acl.html) should be used, but this option is useful for removing access to HTTP API endpoints completely, or on specific agents. This is available in Consul 0.9.0 and later.

* + [response\_headers](https://www.consul.io/docs/agent/options.html" \l "response_headers) This object allows adding headers to the HTTP API responses. For example, the following config can be used to enable [CORS](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) on the HTTP API endpoints:
  + {
  + "http\_config": {
  + "response\_headers": {
  + "Access-Control-Allow-Origin": "\*"
  + }
  + }
  + }

* + [allow\_write\_http\_from](https://www.consul.io/docs/agent/options.html" \l "allow_write_http_from) This object is a list of networks in CIDR notation (eg "127.0.0.0/8") that are allowed to call the agent write endpoints. It defaults to an empty list, which means all networks are allowed. This is used to make the agent read-only, except for select ip ranges.
    - To block write calls from anywhere, use [ "255.255.255.255/32" ].
    - To only allow write calls from localhost, use [ "127.0.0.0/8" ]
    - To only allow specific IPs, use [ "10.0.0.1/32", "10.0.0.2/32" ]

* [leave\_on\_terminate](https://www.consul.io/docs/agent/options.html" \l "leave_on_terminate) If enabled, when the agent receives a TERM signal, it will send a Leave message to the rest of the cluster and gracefully leave. The default behavior for this feature varies based on whether or not the agent is running as a client or a server (prior to Consul 0.7 the default value was unconditionally set to false). On agents in client-mode, this defaults to true and for agents in server-mode, this defaults to false.

* [limits](https://www.consul.io/docs/agent/options.html" \l "limits) Available in Consul 0.9.3 and later, this is a nested object that configures limits that are enforced by the agent. Currently, this only applies to agents in client mode, not Consul servers. The following parameters are available:

* + [rpc\_rate](https://www.consul.io/docs/agent/options.html" \l "rpc_rate) - Configures the RPC rate limiter by setting the maximum request rate that this agent is allowed to make for RPC requests to Consul servers, in requests per second. Defaults to infinite, which disables rate limiting.
  + [rpc\_max\_burst](https://www.consul.io/docs/agent/options.html" \l "rpc_max_burst) - The size of the token bucket used to recharge the RPC rate limiter. Defaults to 1000 tokens, and each token is good for a single RPC call to a Consul server. See <https://en.wikipedia.org/wiki/Token_bucket> for more details about how token bucket rate limiters operate.

* [log\_file](https://www.consul.io/docs/agent/options.html" \l "log_file) Equivalent to the [-log-file command-line flag](https://www.consul.io/docs/agent/options.html#_log_file).

* [log\_level](https://www.consul.io/docs/agent/options.html" \l "log_level) Equivalent to the [-log-level command-line flag](https://www.consul.io/docs/agent/options.html#_log_level).

* [node\_id](https://www.consul.io/docs/agent/options.html" \l "node_id) Equivalent to the [-node-id command-line flag](https://www.consul.io/docs/agent/options.html#_node_id).

* [node\_name](https://www.consul.io/docs/agent/options.html" \l "node_name) Equivalent to the [-node command-line flag](https://www.consul.io/docs/agent/options.html#_node).

* [node\_meta](https://www.consul.io/docs/agent/options.html" \l "node_meta) Available in Consul 0.7.3 and later, This object allows associating arbitrary metadata key/value pairs with the local node, which can then be used for filtering results from certain catalog endpoints. See the [-node-metacommand-line flag](https://www.consul.io/docs/agent/options.html#_node_meta) for more information.
* {
* "node\_meta": {
* "instance\_type": "t2.medium"
* }
* }

* [performance](https://www.consul.io/docs/agent/options.html" \l "performance) Available in Consul 0.7 and later, this is a nested object that allows tuning the performance of different subsystems in Consul. See the [Server Performance](https://www.consul.io/docs/guides/performance.html) guide for more details. The following parameters are available:

* + [leave\_drain\_time](https://www.consul.io/docs/agent/options.html" \l "leave_drain_time) - A duration that a server will dwell during a graceful leave in order to allow requests to be retried against other Consul servers. Under normal circumstances, this can prevent clients from experiencing "no leader" errors when performing a rolling update of the Consul servers. This was added in Consul 1.0. Must be a duration value such as 10s. Defaults to 5s.

* + [raft\_multiplier](https://www.consul.io/docs/agent/options.html" \l "raft_multiplier) - An integer multiplier used by Consul servers to scale key Raft timing parameters. Omitting this value or setting it to 0 uses default timing described below. Lower values are used to tighten timing and increase sensitivity while higher values relax timings and reduce sensitivity. Tuning this affects the time it takes Consul to detect leader failures and to perform leader elections, at the expense of requiring more network and CPU resources for better performance.

By default, Consul will use a lower-performance timing that's suitable for [minimal Consul servers](https://www.consul.io/docs/guides/performance.html#minimum), currently equivalent to setting this to a value of 5 (this default may be changed in future versions of Consul, depending if the target minimum server profile changes). Setting this to a value of 1 will configure Raft to its highest-performance mode, equivalent to the default timing of Consul prior to 0.7, and is recommended for [production Consul servers](https://www.consul.io/docs/guides/performance.html#production). See the note on [last contact](https://www.consul.io/docs/guides/performance.html#last-contact) timing for more details on tuning this parameter. The maximum allowed value is 10.

* + [rpc\_hold\_timeout](https://www.consul.io/docs/agent/options.html" \l "rpc_hold_timeout) - A duration that a client or server will retry internal RPC requests during leader elections. Under normal circumstances, this can prevent clients from experiencing "no leader" errors. This was added in Consul 1.0. Must be a duration value such as 10s. Defaults to 7s.

* [ports](https://www.consul.io/docs/agent/options.html" \l "ports) This is a nested object that allows setting the bind ports for the following keys:

* + [dns](https://www.consul.io/docs/agent/options.html" \l "dns_port) - The DNS server, -1 to disable. Default 8600.

* + [http](https://www.consul.io/docs/agent/options.html" \l "http_port) - The HTTP API, -1 to disable. Default 8500.

* + [https](https://www.consul.io/docs/agent/options.html" \l "https_port) - The HTTPS API, -1 to disable. Default -1 (disabled). **We recommend using 8501** for https by convention as some tooling will work automatically with this.

* + [grpc](https://www.consul.io/docs/agent/options.html" \l "grpc_port) - The gRPC API, -1 to disable. Default -1 (disabled). **We recommend using 8502** for grpc by convention as some tooling will work automatically with this. This is set to 8502 by default when the agent runs in -devmode. Currently gRPC is only used to expose Envoy xDS API to Envoy proxies.

* + [serf\_lan](https://www.consul.io/docs/agent/options.html" \l "serf_lan_port) - The Serf LAN port. Default 8301.

* + [serf\_wan](https://www.consul.io/docs/agent/options.html" \l "serf_wan_port) - The Serf WAN port. Default 8302. Set to -1 to disable. **Note**: this will disable WAN federation which is not recommended. Various catalog and WAN related endpoints will return errors or empty results.

* + [server](https://www.consul.io/docs/agent/options.html" \l "server_rpc_port) - Server RPC address. Default 8300.

* + [proxy\_min\_port](https://www.consul.io/docs/agent/options.html" \l "proxy_min_port) [**Deprecated**](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) - Minimum port number to use for automatically assigned [managed proxies](https://www.consul.io/docs/connect/proxies/managed-deprecated.html). Default 20000.

* + [proxy\_max\_port](https://www.consul.io/docs/agent/options.html" \l "proxy_max_port) [**Deprecated**](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) - Maximum port number to use for automatically assigned [managed proxies](https://www.consul.io/docs/connect/proxies/managed-deprecated.html). Default 20255.

* + [sidecar\_min\_port](https://www.consul.io/docs/agent/options.html" \l "sidecar_min_port) - Inclusive minimum port number to use for automatically assigned [sidecar service registrations](https://www.consul.io/docs/connect/proxies/sidecar-service.html). Default 21000. Set to 0 to disable automatic port assignment.

* + [sidecar\_max\_port](https://www.consul.io/docs/agent/options.html" \l "sidecar_max_port) - Inclusive maximum port number to use for automatically assigned [sidecar service registrations](https://www.consul.io/docs/connect/proxies/sidecar-service.html). Default 21255. Set to 0 to disable automatic port assignment.

* [protocol](https://www.consul.io/docs/agent/options.html" \l "protocol) Equivalent to the [-protocol command-line flag](https://www.consul.io/docs/agent/options.html#_protocol).

* [primary\_datacenter](https://www.consul.io/docs/agent/options.html" \l "primary_datacenter) - This designates the datacenter which is authoritative for ACL information, intentions and is the root Certificate Authority for Connect. It must be provided to enable ACLs. All servers and datacenters must agree on the primary datacenter. Setting it on the servers is all you need for cluster-level enforcement, but for the APIs to forward properly from the clients, it must be set on them too. In Consul 0.8 and later, this also enables agent-level enforcement of ACLs. Please see the [ACL Guide](https://www.consul.io/docs/guides/acl.html) for more details.

* [raft\_protocol](https://www.consul.io/docs/agent/options.html" \l "raft_protocol) Equivalent to the [-raft-protocol command-line flag](https://www.consul.io/docs/agent/options.html#_raft_protocol).

* [raft\_snapshot\_threshold](https://www.consul.io/docs/agent/options.html" \l "raft_snapshot_threshold) Equivalent to the [-raft-snapshot-threshold command-line flag](https://www.consul.io/docs/agent/options.html#_raft_snapshot_threshold).

* [raft\_snapshot\_interval](https://www.consul.io/docs/agent/options.html" \l "raft_snapshot_interval) Equivalent to the [-raft-snapshot-interval command-line flag](https://www.consul.io/docs/agent/options.html#_raft_snapshot_interval).

* [reap](https://www.consul.io/docs/agent/options.html" \l "reap) This controls Consul's automatic reaping of child processes, which is useful if Consul is running as PID 1 in a Docker container. If this isn't specified, then Consul will automatically reap child processes if it detects it is running as PID 1. If this is set to true or false, then it controls reaping regardless of Consul's PID (forces reaping on or off, respectively). This option was removed in Consul 0.7.1. For later versions of Consul, you will need to reap processes using a wrapper, please see the [Consul Docker image entry point script](https://github.com/hashicorp/docker-consul/blob/master/0.X/docker-entrypoint.sh) for an example. If you are using Docker 1.13.0 or later, you can use the new --init option of the docker run command and docker will enable an init process with PID 1 that reaps child processes for the container. More info on [Docker docs](https://docs.docker.com/engine/reference/commandline/run/#options).

* [reconnect\_timeout](https://www.consul.io/docs/agent/options.html" \l "reconnect_timeout) This controls how long it takes for a failed node to be completely removed from the cluster. This defaults to 72 hours and it is recommended that this is set to at least double the maximum expected recoverable outage time for a node or network partition. WARNING: Setting this time too low could cause Consul servers to be removed from quorum during an extended node failure or partition, which could complicate recovery of the cluster. The value is a time with a unit suffix, which can be "s", "m", "h" for seconds, minutes, or hours. The value must be >= 8 hours.

* [reconnect\_timeout\_wan](https://www.consul.io/docs/agent/options.html" \l "reconnect_timeout_wan) This is the WAN equivalent of the [reconnect\_timeout](https://www.consul.io/docs/agent/options.html#reconnect_timeout) parameter, which controls how long it takes for a failed server to be completely removed from the WAN pool. This also defaults to 72 hours, and must be >= 8 hours.

* [recursors](https://www.consul.io/docs/agent/options.html" \l "recursors) This flag provides addresses of upstream DNS servers that are used to recursively resolve queries if they are not inside the service domain for Consul. For example, a node can use Consul directly as a DNS server, and if the record is outside of the "consul." domain, the query will be resolved upstream. As of Consul 1.0.1 recursors can be provided as IP addresses or as go-sockaddr templates. IP addresses are resolved in order, and duplicates are ignored.

* [rejoin\_after\_leave](https://www.consul.io/docs/agent/options.html" \l "rejoin_after_leave) Equivalent to the [-rejoin command-line flag](https://www.consul.io/docs/agent/options.html#_rejoin).

* [retry\_join](https://www.consul.io/docs/agent/options.html" \l "retry_join) - Equivalent to the [-retry-join](https://www.consul.io/docs/agent/options.html#retry-join) command-line flag.

* [retry\_interval](https://www.consul.io/docs/agent/options.html" \l "retry_interval) Equivalent to the [-retry-interval command-line flag](https://www.consul.io/docs/agent/options.html#_retry_interval).

* [retry\_join\_wan](https://www.consul.io/docs/agent/options.html" \l "retry_join_wan) Equivalent to the [-retry-join-wan command-line flag](https://www.consul.io/docs/agent/options.html#_retry_join_wan). Takes a list of addresses to attempt joining to WAN every [retry\_interval\_wan](https://www.consul.io/docs/agent/options.html#_retry_interval_wan) until at least one join works.

* [retry\_interval\_wan](https://www.consul.io/docs/agent/options.html" \l "retry_interval_wan) Equivalent to the [-retry-interval-wan command-line flag](https://www.consul.io/docs/agent/options.html#_retry_interval_wan).
* [segment](https://www.consul.io/docs/agent/options.html#segment) (Enterprise-only) Equivalent to the [-segment command-line flag](https://www.consul.io/docs/agent/options.html#_segment).

* [segments](https://www.consul.io/docs/agent/options.html" \l "segments) (Enterprise-only) This is a list of nested objects that allows setting the bind/advertise information for network segments. This can only be set on servers. See the [Network Segments Guide](https://www.consul.io/docs/guides/segments.html) for more details.

* + [name](https://www.consul.io/docs/agent/options.html" \l "segment_name) - The name of the segment. Must be a string between 1 and 64 characters in length.

* + [bind](https://www.consul.io/docs/agent/options.html" \l "segment_bind) - The bind address to use for the segment's gossip layer. Defaults to the [-bind](https://www.consul.io/docs/agent/options.html#_bind) value if not provided.

* + [port](https://www.consul.io/docs/agent/options.html" \l "segment_port) - The port to use for the segment's gossip layer (required).

* + [advertise](https://www.consul.io/docs/agent/options.html" \l "segment_advertise) - The advertise address to use for the segment's gossip layer. Defaults to the [-advertise](https://www.consul.io/docs/agent/options.html#_advertise) value if not provided.

* + [rpc\_listener](https://www.consul.io/docs/agent/options.html" \l "segment_rpc_listener) - If true, a separate RPC listener will be started on this segment's [-bind](https://www.consul.io/docs/agent/options.html#_bind) address on the rpc port. Only valid if the segment's bind address differs from the [-bind](https://www.consul.io/docs/agent/options.html#_bind) address. Defaults to false.
* [server](https://www.consul.io/docs/agent/options.html#server) Equivalent to the [-server command-line flag](https://www.consul.io/docs/agent/options.html#_server).

* [non\_voting\_server](https://www.consul.io/docs/agent/options.html" \l "non_voting_server) - Equivalent to the [-non-voting-server command-line flag](https://www.consul.io/docs/agent/options.html#_non_voting_server).

* [server\_name](https://www.consul.io/docs/agent/options.html" \l "server_name) When provided, this overrides the [node\_name](https://www.consul.io/docs/agent/options.html#_node) for the TLS certificate. It can be used to ensure that the certificate name matches the hostname we declare.

* [session\_ttl\_min](https://www.consul.io/docs/agent/options.html" \l "session_ttl_min) The minimum allowed session TTL. This ensures sessions are not created with TTL's shorter than the specified limit. It is recommended to keep this limit at or above the default to encourage clients to send infrequent heartbeats. Defaults to 10s.

* [skip\_leave\_on\_interrupt](https://www.consul.io/docs/agent/options.html" \l "skip_leave_on_interrupt) This is similar to [leave\_on\_terminate](https://www.consul.io/docs/agent/options.html#leave_on_terminate) but only affects interrupt handling. When Consul receives an interrupt signal (such as hitting Control-C in a terminal), Consul will gracefully leave the cluster. Setting this to true disables that behavior. The default behavior for this feature varies based on whether or not the agent is running as a client or a server (prior to Consul 0.7 the default value was unconditionally set to false). On agents in client-mode, this defaults to false and for agents in server-mode, this defaults to true (i.e. Ctrl-C on a server will keep the server in the cluster and therefore quorum, and Ctrl-C on a client will gracefully leave).

* [start\_join](https://www.consul.io/docs/agent/options.html" \l "start_join) An array of strings specifying addresses of nodes to [-join](https://www.consul.io/docs/agent/options.html#_join) upon startup. Note that using [retry\_join](https://www.consul.io/docs/agent/options.html#retry_join)could be more appropriate to help mitigate node startup race conditions when automating a Consul cluster deployment.

* [start\_join\_wan](https://www.consul.io/docs/agent/options.html" \l "start_join_wan) An array of strings specifying addresses of WAN nodes to [-join-wan](https://www.consul.io/docs/agent/options.html#_join_wan) upon startup.

* [telemetry](https://www.consul.io/docs/agent/options.html" \l "telemetry) This is a nested object that configures where Consul sends its runtime telemetry, and contains the following keys:

* + [circonus\_api\_token](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_api_token) A valid API Token used to create/manage check. If provided, metric management is enabled.

* + [circonus\_api\_app](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_api_app) A valid app name associated with the API token. By default, this is set to "consul".

* + [circonus\_api\_url](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_api_url) The base URL to use for contacting the Circonus API. By default, this is set to "<https://api.circonus.com/v2>".

* + [circonus\_submission\_interval](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_submission_interval) The interval at which metrics are submitted to Circonus. By default, this is set to "10s" (ten seconds).

* + [circonus\_submission\_url](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_submission_url) The check.config.submission\_url field, of a Check API object, from a previously created HTTPTRAP check.

* + [circonus\_check\_id](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_check_id) The Check ID (not **check bundle**) from a previously created HTTPTRAP check. The numeric portion of the check.\_cid field in the Check API object.

* + [circonus\_check\_force\_metric\_activation](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_check_force_metric_activation) Force activation of metrics which already exist and are not currently active. If check management is enabled, the default behavior is to add new metrics as they are encountered. If the metric already exists in the check, it will **not** be activated. This setting overrides that behavior. By default, this is set to false.

* + [circonus\_check\_instance\_id](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_check_instance_id) Uniquely identifies the metrics coming from this instance. It can be used to maintain metric continuity with transient or ephemeral instances as they move around within an infrastructure. By default, this is set to hostname:application name (e.g. "host123:consul").

* + [circonus\_check\_search\_tag](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_check_search_tag) A special tag which, when coupled with the instance id, helps to narrow down the search results when neither a Submission URL or Check ID is provided. By default, this is set to service:application name (e.g. "service:consul").

* + [circonus\_check\_display\_name](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_check_display_name) Specifies a name to give a check when it is created. This name is displayed in the Circonus UI Checks list. Available in Consul 0.7.2 and later.

* + [circonus\_check\_tags](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_check_tags) Comma separated list of additional tags to add to a check when it is created. Available in Consul 0.7.2 and later.

* + [circonus\_broker\_id](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_broker_id) The ID of a specific Circonus Broker to use when creating a new check. The numeric portion of broker.\_cid field in a Broker API object. If metric management is enabled and neither a Submission URL nor Check ID is provided, an attempt will be made to search for an existing check using Instance ID and Search Tag. If one is not found, a new HTTPTRAP check will be created. By default, this is not used and a random Enterprise Broker is selected, or the default Circonus Public Broker.

* + [circonus\_broker\_select\_tag](https://www.consul.io/docs/agent/options.html" \l "telemetry-circonus_broker_select_tag) A special tag which will be used to select a Circonus Broker when a Broker ID is not provided. The best use of this is to as a hint for which broker should be used based on where this particular instance is running (e.g. a specific geo location or datacenter, dc:sfo). By default, this is left blank and not used.

* + [disable\_hostname](https://www.consul.io/docs/agent/options.html" \l "telemetry-disable_hostname) This controls whether or not to prepend runtime telemetry with the machine's hostname, defaults to false.

* + [dogstatsd\_addr](https://www.consul.io/docs/agent/options.html" \l "telemetry-dogstatsd_addr) This provides the address of a DogStatsD instance in the format host:port. DogStatsD is a protocol-compatible flavor of statsd, with the added ability to decorate metrics with tags and event information. If provided, Consul will send various telemetry information to that instance for aggregation. This can be used to capture runtime information.

* + [dogstatsd\_tags](https://www.consul.io/docs/agent/options.html" \l "telemetry-dogstatsd_tags) This provides a list of global tags that will be added to all telemetry packets sent to DogStatsD. It is a list of strings, where each string looks like "my\_tag\_name:my\_tag\_value".

* + [filter\_default](https://www.consul.io/docs/agent/options.html" \l "telemetry-filter_default) This controls whether to allow metrics that have not been specified by the filter. Defaults to true, which will allow all metrics when no filters are provided. When set to false with no filters, no metrics will be sent.

* + [metrics\_prefix](https://www.consul.io/docs/agent/options.html" \l "telemetry-metrics_prefix) The prefix used while writing all telemetry data. By default, this is set to "consul". This was added in Consul 1.0. For previous versions of Consul, use the config option statsite\_prefix in this same structure. This was renamed in Consul 1.0 since this prefix applied to all telemetry providers, not just statsite.

* + [prefix\_filter](https://www.consul.io/docs/agent/options.html" \l "telemetry-prefix_filter) This is a list of filter rules to apply for allowing/blocking metrics by prefix in the following format:
  + [
  + "+consul.raft.apply",
  + "-consul.http",
  + "+consul.http.GET"
  + ]

A leading "**+**" will enable any metrics with the given prefix, and a leading "**-**" will block them. If there is overlap between two rules, the more specific rule will take precedence. Blocking will take priority if the same prefix is listed multiple times.

* + [prometheus\_retention\_time](https://www.consul.io/docs/agent/options.html" \l "telemetry-prometheus_retention_time) If the value is greater than 0s (the default), this enables [Prometheus](https://prometheus.io/) export of metrics. The duration can be expressed using the duration semantics and will aggregates all counters for the duration specified (it might have an impact on Consul's memory usage). A good value for this parameter is at least 2 times the interval of scrape of Prometheus, but you might also put a very high retention time such as a few days (for instance 744h to enable retention to 31 days). Fetching the metrics using prometheus can then be performed using the [/v1/agent/metrics?format=prometheus](https://www.consul.io/api/agent.html#view-metrics) endpoint. The format is compatible natively with prometheus. When running in this mode, it is recommended to also enable the option[disable\_hostname](https://www.consul.io/docs/agent/options.html#telemetry-disable_hostname) to avoid having prefixed metrics with hostname. Consul does not use the default Prometheus path, so Prometheus must be configured as follows. Note that using ?format=prometheus in the path won't work as ? will be escaped, so it must be specified as a parameter.
  + metrics\_path: "/v1/agent/metrics"
  + params:
  + format: ['prometheus']

* + [statsd\_address](https://www.consul.io/docs/agent/options.html" \l "telemetry-statsd_address) This provides the address of a statsd instance in the format host:port. If provided, Consul will send various telemetry information to that instance for aggregation. This can be used to capture runtime information. This sends UDP packets only and can be used with statsd or statsite.

* + [statsite\_address](https://www.consul.io/docs/agent/options.html" \l "telemetry-statsite_address) This provides the address of a statsite instance in the format host:port. If provided, Consul will stream various telemetry information to that instance for aggregation. This can be used to capture runtime information. This streams via TCP and can only be used with statsite.

* [syslog\_facility](https://www.consul.io/docs/agent/options.html" \l "syslog_facility) When [enable\_syslog](https://www.consul.io/docs/agent/options.html#enable_syslog) is provided, this controls to which facility messages are sent. By default, LOCAL0 will be used.

* [tls\_min\_version](https://www.consul.io/docs/agent/options.html" \l "tls_min_version) Added in Consul 0.7.4, this specifies the minimum supported version of TLS. Accepted values are "tls10", "tls11" or "tls12". This defaults to "tls12". WARNING: TLS 1.1 and lower are generally considered less secure; avoid using these if possible.

* [tls\_cipher\_suites](https://www.consul.io/docs/agent/options.html" \l "tls_cipher_suites) Added in Consul 0.8.2, this specifies the list of supported ciphersuites as a comma-separated-list. The list of all supported ciphersuites is available in the [source code](https://github.com/hashicorp/consul/blob/master/tlsutil/config.go#L363).

* [tls\_prefer\_server\_cipher\_suites](https://www.consul.io/docs/agent/options.html" \l "tls_prefer_server_cipher_suites) Added in Consul 0.8.2, this will cause Consul to prefer the server's ciphersuite over the client ciphersuites.

* [translate\_wan\_addrs](https://www.consul.io/docs/agent/options.html" \l "translate_wan_addrs) If set to true, Consul will prefer a node's configured [WAN address](https://www.consul.io/docs/agent/options.html#_advertise-wan) when servicing DNS and HTTP requests for a node in a remote datacenter. This allows the node to be reached within its own datacenter using its local address, and reached from other datacenters using its WAN address, which is useful in hybrid setups with mixed networks. This is disabled by default.

Starting in Consul 0.7 and later, node addresses in responses to HTTP requests will also prefer a node's configured [WAN address](https://www.consul.io/docs/agent/options.html#_advertise-wan) when querying for a node in a remote datacenter. An [X-Consul-Translate-Addresses](https://www.consul.io/api/index.html#translated-addresses) header will be present on all responses when translation is enabled to help clients know that the addresses may be translated. The TaggedAddresses field in responses also have a lan address for clients that need knowledge of that address, regardless of translation.

The following endpoints translate addresses:

* + [/v1/catalog/nodes](https://www.consul.io/api/catalog.html#catalog_nodes)
  + [/v1/catalog/node/<node>](https://www.consul.io/api/catalog.html#catalog_node)
  + [/v1/catalog/service/<service>](https://www.consul.io/api/catalog.html#catalog_service)
  + [/v1/health/service/<service>](https://www.consul.io/api/health.html#health_service)
  + [/v1/query/<query or name>/execute](https://www.consul.io/api/query.html#execute)

* [ui](https://www.consul.io/docs/agent/options.html" \l "ui) - Equivalent to the [-ui](https://www.consul.io/docs/agent/options.html#_ui) command-line flag.

* [ui\_dir](https://www.consul.io/docs/agent/options.html" \l "ui_dir) - Equivalent to the [-ui-dir](https://www.consul.io/docs/agent/options.html#_ui_dir) command-line flag. This configuration key is not required as of Consul version 0.7.0 and later. Specifying this configuration key will enable the web UI. There is no need to specify both ui-dir and ui. Specifying both will result in an error.

* [unix\_sockets](https://www.consul.io/docs/agent/options.html" \l "unix_sockets) - This allows tuning the ownership and permissions of the Unix domain socket files created by Consul. Domain sockets are only used if the HTTP address is configured with the unix:// prefix.

It is important to note that this option may have different effects on different operating systems. Linux generally observes socket file permissions while many BSD variants ignore permissions on the socket file itself. It is important to test this feature on your specific distribution. This feature is currently not functional on Windows hosts.

The following options are valid within this construct and apply globally to all sockets created by Consul:

* + [user](https://www.consul.io/docs/agent/options.html" \l "user) - The name or ID of the user who will own the socket file.

* + [group](https://www.consul.io/docs/agent/options.html" \l "group) - The group ID ownership of the socket file. This option currently only supports numeric IDs.

* + [mode](https://www.consul.io/docs/agent/options.html" \l "mode) - The permission bits to set on the file.

* [verify\_incoming](https://www.consul.io/docs/agent/options.html" \l "verify_incoming) - If set to true, Consul requires that all incoming connections make use of TLS and that the client provides a certificate signed by a Certificate Authority from the [ca\_file](https://www.consul.io/docs/agent/options.html#ca_file) or [ca\_path](https://www.consul.io/docs/agent/options.html#ca_path). This applies to both server RPC and to the HTTPS API. By default, this is false, and Consul will not enforce the use of TLS or verify a client's authenticity.

* [verify\_incoming\_rpc](https://www.consul.io/docs/agent/options.html" \l "verify_incoming_rpc) - If set to true, Consul requires that all incoming RPC connections make use of TLS and that the client provides a certificate signed by a Certificate Authority from the [ca\_file](https://www.consul.io/docs/agent/options.html#ca_file) or [ca\_path](https://www.consul.io/docs/agent/options.html#ca_path). By default, this is false, and Consul will not enforce the use of TLS or verify a client's authenticity.

* [verify\_incoming\_https](https://www.consul.io/docs/agent/options.html" \l "verify_incoming_https) - If set to true, Consul requires that all incoming HTTPS connections make use of TLS and that the client provides a certificate signed by a Certificate Authority from the [ca\_file](https://www.consul.io/docs/agent/options.html#ca_file) or [ca\_path](https://www.consul.io/docs/agent/options.html#ca_path). By default, this is false, and Consul will not enforce the use of TLS or verify a client's authenticity. To enable the HTTPS API, you must define an HTTPS port via the [ports](https://www.consul.io/docs/agent/options.html#ports) configuration. By default, HTTPS is disabled.

* [verify\_outgoing](https://www.consul.io/docs/agent/options.html" \l "verify_outgoing) - If set to true, Consul requires that all outgoing connections from this agent make use of TLS and that the server provides a certificate that is signed by a Certificate Authority from the [ca\_file](https://www.consul.io/docs/agent/options.html#ca_file) or [ca\_path](https://www.consul.io/docs/agent/options.html#ca_path). By default, this is false, and Consul will not make use of TLS for outgoing connections. This applies to clients and servers as both will make outgoing connections.

**Security Note:** Note that servers that specify verify\_outgoing = true will always talk to other servers over TLS, but they still accept non-TLS connections to allow for a transition of all clients to TLS. Currently the only way to enforce that no client can communicate with a server unencrypted is to also enable verify\_incomingwhich requires client certificates too.

* [verify\_server\_hostname](https://www.consul.io/docs/agent/options.html" \l "verify_server_hostname) - If set to true, Consul verifies for all outgoing TLS connections that the TLS certificate presented by the servers matches "server.<datacenter>.<domain>" hostname. By default, this is false, and Consul does not verify the hostname of the certificate, only that it is signed by a trusted CA. This setting is critical to prevent a compromised client from being restarted as a server and having all cluster state including all ACL tokens and Connect CA root keys replicated to it. This is new in 0.5.1.

**Security Note:** From versions 0.5.1 to 1.4.0, due to a bug, setting this flag alone does not imply verify\_outgoing and leaves client to server and server to server RPCs unencrypted despite the documentation stating otherwise. See [CVE-2018-19653](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-19653) for more details. For those versions you **must also set verify\_outgoing = true** to ensure encrypted RPC connections.

* [watches](https://www.consul.io/docs/agent/options.html" \l "watches) - Watches is a list of watch specifications which allow an external process to be automatically invoked when a particular data view is updated. See the [watch documentation](https://www.consul.io/docs/agent/watches.html) for more detail. Watches can be modified when the configuration is reloaded.

## [»](https://www.consul.io/docs/agent/options.html" \l "ports-used)Ports Used

Consul requires up to 6 different ports to work properly, some on TCP, UDP, or both protocols. Below we document the requirements for each port.

* Server RPC (Default 8300). This is used by servers to handle incoming requests from other agents. TCP only.
* Serf LAN (Default 8301). This is used to handle gossip in the LAN. Required by all agents. TCP and UDP.
* Serf WAN (Default 8302). This is used by servers to gossip over the WAN, to other servers. TCP and UDP. As of Consul 0.8 the WAN join flooding feature requires the Serf WAN port (TCP/UDP) to be listening on both WAN and LAN interfaces. See also: [Consul 0.8.0 CHANGELOG](https://github.com/hashicorp/consul/blob/master/CHANGELOG.md#080-april-5-2017) and [GH-3058](https://github.com/hashicorp/consul/issues/3058)
* HTTP API (Default 8500). This is used by clients to talk to the HTTP API. TCP only.
* DNS Interface (Default 8600). Used to resolve DNS queries. TCP and UDP.

## [»](https://www.consul.io/docs/agent/options.html" \l "reloadable-configuration)Reloadable Configuration

Reloading configuration does not reload all configuration items. The items which are reloaded include:

* Log level
* Checks
* Services
* Watches
* HTTP Client Address
* [Node Metadata](https://www.consul.io/docs/agent/options.html#node_meta)
* [Metric Prefix Filter](https://www.consul.io/docs/agent/options.html#telemetry-prefix_filter)
* [Discard Check Output](https://www.consul.io/docs/agent/options.html#discard_check_output)
* [RPC rate limiting](https://www.consul.io/docs/agent/options.html#limits)

# Cloud Auto-joining

As of Consul 0.9.1, retry-join accepts a unified interface using the [go-discover](https://github.com/hashicorp/go-discover) library for doing automatic cluster joining using cloud metadata. To use retry-join with a supported cloud provider, specify the configuration on the command line or configuration file as a key=value key=value ... string.

In Consul 0.9.1-0.9.3 the values need to be URL encoded but for most practical purposes you need to replace spaces with + signs.

As of Consul 1.0 the values are taken literally and must not be URL encoded. If the values contain spaces, equals, backslashes or double quotes then they need to be double quoted and the usual escaping rules apply.

$ consul agent -retry-join 'provider=my-cloud config=val config2="some other val" ...'

or via a configuration file:

{

"retry\_join": ["provider=my-cloud config=val config2=\"some other val\" ..."]

}

The cloud provider-specific configurations are detailed below. This can be combined with static IP or DNS addresses or even multiple configurations for different providers.

In order to use discovery behind a proxy, you will need to set HTTP\_PROXY, HTTPS\_PROXY and NO\_PROXY environment variables per [Golang net/http library](https://golang.org/pkg/net/http/#ProxyFromEnvironment).

The following sections give the options specific to each supported cloud provider.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "amazon-ec2)Amazon EC2

This returns the first private IP address of all servers in the given region which have the given tag\_key and tag\_value.

$ consul agent -retry-join "provider=aws tag\_key=... tag\_value=..."

{

"retry\_join": ["provider=aws tag\_key=... tag\_value=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider) (required) - the name of the provider ("aws" in this case).

* [tag\_key](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_key) (required) - the key of the tag to auto-join on.

* [tag\_value](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_value) (required) - the value of the tag to auto-join on.

* [region](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "region) (optional) - the AWS region to authenticate in.

* [addr\_type](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "addr_type) (optional) - the type of address to discover: private\_v4, public\_v4, public\_v6. Default is private\_v4. (>= 1.0)

* [access\_key\_id](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "access_key_id) (optional) - the AWS access key for authentication (see below for more information about authenticating).

* [secret\_access\_key](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "secret_access_key) (optional) - the AWS secret access key for authentication (see below for more information about authenticating).

#### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "authentication-amp-precedence)Authentication & Precedence

* Static credentials access\_key\_id=... secret\_access\_key=...
* Environment variables (AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY)
* Shared credentials file (~/.aws/credentials or the path specified by AWS\_SHARED\_CREDENTIALS\_FILE)
* ECS task role metadata (container-specific).
* EC2 instance role metadata.

The only required IAM permission is ec2:DescribeInstances, and it is recommended that you make a dedicated key used only for auto-joining. If the region is omitted it will be discovered through the local instance's [EC2 metadata endpoint](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-identity-documents.html).

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "microsoft-azure)Microsoft Azure

This returns the first private IP address of all servers in the given region which have the given tag\_key and tag\_value in the tenant and subscription, or in the given resource\_group of a vm\_scale\_set for Virtual Machine Scale Sets.

$ consul agent -retry-join "provider=azure tag\_name=... tag\_value=... tenant\_id=... client\_id=... subscription\_id=... secret\_access\_key=..."

{

"retry\_join": ["provider=azure tag\_name=... tag\_value=... tenant\_id=... client\_id=... subscription\_id=... secret\_access\_key=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-1) (required) - the name of the provider ("azure" in this case).

* [tenant\_id](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tenant_id) (required) - the tenant to join machines in.

* [client\_id](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "client_id) (required) - the client to authenticate with.

* [secret\_access\_key](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "secret_access_key-1) (required) - the secret client key. **NOTE** This value often may have an equals sign in it's value, especially if generated from the Azure Portal, so is important to wrap in single quotes eg. secret\_acccess\_key='fpOfcHQJAQBczjAxiVpeyLmX1M0M0KPBST+GU2GvEN4='

Variables can also be provided by environmental variables:

* [ARM\_SUBSCRIPTION\_ID](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "arm_subscription_id) for subscription

* [ARM\_TENANT\_ID](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "arm_tenant_id) for tenant

* [ARM\_CLIENT\_ID](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "arm_client_id) for client

* [ARM\_CLIENT\_SECRET](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "arm_client_secret) for secret access key

Use these configuration parameters when using tags:

* [tag\_name](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_name) - the name of the tag to auto-join on.

* [tag\_value](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_value-1) - the value of the tag to auto-join on.

Use these configuration parameters when using Virtual Machine Scale Sets (Consul 1.0.3 and later):

* [resource\_group](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "resource_group) - the name of the resource group to filter on.

* [vm\_scale\_set](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "vm_scale_set) - the name of the virtual machine scale set to filter on.

When using tags the only permission needed is Microsoft.Network/networkInterfaces.

When using Virtual Machine Scale Sets the only role action needed is Microsoft.Compute/virtualMachineScaleSets/\*/read.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "google-compute-engine)Google Compute Engine

This returns the first private IP address of all servers in the given project which have the given tag\_value.

$ consul agent -retry-join "provider=gce project\_name=... tag\_value=..."

{

"retry\_join": ["provider=gce project\_name=... tag\_value=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-2) (required) - the name of the provider ("gce" in this case).

* [tag\_value](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_value-2) (required) - the value of the tag to auto-join on.

* [project\_name](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "project_name) (optional) - the name of the project to auto-join on. Discovered if not set.

* [zone\_pattern](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "zone_pattern) (optional) - the list of zones can be restricted through an RE2 compatible regular expression. If omitted, servers in all zones are returned.

* [credentials\_file](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "credentials_file) (optional) - the credentials file for authentication. Note, if you set -config-dir do not store the credentials.json file in the configuration directory as it will be parsed as a config file and Consul will fail to start. See below for more information.

#### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "authentication-amp-precedence-1)Authentication & Precedence

* Use credentials from credentials\_file, if provided.
* Use JSON file from GOOGLE\_APPLICATION\_CREDENTIALS environment variable.
* Use JSON file in a location known to the gcloud command-line tool.
  + On Windows, this is %APPDATA%/gcloud/application\_default\_credentials.json.
  + On other systems, $HOME/.config/gcloud/application\_default\_credentials.json.
* On Google Compute Engine, use credentials from the metadata server. In this final case any provided scopes are ignored.

Discovery requires a [GCE Service Account](https://cloud.google.com/compute/docs/access/service-accounts). Credentials are searched using the following paths, in order of precedence.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "ibm-softlayer)IBM SoftLayer

This returns the first private IP address of all servers for the given datacenter with the given tag\_value.

$ consul agent -retry-join "provider=softlayer datacenter=... tag\_value=... username=... api\_key=..."

{

"retry\_join": ["provider=softlayer datacenter=... tag\_value=... username=... api\_key=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-3) (required) - the name of the provider ("softlayer" in this case).

* [datacenter](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "sl_datacenter) (required) - the name of the datacenter to auto-join in.

* [tag\_value](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_value-3) (required) - the value of the tag to auto-join on.

* [username](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "username) (required) - the username to use for auth.

* [api\_key](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "api_key) (required) - the api key to use for auth.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "aliyun-alibaba-cloud-)Aliyun (Alibaba Cloud)

This returns the first private IP address of all servers for the given region with the given tag\_key and tag\_value.

$ consul agent -retry-join "provider=aliyun region=... tag\_key=consul tag\_value=... access\_key\_id=... access\_key\_secret=..."

{

"retry\_join": ["provider=aliyun region=... tag\_key=consul tag\_value=... access\_key\_id=... access\_key\_secret=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-4) (required) - the name of the provider ("aliyun" in this case).

* [region](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "region-1) (required) - the name of the region.

* [tag\_key](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_key-1) (required) - the key of the tag to auto-join on.

* [tag\_value](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_value-4) (required) - the value of the tag to auto-join on.

* [access\_key\_id](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "access_key_id-1) (required) -the access key to use for auth.

* [access\_key\_secret](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "access_key_secret) (required) - the secret key to use for auth.

The required RAM permission is ecs:DescribeInstances. It is recommended you make a dedicated key used only for auto-joining.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "digital-ocean)Digital Ocean

This returns the first private IP address of all servers for the given region with the given tag\_name.

$ consul agent -retry-join "provider=digitalocean region=... tag\_name=... api\_token=..."

{

"retry\_join": ["provider=digitalocean region=... tag\_name=... api\_token=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-5) (required) - the name of the provider ("digitalocean" in this case).

* [region](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "region-2) (required) - the name of the region.

* [tag\_name](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_name-1) (required) - the value of the tag to auto-join on.

* [api\_token](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "api_token) (required) -the token to use for auth.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "openstack)Openstack

This returns the first private IP address of all servers for the given region with the given tag\_key and tag\_value.

$ consul agent -retry-join "provider=os tag\_key=consul tag\_value=server username=... password=... auth\_url=..."

{

"retry\_join": ["provider=os tag\_key=consul tag\_value=server username=... password=... auth\_url=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-6) (required) - the name of the provider ("os" in this case).

* [tag\_key](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_key-2) (required) - the key of the tag to auto-join on.

* [tag\_value](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_value-5) (required) - the value of the tag to auto-join on.

* [project\_id](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "project_id) (optional) - the id of the project (tenant id).

* [username](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "username-1) (optional) - the username to use for auth.

* [password](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "password) (optional) - the password to use for auth.

* [token](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "token) (optional) - the token to use for auth.

* [auth\_url](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "auth_url) (optional) - the identity endpoint to use for auth.

* [insecure](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "insecure) (optional) - indicates whether the API certificate should not be checked. Any value means true.

The configuration can also be provided by environment variables.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "scaleway)Scaleway

This returns the first private IP address of all servers for the given region with the given tag\_name.

$ consul agent -retry-join "provider=scaleway organization=my-org tag\_name=consul-server token=... region=..."

{

"retry\_join": ["provider=scaleway organization=my-org tag\_name=consul-server token=... region=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-7) (required) - the name of the provider ("scaleway" in this case).

* [region](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "region-3) (required) - the name of the region.

* [tag\_name](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_name-2) (required) - the name of the tag to auto-join on.

* [organization](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "organization) (required) - the organization access key to use for auth (equal to access key).

* [token](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "token-1) (required) - the token to use for auth.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "joyent-triton)Joyent Triton

This returns the first PrimaryIP addresses for all servers with the given tag\_key and tag\_value.

$ consul agent -retry-join "provider=triton account=testaccount url=https://us-sw-1.api.joyentcloud.com key\_id=... tag\_key=consul-role tag\_value=server"

{

"retry\_join": ["provider=triton account=testaccount url=https://us-sw-1.api.joyentcloud.com key\_id=... tag\_key=consul-role tag\_value=server"]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-8) (required) - the name of the provider ("triton" in this case).

* [account](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "account) (required) - the name of the account.

* [url](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "url) (required) - the URL of the Triton api endpoint to use.

* [key\_id](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "key_id) (required) - the key id to use.

* [tag\_key](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_key-3) (optional) - the instance tag key to use.

* [tag\_value](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_value-6) (optional) - the tag value to use.

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "vsphere)vSphere

This returns the first private IP address of all servers for the given region with the given tag\_name and category\_name.

$ consul agent -retry-join "provider=vsphere category\_name=consul-role tag\_name=consul-server host=... user=... password=... insecure\_ssl=[true|false]"

{

"retry-join": ["provider=vsphere category\_name=consul-role tag\_name=consul-server host=... user=... password=... insecure\_ssl=[true|false]"]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-9) (required) - the name of the provider ("vsphere" is the provider here)

* [tag\_name](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "tag_name-3) (required) - The name of the tag to look up.

* [category\_name](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "category_name) (required) - The category of the tag to look up.

* [host](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "host) (required) - The host of the vSphere server to connect to.
* [user](https://www.consul.io/docs/agent/cloud-auto-join.html#user) (required) - The username to connect as.

* [password](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "password-1) (required) - The password of the user to connect to vSphere as.

* [insecure\_ssl](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "insecure_ssl) (optional) - Whether or not to skip SSL certificate validation.
* [timeout](https://www.consul.io/docs/agent/cloud-auto-join.html#timeout) (optional) - Discovery context timeout (default: 10m)

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "packet)Packet

This returns the first private IP address (or the IP addresso of address type) of all servers with the given project and auth\_token.

$ consul agent -retry-join "provider=packet auth\_token=token project=uuid url=... address\_type=..."

{

"retry-join": ["provider=packet auth\_token=token project=uuid url=... address\_type=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-10) (required) - the name of the provider ("packet" is the provider here)

* [project](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "project) (required) - the UUID of packet project

* [auth\_token](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "auth_token) (required) - the authentication token for packet

* [url](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "url-1) (optional) - a REST URL for packet

* [address\_type](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "address_type) (optional) - the type of address to check for in this provider ("private\_v4", "public\_v4" or "public\_v6". Defaults to "private\_v4")

### [»](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "kubernetes-k8s-)Kubernetes (k8s)

The Kubernetes provider finds the IP addresses of pods with the matching [label or field selector](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/). This is useful for non-Kubernetes agents that are joining a server cluster running within Kubernetes.

The pod IP is used by default, which requires that the agent connecting can network to the pod IP. The host\_networkboolean can be set to true to use the host IP instead, but this requires the agent ports (Gossip, RPC, etc.) to be exported to the host as well.

By default, no port is specified. This causes Consul to use the default gossip port (default behavior with all join requests). The pod may specify the consul.hashicorp.com/auto-join-port annotation to set the port. The value may be an integer or a named port.

$ consul agent -retry-join "provider=k8s label\_selector=\"app=consul,component=server\""

{

"retry-join": ["provider=k8s label\_selector=..."]

}

* [provider](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "provider-11) (required) - the name of the provider ("k8s" is the provider here)

* [kubeconfig](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "kubeconfig) (optional) - path to the kubeconfig file. If this isn't set, then in-cluster auth will be attempted. If that fails, the default kubeconfig paths are tried ($HOME/.kube/config).

* [namespace](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "namespace) (optional) - the namespace to search for pods. If this isn't set, it defaults to all namespaces.

* [label\_selector](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "label_selector) (optional) - the label selector for matching pods.

* [field\_selector](https://www.consul.io/docs/agent/cloud-auto-join.html" \l "field_selector) (optional) - the field selector for matching pods.

# Services

One of the main goals of service discovery is to provide a catalog of available services. To that end, the agent provides a simple service definition format to declare the availability of a service and to potentially associate it with a health check. A health check is considered to be application level if it is associated with a service. A service is defined in a configuration file or added at runtime over the HTTP interface.

## [»](https://www.consul.io/docs/agent/services.html" \l "service-definition)Service Definition

To configure a service, either provide the service definition as a -config-file option to the agent or place it inside the -config-dir of the agent. The file must end in the .json or .hcl extension to be loaded by Consul. Check definitions can be updated by sending a SIGHUP to the agent. Alternatively, the service can be registered dynamically using the [HTTP API](https://www.consul.io/api/index.html).

A service definition is a configuration that looks like the following. This example shows all possible fields, but note that only a few are required.

{

"service": {

"id": "redis",

"name": "redis",

"tags": ["primary"],

"address": "",

"meta": {

"meta": "for my service"

},

"port": 8000,

"enable\_tag\_override": **false**,

"checks": [

{

"args": ["/usr/local/bin/check\_redis.py"],

"interval": "10s"

}

],

"kind": "connect-proxy",

"proxy\_destination": "redis", // Deprecated

"proxy": {

"destination\_service\_name": "redis",

"destination\_service\_id": "redis1",

"local\_service\_address": "127.0.0.1",

"local\_service\_port": 9090,

"config": {},

"upstreams": []

},

"connect": {

"native": **false**,

"sidecar\_service": {}

"proxy": { // Deprecated

"command": [],

"config": {}

}

},

"weights": {

"passing": 5,

"warning": 1

}

}

}

A service definition must include a name and may optionally provide an id, tags, address, meta, port, enable\_tag\_override, and check. The id is set to the name if not provided. It is required that all services have a unique ID per node, so if names might conflict then unique IDs should be provided.

The tags property is a list of values that are opaque to Consul but can be used to distinguish between primary or secondary nodes, different versions, or any other service level labels.

The address field can be used to specify a service-specific IP address. By default, the IP address of the agent is used, and this does not need to be provided. The port field can be used as well to make a service-oriented architecture simpler to configure; this way, the address and port of a service can be discovered.

The meta object is a map of max 64 key/values with string semantics. Key can contain only ASCII chars and no special characters (A-Z a-z 0-9 \_ and -). For performance and security reasons, values as well as keys are limited to 128 characters for keys, 512 for values. This object has the same limitations as the node meta object in node definition. All those meta data can be retrieved individually per instance of the service and all the instances of a given service have their own copy of it.

Services may also contain a token field to provide an ACL token. This token is used for any interaction with the catalog for the service, including [anti-entropy syncs](https://www.consul.io/docs/internals/anti-entropy.html) and deregistration.

The enable\_tag\_override can optionally be specified to disable the anti-entropy feature for this service. If enable\_tag\_override is set to TRUE then external agents can update this service in the [catalog](https://www.consul.io/api/catalog.html) and modify the tags. Subsequent local sync operations by this agent will ignore the updated tags. For example, if an external agent modified both the tags and the port for this service and enable\_tag\_override was set to TRUE then after the next sync cycle the service's port would revert to the original value but the tags would maintain the updated value. As a counter example: If an external agent modified both the tags and port for this service and enable\_tag\_override was set to FALSE then after the next sync cycle the service's port and the tags would revert to the original value and all modifications would be lost.

It's important to note that this applies only to the locally registered service. If you have multiple nodes all registering the same service their enable\_tag\_override configuration and all other service configuration items are independent of one another. Updating the tags for the service registered on one node is independent of the same service (by name) registered on another node. If enable\_tag\_override is not specified the default value is false. See [anti-entropy syncs](https://www.consul.io/docs/internals/anti-entropy.html) for more info.

For Consul 0.9.3 and earlier you need to use enableTagOverride. Consul 1.0 supports both enable\_tag\_override and enableTagOverride but the latter is deprecated and has been removed as of Consul 1.1.

### [»](https://www.consul.io/docs/agent/services.html#connect)Connect

The kind field is used to optionally identify the service as a [Connect proxy](https://www.consul.io/docs/connect/proxies.html) instance with the value connect-proxy. For typical non-proxy instances the kind field must be omitted. The proxy field is also required for Connect proxy registrations and is only valid if kind is connect-proxy. The only required proxy field is destination\_service\_name. For more detail please see [complete proxy configuration example](https://www.consul.io/docs/connect/proxies.html#complete-configuration-example)

**Deprecation Notice:** From version 1.2.0 to 1.3.0, proxy destination was specified using proxy\_destination at the top level. This will continue to work until at least 1.5.0 but it's highly recommended to switch to usingproxy.destination\_service\_name.

The connect field can be specified to configure [Connect](https://www.consul.io/docs/connect/index.html) for a service. This field is available in Consul 1.2.0 and later. The native value can be set to true to advertise the service as [Connect-native](https://www.consul.io/docs/connect/native.html). The sidecar\_service field is an optional nested service definition its behavior and defaults are described in [Sidecar Service Registration](https://www.consul.io/docs/connect/proxies/sidecar-service.html). If native is true, it is an error to also specify a sidecar service registration.

**Deprecation Notice:** From version 1.2.0 to 1.3.0 during beta, Connect supported "Managed" proxies which are specified with the connect.proxy field. [Managed Proxies are deprecated](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) and the connect.proxy field will be removed in a future major release.

### [»](https://www.consul.io/docs/agent/services.html" \l "checks)Checks

A service can have an associated health check. This is a powerful feature as it allows a web balancer to gracefully remove failing nodes, a database to replace a failed secondary, etc. The health check is strongly integrated in the DNS interface as well. If a service is failing its health check or a node has any failing system-level check, the DNS interface will omit that node from any service query.

There are several check types that have differing required options as [documented here](https://www.consul.io/docs/agent/checks.html). The check name is automatically generated as service:<service-id>. If there are multiple service checks registered, the ID will be generated as service:<service-id>:<num> where <num> is an incrementing number starting from 1.

**Note:** There is more information about [checks here](https://www.consul.io/docs/agent/checks.html).

### [»](https://www.consul.io/docs/agent/services.html" \l "dns-srv-weights)DNS SRV Weights

The weights field is an optional field to specify the weight of a service in DNS SRV responses. If this field is not specified, its default value is: "weights": {"passing": 1, "warning": 1}. When a service is critical, it is excluded from DNS responses. Services with warning checks are included in responses by default, but excluded if the optional param only\_passing = true is present in agent DNS configuration or ?passing is used via the API.

When DNS SRV requests are made, the response will include the weights specified given the state of the service. This allows some instances to be given higher weight if they have more capacity, and optionally allows reducing load on services with checks in warning status by giving passing instances a higher weight.

### [»](https://www.consul.io/docs/agent/services.html" \l "enable-tag-override-and-anti-entropy)Enable Tag Override and Anti-Entropy

Services may also contain a token field to provide an ACL token. This token is used for any interaction with the catalog for the service, including [anti-entropy syncs](https://www.consul.io/docs/internals/anti-entropy.html) and deregistration.

You can optionally disable the anti-entropy feature for this service using the enable\_tag\_override flag. External agents can modify tags on services in the catalog, so subsequent sync operations can either maintain tag modifications or revert them. If enable\_tag\_override is set to TRUE, the next sync cycle may revert some service properties, **but** the tags would maintain the updated value. If enable\_tag\_override is set to FALSE, the next sync cycle will revert any updated service properties, **including** tags, to their original value.

It's important to note that this applies only to the locally registered service. If you have multiple nodes all registering the same service their enable\_tag\_override configuration and all other service configuration items are independent of one another. Updating the tags for the service registered on one node is independent of the same service (by name) registered on another node. If enable\_tag\_override is not specified the default value is false. See [anti-entropy syncs](https://www.consul.io/docs/internals/anti-entropy.html) for more info.

For Consul 0.9.3 and earlier you need to use enableTagOverride. Consul 1.0 supports both enable\_tag\_override and enableTagOverride but the latter is deprecated and has been removed as of Consul 1.1.

## [»](https://www.consul.io/docs/agent/services.html" \l "multiple-service-definitions)Multiple Service Definitions

Multiple services definitions can be provided at once using the plural services key in your configuration file.

{

"services": [

{

"id": "red0",

"name": "redis",

"tags": [

"primary"

],

"address": "",

"port": 6000,

"checks": [

{

"args": ["/bin/check\_redis", "-p", "6000"],

"interval": "5s",

"ttl": "20s"

}

]

},

{

"id": "red1",

"name": "redis",

"tags": [

"delayed",

"secondary"

],

"address": "",

"port": 7000,

"checks": [

{

"args": ["/bin/check\_redis", "-p", "7000"],

"interval": "30s",

"ttl": "60s"

}

]

},

...

]

}

## [»](https://www.consul.io/docs/agent/services.html" \l "service-and-tag-names-with-dns)Service and Tag Names with DNS

Consul exposes service definitions and tags over the [DNS](https://www.consul.io/docs/agent/dns.html) interface. DNS queries have a strict set of allowed characters and a well-defined format that Consul cannot override. While it is possible to register services or tags with names that don't match the conventions, those services and tags will not be discoverable via the DNS interface. It is recommended to always use DNS-compliant service and tag names.

DNS-compliant service and tag names may contain any alpha-numeric characters, as well as dashes. Dots are not supported because Consul internally uses them to delimit service tags.

## [»](https://www.consul.io/docs/agent/services.html" \l "service-definition-parameter-case)Service Definition Parameter Case

For historical reasons Consul's API uses CamelCased parameter names in responses, however it's configuration file uses snake\_case for both HCL and JSON representations. For this reason the registration HTTP APIs accept both name styles for service definition parameters although APIs will return the listings using CamelCase.

Note though that **all config file formats require snake\_case fields**. We always document service definition examples using snake\_case and JSON since this format works in both config files and API calls.

# Checks

One of the primary roles of the agent is management of system-level and application-level health checks. A health check is considered to be application-level if it is associated with a service. If not associated with a service, the check monitors the health of the entire node.

A check is defined in a configuration file or added at runtime over the HTTP interface. Checks created via the HTTP interface persist with that node.

There are several different kinds of checks:

* Script + Interval - These checks depend on invoking an external application that performs the health check, exits with an appropriate exit code, and potentially generates some output. A script is paired with an invocation interval (e.g. every 30 seconds). This is similar to the Nagios plugin system. The output of a script check is limited to 4KB. Output larger than this will be truncated. By default, Script checks will be configured with a timeout equal to 30 seconds. It is possible to configure a custom Script check timeout value by specifying the timeout field in the check definition. When the timeout is reached on Windows, Consul will wait for any child processes spawned by the script to finish. For any other system, Consul will attempt to force-kill the script and any child processes it has spawned once the timeout has passed. In Consul 0.9.0 and later, script checks are not enabled by default. To use them you can either use :
  + [enable\_local\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_local_script_checks): enable script checks defined in local config files. Script checks defined via the HTTP API will not be allowed.
  + [enable\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_script_checks): enable script checks regardless of how they are defined.

**Security Warning:** Enabling script checks in some configurations may introduce a remote execution vulnerability which is known to be targeted by malware. We strongly recommend enable\_local\_script\_checks instead. See [this blog post](https://www.hashicorp.com/blog/protecting-consul-from-rce-risk-in-specific-configurations) for more details.

* HTTP + Interval - These checks make an HTTP GET request every Interval (e.g. every 30 seconds) to the specified URL. The status of the service depends on the HTTP response code: any 2xx code is considered passing, a 429 Too Many Requests is a warning, and anything else is a failure. This type of check should be preferred over a script that uses curl or another external process to check a simple HTTP operation. By default, HTTP checks are GET requests unless the method field specifies a different method. Additional header fields can be set through the header field which is a map of lists of strings, e.g. {"x-foo": ["bar", "baz"]}. By default, HTTP checks will be configured with a request timeout equal to the check interval, with a max of 10 seconds. It is possible to configure a custom HTTP check timeout value by specifying the timeout field in the check definition. The output of the check is limited to roughly 4KB. Responses larger than this will be truncated. HTTP checks also support TLS. By default, a valid TLS certificate is expected. Certificate verification can be turned off by setting the tls\_skip\_verify field to true in the check definition.
* TCP + Interval - These checks make a TCP connection attempt every Interval (e.g. every 30 seconds) to the specified IP/hostname and port. If no hostname is specified, it defaults to "localhost". The status of the service depends on whether the connection attempt is successful (ie - the port is currently accepting connections). If the connection is accepted, the status is success, otherwise the status is critical. In the case of a hostname that resolves to both IPv4 and IPv6 addresses, an attempt will be made to both addresses, and the first successful connection attempt will result in a successful check. This type of check should be preferred over a script that uses netcat or another external process to check a simple socket operation. By default, TCP checks will be configured with a request timeout equal to the check interval, with a max of 10 seconds. It is possible to configure a custom TCP check timeout value by specifying the timeout field in the check definition.
* Time to Live (TTL) - These checks retain their last known state for a given TTL. The state of the check must be updated periodically over the HTTP interface. If an external system fails to update the status within a given TTL, the check is set to the failed state. This mechanism, conceptually similar to a dead man's switch, relies on the application to directly report its health. For example, a healthy app can periodically PUT a status update to the HTTP endpoint; if the app fails, the TTL will expire and the health check enters a critical state. The endpoints used to update health information for a given check are: [pass](https://www.consul.io/api/agent/check.html#ttl-check-pass), [warn](https://www.consul.io/api/agent/check.html#ttl-check-warn), [fail](https://www.consul.io/api/agent/check.html#ttl-check-fail), and [update](https://www.consul.io/api/agent/check.html#ttl-check-update). TTL checks also persist their last known status to disk. This allows the Consul agent to restore the last known status of the check across restarts. Persisted check status is valid through the end of the TTL from the time of the last check.
* Docker + Interval - These checks depend on invoking an external application which is packaged within a Docker Container. The application is triggered within the running container via the Docker Exec API. We expect that the Consul agent user has access to either the Docker HTTP API or the unix socket. Consul uses $DOCKER\_HOST to determine the Docker API endpoint. The application is expected to run, perform a health check of the service running inside the container, and exit with an appropriate exit code. The check should be paired with an invocation interval. The shell on which the check has to be performed is configurable which makes it possible to run containers which have different shells on the same host. Check output for Docker is limited to 4KB. Any output larger than this will be truncated. In Consul 0.9.0 and later, the agent must be configured with [enable\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_script_checks) set to true in order to enable Docker health checks.
* gRPC + Interval - These checks are intended for applications that support the standard [gRPC health checking protocol](https://github.com/grpc/grpc/blob/master/doc/health-checking.md). The state of the check will be updated at the given interval by probing the configured endpoint. By default, gRPC checks will be configured with a default timeout of 10 seconds. It is possible to configure a custom timeout value by specifying the timeout field in the check definition. gRPC checks will default to not using TLS, but TLS can be enabled by setting grpc\_use\_tls in the check definition. If TLS is enabled, then by default, a valid TLS certificate is expected. Certificate verification can be turned off by setting the tls\_skip\_verify field to true in the check definition.
* Alias - These checks alias the health state of another registered node or service. The state of the check will be updated asynchronously, but is nearly instant. For aliased services on the same agent, the local state is monitored and no additional network resources are consumed. For other services and nodes, the check maintains a blocking query over the agent's connection with a current server and allows stale requests. If there are any errors in watching the aliased node or service, the check state will be critical. For the blocking query, the check will use the ACL token set on the service or check definition or otherwise will fall back to the default ACL token set with the agent (acl\_token).

## [»](https://www.consul.io/docs/agent/checks.html" \l "check-definition)Check Definition

A script check:

{

"check": {

"id": "mem-util",

"name": "Memory utilization",

"args": ["/usr/local/bin/check\_mem.py", "-limit", "256MB"],

"interval": "10s",

"timeout": "1s"

}

}

A HTTP check:

{

"check": {

"id": "api",

"name": "HTTP API on port 5000",

"http": "https://localhost:5000/health",

"tls\_skip\_verify": **false**,

"method": "POST",

"header": {"x-foo":["bar", "baz"]},

"interval": "10s",

"timeout": "1s"

}

}

A TCP check:

{

"check": {

"id": "ssh",

"name": "SSH TCP on port 22",

"tcp": "localhost:22",

"interval": "10s",

"timeout": "1s"

}

}

A TTL check:

{

"check": {

"id": "web-app",

"name": "Web App Status",

"notes": "Web app does a curl internally every 10 seconds",

"ttl": "30s"

}

}

A Docker check:

{

"check": {

"id": "mem-util",

"name": "Memory utilization",

"docker\_container\_id": "f972c95ebf0e",

"shell": "/bin/bash",

"args": ["/usr/local/bin/check\_mem.py"],

"interval": "10s"

}

}

A gRPC check:

{

"check": {

"id": "mem-util",

"name": "Service health status",

"grpc": "127.0.0.1:12345",

"grpc\_use\_tls": **true**,

"interval": "10s"

}

}

An alias check for a local service:

{

"check": {

"id": "web-alias",

"alias\_service": "web"

}

}

Each type of definition must include a name and may optionally provide an id and notes field. The id must be unique per agent otherwise only the last defined check with that id will be registered. If the id is not set and the check is embedded within a service definition a unique check id is generated. Otherwise, id will be set to name. If names might conflict, unique IDs should be provided.

The notes field is opaque to Consul but can be used to provide a human-readable description of the current state of the check. Similarly, an external process updating a TTL check via the HTTP interface can set the notes value.

Checks may also contain a token field to provide an ACL token. This token is used for any interaction with the catalog for the check, including [anti-entropy syncs](https://www.consul.io/docs/internals/anti-entropy.html) and deregistration. For Alias checks, this token is used if a remote blocking query is necessary to watch the state of the aliased node or service.

Script, TCP, HTTP, Docker, and gRPC checks must include an interval field. This field is parsed by Go's time package, and has the following [formatting specification](https://golang.org/pkg/time/#ParseDuration):

A duration string is a possibly signed sequence of decimal numbers, each with optional fraction and a unit suffix, such as "300ms", "-1.5h" or "2h45m". Valid time units are "ns", "us" (or "µs"), "ms", "s", "m", "h".

In Consul 0.7 and later, checks that are associated with a service may also contain an optional deregister\_critical\_service\_after field, which is a timeout in the same Go time format as interval and ttl. If a check is in the critical state for more than this configured value, then its associated service (and all of its associated checks) will automatically be deregistered. The minimum timeout is 1 minute, and the process that reaps critical services runs every 30 seconds, so it may take slightly longer than the configured timeout to trigger the deregistration. This should generally be configured with a timeout that's much, much longer than any expected recoverable outage for the given service.

To configure a check, either provide it as a -config-file option to the agent or place it inside the -config-dir of the agent. The file must end in a ".json" or ".hcl" extension to be loaded by Consul. Check definitions can also be updated by sending a SIGHUP to the agent. Alternatively, the check can be registered dynamically using the [HTTP API](https://www.consul.io/api/index.html).

## [»](https://www.consul.io/docs/agent/checks.html" \l "check-scripts)Check Scripts

A check script is generally free to do anything to determine the status of the check. The only limitations placed are that the exit codes must obey this convention:

* Exit code 0 - Check is passing
* Exit code 1 - Check is warning
* Any other code - Check is failing

This is the only convention that Consul depends on. Any output of the script will be captured and stored in the outputfield.

In Consul 0.9.0 and later, the agent must be configured with [enable\_script\_checks](https://www.consul.io/docs/agent/options.html#_enable_script_checks) set to true in order to enable script checks.

## [»](https://www.consul.io/docs/agent/checks.html" \l "initial-health-check-status)Initial Health Check Status

By default, when checks are registered against a Consul agent, the state is set immediately to "critical". This is useful to prevent services from being registered as "passing" and entering the service pool before they are confirmed to be healthy. In certain cases, it may be desirable to specify the initial state of a health check. This can be done by specifying the status field in a health check definition, like so:

{

"check": {

"id": "mem",

"args": ["/bin/check\_mem", "-limit", "256MB"],

"interval": "10s",

"status": "passing"

}

}

The above service definition would cause the new "mem" check to be registered with its initial state set to "passing".

## [»](https://www.consul.io/docs/agent/checks.html" \l "service-bound-checks)Service-bound checks

Health checks may optionally be bound to a specific service. This ensures that the status of the health check will only affect the health status of the given service instead of the entire node. Service-bound health checks may be provided by adding a service\_id field to a check configuration:

{

"check": {

"id": "web-app",

"name": "Web App Status",

"service\_id": "web-app",

"ttl": "30s"

}

}

In the above configuration, if the web-app health check begins failing, it will only affect the availability of the web-app service. All other services provided by the node will remain unchanged.

## [»](https://www.consul.io/docs/agent/checks.html" \l "agent-certificates-for-tls-checks)Agent Certificates for TLS Checks

The [enable\_agent\_tls\_for\_checks](https://www.consul.io/docs/agent/options.html#enable_agent_tls_for_checks) agent configuration option can be utilized to have HTTP or gRPC health checks to use the agent's credentials when configured for TLS.

## [»](https://www.consul.io/docs/agent/checks.html" \l "multiple-check-definitions)Multiple Check Definitions

Multiple check definitions can be defined using the checks (plural) key in your configuration file.

{

"checks": [

{

"id": "chk1",

"name": "mem",

"args": ["/bin/check\_mem", "-limit", "256MB"],

"interval": "5s"

},

{

"id": "chk2",

"name": "/health",

"http": "http://localhost:5000/health",

"interval": "15s"

},

{

"id": "chk3",

"name": "cpu",

"args": ["/bin/check\_cpu"],

"interval": "10s"

},

...

]

}

# Consul KV

Consul KV is a core feature of Consul and is installed with the Consul agent. Once installed with the agent, it will have sane defaults. Consul KV allows users to store indexed objects, though its main uses are storing configuration parameters and metadata. Please note that it is a simple KV store and is not intended to be a full featured datastore (such as DynamoDB) but has some similarities to one.

The Consul KV datastore is located on the servers, but can be accessed by any agent (client or server). The natively integrated [RPC functionality](https://www.consul.io/docs/internals/architecture.html) allows clients to forward requests to servers, including key/value reads and writes. Part of Consul’s core design allows data to be replicated automatically across all the servers. Having a quorum of servers will decrease the risk of data loss if an outage occurs.

## [»](https://www.consul.io/docs/agent/kv.html" \l "accessing-the-kv-store)Accessing the KV store

The KV store can be accessed by the [consul kv CLI subcommands](https://www.consul.io/docs/commands/kv.html), [HTTP API](https://www.consul.io/api/kv.html), and Consul UI. To restrict access, enable and configure [ACLs](https://learn.hashicorp.com/consul/advanced/day-1-operations/acl-guide). Once the ACL system has been bootstrapped, users and services, will need a valid token with KV [privileges](https://www.consul.io/docs/agent/acl-rules.html#key-value-rules) to access the the data store, this includes even reads. We recommend creating a token with limited privileges, for example, you could create a token with write privileges on one key for developers to update the value related to their application.

The datastore itself is located on the Consul servers in the [data directory](https://www.consul.io/docs/agent/options.html#_data_dir). To ensure data is not lost in the event of a complete outage, use the [consul snapshot](https://www.consul.io/docs/commands/snapshot/restore.html) feature to backup the data.

## [»](https://www.consul.io/docs/agent/kv.html" \l "using-consul-kv)Using Consul KV

Objects are opaque to Consul, meaning there are no restrictions on the type of object stored in a key/value entry. The main restriction on an object is size - the maximum is 512 KB. Due to the maximum object size and main use cases, you should not need extra storage; the general [sizing recommendations](https://www.consul.io/docs/commands/snapshot/restore.html) are usually sufficient.

Keys, like objects are not restricted by type and can include any character. However, we recommend using URL-safe chars - [a-zA-Z0-9-\_] with the exception of /, which can be used to help organize data. Note, / will be treated like any other character and is not fixed to the file system. Meaning, including / in a key does not fix it to a directory structure. This model is similar to Amazon S3 buckets. However, / is still useful for organizing data and when recursively searching within the data store. We also recommend that you avoid the use of \*, ?, ', and % because they can cause issues when using the API and in shell scripts.

If you have not used Consul KV, check out this [Getting Started guide](https://learn.hashicorp.com/consul/getting-started/kv) on HashiCorp Learn.

## [»](https://www.consul.io/docs/agent/kv.html" \l "extending-consul-kv)Extending Consul KV

### [»](https://www.consul.io/docs/agent/kv.html" \l "consul-template)Consul Template

If you plan to use Consul KV as part of your configuration management process review the [Consul Template](https://www.consul.io/docs/guides/consul-template.html) guide on how to update configuration based on value updates in the KV. Consul Template is based on Go Templates and allows for a series of scripted actions to be initiated on value changes to a Consul key.

### [»](https://www.consul.io/docs/agent/kv.html#watches)Watches

Consul KV can also be extended with the use of watches. [Watches](https://www.consul.io/docs/agent/watches.html) are a way to monitor data for updates. When an update is detected, an external handler is invoked. To use watches with the KV store the [key](https://www.consul.io/docs/agent/watches.html#key) watch type should be used.

### [»](https://www.consul.io/docs/agent/kv.html" \l "consul-sessions)Consul Sessions

Consul sessions can be used to build distributed locks with Consul KV. Sessions act as a binding layer between nodes, health checks, and key/value data. The KV API supports an acquire and release operation. The acquire operation acts like a Check-And-Set operation. On success, there is a key update and an increment to the LockIndex and the session value is updated to reflect the session holding the lock. Review the session documentation for more information on the [integration](https://www.consul.io/docs/internals/sessions.html#k-v-integration)

### [»](https://www.consul.io/docs/agent/kv.html" \l "vault)Vault

If you plan to use Consul KV as a backend for Vault, please review [this guide](https://learn.hashicorp.com/vault/operations/ops-vault-ha-consul).

# Sentinel Overview

**ENTERPRISE**

This feature requires [Consul Enterprise](https://www.hashicorp.com/products/consul/)

Consul 1.0 adds integration with [Sentinel](https://hashicorp.com/sentinel) for policy enforcement. Sentinel policies help extend the ACL system in Consul beyond the static "read", "write", and "deny" policies to support full conditional logic and integration with external systems.

## [»](https://www.consul.io/docs/agent/sentinel.html" \l "sentinel-in-consul)Sentinel in Consul

Sentinel policies are applied during writes to the KV Store.

An optional sentinel field specifying code and enforcement level can be added to [ACL policy definitions](https://www.consul.io/docs/agent/acl-rules.html#sentinel-integration) for Consul KV. The following policy ensures that the value written during a KV update must end with "dc1".

key "datacenter\_name" {

policy = "write"

sentinel {

code = <<EOF

import "strings"

main = rule { strings.has\_suffix(value, "dc1") }

EOF

enforcementlevel = "soft-mandatory"

}

}

If the enforcementlevel property is not set, it defaults to "hard-mandatory".

## [»](https://www.consul.io/docs/agent/sentinel.html" \l "imports)Imports

Consul imports all the [standard imports](https://docs.hashicorp.com/sentinel/imports/) from Sentinel. All functions in these imports are available to be used in policies.

## [»](https://www.consul.io/docs/agent/sentinel.html" \l "injected-variables)Injected Variables

Consul passes some context as variables into Sentinel, which are available to use inside any policies you write.

#### [»](https://www.consul.io/docs/agent/sentinel.html" \l "variables-injected-during-kv-store-writes)Variables injected during KV store writes

| **Variable Name** | **Type** | **Description** |
| --- | --- | --- |
| key | string | Key being written |
| value | string | Value being written |
| flags | uint64 | [Flags](https://www.consul.io/api/kv.html#flags) |

## [»](https://www.consul.io/docs/agent/sentinel.html" \l "sentinel-examples)Sentinel Examples

The following are two examples of ACL policies with Sentinel rules.

### [»](https://www.consul.io/docs/agent/sentinel.html" \l "required-key-suffix)Required Key Suffix

Any values stored under the key prefix "dc1" must end with "dev"

key "dc1" {

policy = "write"

sentinel {

code = <<EOF

import "strings"

main = rule { strings.has\_suffix(value, "dev") }

EOF

}

}

### [»](https://www.consul.io/docs/agent/sentinel.html" \l "restrited-update-time)Restrited Update Time

The key "haproxy\_version" can only be updated during business hours.

key "haproxy\_version" {

policy = "write"

sentinel {

code = <<EOF

import "time"

main = rule { time.hour > 8 and time.hour < 17 }

EOF

}

}

# Encryption

The Consul agent supports encrypting all of its network traffic. The exact method of encryption is described on the [encryption internals page](https://www.consul.io/docs/internals/security.html). There are two separate encryption systems, one for gossip traffic and one for RPC. If you are configuring encryption, review this [guide](https://www.consul.io/docs/guides/agent-encryption.html).

## [»](https://www.consul.io/docs/agent/encryption.html" \l "gossip-encryption)Gossip Encryption

Enabling gossip encryption only requires that you set an encryption key when starting the Consul agent. The key can be set via the encrypt parameter.

**WAN Joined Datacenters Note:** If using multiple WAN joined datacenters, be sure to use the same encryption key in all datacenters.

The key must be 16-bytes, Base64 encoded. As a convenience, Consul provides the [consul keygen](https://www.consul.io/docs/commands/keygen.html) command to generate a cryptographically suitable key:

$ consul keygen

cg8StVXbQJ0gPvMd9o7yrg==

With that key, you can enable encryption on the agent. If encryption is enabled, the output of [consul agent](https://www.consul.io/docs/commands/agent.html) will include "Encrypt: true":

$ cat encrypt.json

{"encrypt": "cg8StVXbQJ0gPvMd9o7yrg=="}

$ consul agent -data-dir=/tmp/consul -config-file=encrypt.json

==> WARNING: LAN keyring exists but -encrypt given, using keyring

==> WARNING: WAN keyring exists but -encrypt given, using keyring

==> Starting Consul agent...

==> Starting Consul agent RPC...

==> Consul agent running!

Node name: 'Armons-MacBook-Air.local'

Datacenter: 'dc1'

Server: false (bootstrap: false)

Client Addr: 127.0.0.1 (HTTP: 8500, HTTPS: -1, DNS: 8600, RPC: 8400)

Cluster Addr: 10.1.10.12 (LAN: 8301, WAN: 8302)

Gossip encrypt: true, RPC-TLS: false, TLS-Incoming: false

...

All nodes within a Consul cluster must share the same encryption key in order to send and receive cluster information.

## [»](https://www.consul.io/docs/agent/encryption.html" \l "configuring-gossip-encryption-on-an-existing-cluster)Configuring Gossip Encryption on an existing cluster

As of version 0.8.4, Consul supports upshifting to encrypted gossip on a running cluster through the following process. Review this [step-by-step guide](https://www.consul.io/docs/guides/agent-encryption.html#enable-gossip-encryption-existing-cluster) to encrypt gossip on an existing cluster.

## [»](https://www.consul.io/docs/agent/encryption.html" \l "rpc-encryption-with-tls)RPC Encryption with TLS

Consul supports using TLS to verify the authenticity of servers and clients. To enable this, Consul requires that all clients and servers have key pairs that are generated by a single Certificate Authority. This can be a private CA, used only internally. The CA then signs keys for each of the agents, as in [this tutorial on generating both a CA and signing keys](https://www.consul.io/docs/guides/creating-certificates.html).

TLS can be used to verify the authenticity of the servers or verify the authenticity of clients. These modes are controlled by the [verify\_outgoing](https://www.consul.io/docs/agent/options.html#verify_outgoing), [verify\_server\_hostname](https://www.consul.io/docs/agent/options.html#verify_server_hostname), and [verify\_incoming](https://www.consul.io/docs/agent/options.html#verify_incoming) options, respectively.

If [verify\_outgoing](https://www.consul.io/docs/agent/options.html#verify_outgoing) is set, agents verify the authenticity of Consul for outgoing connections. Server nodes must present a certificate signed by a common certificate authority present on all agents, set via the agent's [ca\_file](https://www.consul.io/docs/agent/options.html#ca_file) and [ca\_path](https://www.consul.io/docs/agent/options.html#ca_path)options. All server nodes must have an appropriate key pair set using [cert\_file](https://www.consul.io/docs/agent/options.html#cert_file) and [key\_file](https://www.consul.io/docs/agent/options.html#key_file).

If [verify\_server\_hostname](https://www.consul.io/docs/agent/options.html#verify_server_hostname) is set, then outgoing connections perform hostname verification. All servers must have a certificate valid for server.<datacenter>.<domain> or the client will reject the handshake. This is a new configuration as of 0.5.1, and it is used to prevent a compromised client from being able to restart in server mode and perform a MITM (Man-In-The-Middle) attack. New deployments should set this to true, and generate the proper certificates, but this is defaulted to false to avoid breaking existing deployments.

If [verify\_incoming](https://www.consul.io/docs/agent/options.html#verify_incoming) is set, the servers verify the authenticity of all incoming connections. All clients must have a valid key pair set using [cert\_file](https://www.consul.io/docs/agent/options.html#cert_file) and [key\_file](https://www.consul.io/docs/agent/options.html#key_file). Servers will also disallow any non-TLS connections. To force clients to use TLS,[verify\_outgoing](https://www.consul.io/docs/agent/options.html#verify_outgoing) must also be set.

TLS is used to secure the RPC calls between agents, but gossip between nodes is done over UDP and is secured using a symmetric key. See above for enabling gossip encryption.

## [»](https://www.consul.io/docs/agent/encryption.html" \l "configuring-tls-on-an-existing-cluster)Configuring TLS on an existing cluster

As of version 0.8.4, Consul supports migrating to TLS-encrypted traffic on a running cluster without downtime. This process assumes a starting point with no TLS settings configured, and involves an intermediate step in order to get to full TLS encryption. Review this step-by-step [guide](https://www.consul.io/docs/guides/agent-encryption.html#enable-tls-existing-cluster) to learn how.

# Telemetry

The Consul agent collects various runtime metrics about the performance of different libraries and subsystems. These metrics are aggregated on a ten second interval and are retained for one minute.

To view this data, you must send a signal to the Consul process: on Unix, this is USR1 while on Windows it is BREAK. Once Consul receives the signal, it will dump the current telemetry information to the agent's stderr.

This telemetry information can be used for debugging or otherwise getting a better view of what Consul is doing.

Additionally, if the [telemetry configuration options](https://www.consul.io/docs/agent/options.html#telemetry) are provided, the telemetry information will be streamed to a [statsite](http://github.com/armon/statsite)or [statsd](http://github.com/etsy/statsd) server where it can be aggregated and flushed to Graphite or any other metrics store. This information can also be viewed with the [metrics endpoint](https://www.consul.io/api/agent.html#view-metrics) in JSON format or using [Prometheus](https://prometheus.io/) format.

Below is sample output of a telemetry dump:

[2014-01-29 10:56:50 -0800 PST][G] 'consul-agent.runtime.num\_goroutines': 19.000

[2014-01-29 10:56:50 -0800 PST][G] 'consul-agent.runtime.alloc\_bytes': 755960.000

[2014-01-29 10:56:50 -0800 PST][G] 'consul-agent.runtime.malloc\_count': 7550.000

[2014-01-29 10:56:50 -0800 PST][G] 'consul-agent.runtime.free\_count': 4387.000

[2014-01-29 10:56:50 -0800 PST][G] 'consul-agent.runtime.heap\_objects': 3163.000

[2014-01-29 10:56:50 -0800 PST][G] 'consul-agent.runtime.total\_gc\_pause\_ns': 1151002.000

[2014-01-29 10:56:50 -0800 PST][G] 'consul-agent.runtime.total\_gc\_runs': 4.000

[2014-01-29 10:56:50 -0800 PST][C] 'consul-agent.agent.ipc.accept': Count: 5 Sum: 5.000

[2014-01-29 10:56:50 -0800 PST][C] 'consul-agent.agent.ipc.command': Count: 10 Sum: 10.000

[2014-01-29 10:56:50 -0800 PST][C] 'consul-agent.serf.events': Count: 5 Sum: 5.000

[2014-01-29 10:56:50 -0800 PST][C] 'consul-agent.serf.events.foo': Count: 4 Sum: 4.000

[2014-01-29 10:56:50 -0800 PST][C] 'consul-agent.serf.events.baz': Count: 1 Sum: 1.000

[2014-01-29 10:56:50 -0800 PST][S] 'consul-agent.memberlist.gossip': Count: 50 Min: 0.007 Mean: 0.020 Max: 0.041 Stddev: 0.007 Sum: 0.989

[2014-01-29 10:56:50 -0800 PST][S] 'consul-agent.serf.queue.Intent': Count: 10 Sum: 0.000

[2014-01-29 10:56:50 -0800 PST][S] 'consul-agent.serf.queue.Event': Count: 10 Min: 0.000 Mean: 2.500 Max: 5.000 Stddev: 2.121 Sum: 25.000

# Key Metrics

These are some metrics emitted that can help you understand the health of your cluster at a glance. For a full list of metrics emitted by Consul, see [Metrics Reference](https://www.consul.io/docs/agent/telemetry.html#metrics-reference)

### [»](https://www.consul.io/docs/agent/telemetry.html" \l "transaction-timing)Transaction timing

| **Metric Name** | **Description** |
| --- | --- |
| consul.kvs.apply | This measures the time it takes to complete an update to the KV store. |
| consul.txn.apply | This measures the time spent applying a transaction operation. |
| consul.raft.apply | This counts the number of Raft transactions occurring over the interval. |
| consul.raft.commitTime | This measures the time it takes to commit a new entry to the Raft log on the leader. |

**Why they're important:** Taken together, these metrics indicate how long it takes to complete write operations in various parts of the Consul cluster. Generally these should all be fairly consistent and no more than a few milliseconds. Sudden changes in any of the timing values could be due to unexpected load on the Consul servers, or due to problems on the servers themselves.

**What to look for:** Deviations (in any of these metrics) of more than 50% from baseline over the previous hour.

### [»](https://www.consul.io/docs/agent/telemetry.html" \l "leadership-changes)Leadership changes

| **Metric Name** | **Description** |
| --- | --- |
| consul.raft.leader.lastContact | Measures the time since the leader was last able to contact the follower nodes when checking its leader lease. |
| consul.raft.state.candidate | This increments whenever a Consul server starts an election. |
| consul.raft.state.leader | This increments whenever a Consul server becomes a leader. |

**Why they're important:** Normally, your Consul cluster should have a stable leader. If there are frequent elections or leadership changes, it would likely indicate network issues between the Consul servers, or that the Consul servers themselves are unable to keep up with the load.

**What to look for:** For a healthy cluster, you're looking for a lastContact lower than 200ms, leader > 0 and candidate == 0. Deviations from this might indicate flapping leadership.

### [»](https://www.consul.io/docs/agent/telemetry.html#autopilot)Autopilot

| **Metric Name** | **Description** |
| --- | --- |
| consul.autopilot.healthy | This tracks the overall health of the local server cluster. If all servers are considered healthy by Autopilot, this will be set to 1. If any are unhealthy, this will be 0. |

**Why it's important:** Obviously, you want your cluster to be healthy.

**What to look for:** Alert if healthy is 0.

### [»](https://www.consul.io/docs/agent/telemetry.html" \l "memory-usage)Memory usage

| **Metric Name** | **Description** |
| --- | --- |
| consul.runtime.alloc\_bytes | This measures the number of bytes allocated by the Consul process. |
| consul.runtime.sys\_bytes | This is the total number of bytes of memory obtained from the OS. |

**Why they're important:** Consul keeps all of its data in memory. If Consul consumes all available memory, it will crash.

**What to look for:** If consul.runtime.sys\_bytes exceeds 90% of total avaliable system memory.

### [»](https://www.consul.io/docs/agent/telemetry.html" \l "garbage-collection)Garbage collection

| **Metric Name** | **Description** |
| --- | --- |
| consul.runtime.total\_gc\_pause\_ns | Number of nanoseconds consumed by stop-the-world garbage collection (GC) pauses since Consul started. |

**Why it's important:** GC pause is a "stop-the-world" event, meaning that all runtime threads are blocked until GC completes. Normally these pauses last only a few nanoseconds. But if memory usage is high, the Go runtime may GC so frequently that it starts to slow down Consul.

**What to look for:** Warning if total\_gc\_pause\_ns exceeds 2 seconds/minute, critical if it exceeds 5 seconds/minute.

**NOTE:** total\_gc\_pause\_ns is a cumulative counter, so in order to calculate rates (such as GC/minute), you will need to apply a function such as InfluxDB's [non\_negative\_difference()](https://docs.influxdata.com/influxdb/v1.5/query_language/functions/#non-negative-difference).

### [»](https://www.consul.io/docs/agent/telemetry.html" \l "network-activity-rpc-count)Network activity - RPC Count

| **Metric Name** | **Description** |
| --- | --- |
| consul.client.rpc | Increments whenever a Consul agent in client mode makes an RPC request to a Consul server |
| consul.client.rpc.exceeded | Increments whenever a Consul agent in client mode makes an RPC request to a Consul server gets rate limited by that agent's [limits](https://www.consul.io/docs/agent/options.html#limits) configuration. |
| consul.client.rpc.failed | Increments whenever a Consul agent in client mode makes an RPC request to a Consul server and fails. |

**Why they're important:** These measurements indicate the current load created from a Consul agent, including when the load becomes high enough to be rate limited. A high RPC count, especially from consul.client.rpcexceeded meaning that the requests are being rate-limited, could imply a misconfigured Consul agent.

**What to look for:** Sudden large changes to the consul.client.rpc metrics (greater than 50% deviation from baseline).consul.client.rpc.exceeded or consul.client.rpc.failed count > 0, as it implies that an agent is being rate-limited or fails to make an RPC request to a Consul server

When telemetry is being streamed to an external metrics store, the interval is defined to be that store's flush interval. Otherwise, the interval can be assumed to be 10 seconds when retrieving metrics from the built-in store using the above described signals.

## [»](https://www.consul.io/docs/agent/telemetry.html" \l "metrics-reference)Metrics Reference

This is a full list of metrics emitted by Consul.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Description** | **Unit** | **Type** |
| consul.acl.blocked.service.registration | This increments whenever a deregistration fails for a service (blocked by an ACL) | requests | counter |
| consul.acl.blocked.&lt;check|node|service&gt;.registration | This increments whenever a registration fails for an entity (check, node or service) is blocked by an ACL | requests | counter |
| consul.client.rpc | This increments whenever a Consul agent in client mode makes an RPC request to a Consul server. This gives a measure of how much a given agent is loading the Consul servers. Currently, this is only generated by agents in client mode, not Consul servers. | requests | counter |
| consul.client.rpc.exceeded | This increments whenever a Consul agent in client mode makes an RPC request to a Consul server gets rate limited by that agent's [limits](https://www.consul.io/docs/agent/options.html#limits)configuration. This gives an indication that there's an abusive application making too many requests on the agent, or that the rate limit needs to be increased. Currently, this only applies to agents in client mode, not Consul servers. | rejected requests | counter |
| consul.client.rpc.failed | This increments whenever a Consul agent in client mode makes an RPC request to a Consul server and fails. | requests | counter |
| consul.client.api.catalog\_register.<node> | This increments whenever a Consul agent receives a catalog register request. | requests | counter |
| consul.client.api.success.catalog\_register.<node> | This increments whenever a Consul agent successfully responds to a catalog register request. | requests | counter |
| consul.client.rpc.error.catalog\_register.<node> | This increments whenever a Consul agent receives an RPC error for a catalog register request. | errors | counter |
| consul.client.api.catalog\_deregister.<node> | This increments whenever a Consul agent receives a catalog de-register request. | requests | counter |
| consul.client.api.success.catalog\_deregister.<node> | This increments whenever a Consul agent successfully responds to a catalog de-register request. | requests | counter |
| consul.client.rpc.error.catalog\_deregister.<node> | This increments whenever a Consul agent receives an RPC error for a catalog de-register request. | errors | counter |
| consul.client.api.catalog\_datacenters.<node> | This increments whenever a Consul agent receives a request to list datacenters in the catalog. | requests | counter |
| consul.client.api.success.catalog\_datacenters.<node> | This increments whenever a Consul agent successfully responds to a request to list datacenters. | requests | counter |
| consul.client.rpc.error.catalog\_datacenters.<node> | This increments whenever a Consul agent receives an RPC error for a request to list datacenters. | errors | counter |
| consul.client.api.catalog\_nodes.<node> | This increments whenever a Consul agent receives a request to list nodes from the catalog. | requests | counter |
| consul.client.api.success.catalog\_nodes.<node> | This increments whenever a Consul agent successfully responds to a request to list nodes. | requests | counter |
| consul.client.rpc.error.catalog\_nodes.<node> | This increments whenever a Consul agent receives an RPC error for a request to list nodes. | errors | counter |
| consul.client.api.catalog\_services.<node> | This increments whenever a Consul agent receives a request to list services from the catalog. | requests | counter |
| consul.client.api.success.catalog\_services.<node> | This increments whenever a Consul agent successfully responds to a request to list services. | requests | counter |
| consul.client.rpc.error.catalog\_services.<node> | This increments whenever a Consul agent receives an RPC error for a request to list services. | errors | counter |
| consul.client.api.catalog\_service\_nodes.<node> | This increments whenever a Consul agent receives a request to list nodes offering a service. | requests | counter |
| consul.client.api.success.catalog\_service\_nodes.<node> | This increments whenever a Consul agent successfully responds to a request to list nodes offering a service. | requests | counter |
| consul.client.rpc.error.catalog\_service\_nodes.<node> | This increments whenever a Consul agent receives an RPC error for a request to list nodes offering a service. | errors | counter |
| consul.client.api.catalog\_node\_services.<node> | This increments whenever a Consul agent receives a request to list services registered in a node. | requests | counter |
| consul.client.api.success.catalog\_node\_services.<node> | This increments whenever a Consul agent successfully responds to a request to list services in a service. | requests | counter |
| consul.client.rpc.error.catalog\_node\_services.<node> | This increments whenever a Consul agent receives an RPC error for a request to list services in a service. | errors | counter |
| consul.runtime.num\_goroutines | This tracks the number of running goroutines and is a general load pressure indicator. This may burst from time to time but should return to a steady state value. | number of goroutines | gauge |
| consul.runtime.alloc\_bytes | This measures the number of bytes allocated by the Consul process. This may burst from time to time but should return to a steady state value. | bytes | gauge |
| consul.runtime.heap\_objects | This measures the number of objects allocated on the heap and is a general memory pressure indicator. This may burst from time to time but should return to a steady state value. | number of objects | gauge |
| consul.acl.cache\_hit | The number of ACL cache hits. | hits | counter |
| consul.acl.cache\_miss | The number of ACL cache misses. | misses | counter |
| consul.acl.replication\_hit | The number of ACL replication cache hits (when not running in the ACL datacenter). | hits | counter |
| consul.dns.stale\_queries | This increments when an agent serves a query within the allowed stale threshold. | queries | counter |
| consul.dns.ptr\_query.<node> | This measures the time spent handling a reverse DNS query for the given node. | ms | timer |
| consul.dns.domain\_query.<node> | This measures the time spent handling a domain query for the given node. | ms | timer |
| consul.http.<verb>.<path> | This tracks how long it takes to service the given HTTP request for the given verb and path. Paths do not include details like service or key names, for these an underscore will be present as a placeholder (eg. consul.http.GET.v1.kv.\_) | ms | timer |

## [»](https://www.consul.io/docs/agent/telemetry.html" \l "server-health)Server Health

These metrics are used to monitor the health of the Consul servers.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Description** | **Unit** | **Type** |
| consul.raft.fsm.snapshot | This metric measures the time taken by the FSM to record the current state for the snapshot. | ms | timer |
| consul.raft.fsm.apply | This metric gives the number of logs committed since the last interval. | commit logs / interval | counter |
| consul.raft.fsm.restore | This metric measures the time taken by the FSM to restore its state from a snapshot. | ms | timer |
| consul.raft.snapshot.create | This metric measures the time taken to initialize the snapshot process. | ms | timer |
| consul.raft.snapshot.persist | This metric measures the time taken to dump the current snapshot taken by the Consul agent to the disk. | ms | timer |
| consul.raft.snapshot.takeSnapshot | This metric measures the total time involved in taking the current snapshot (creating one and persisting it) by the Consul agent. | ms | timer |
| consul.raft.replication.heartbeat | This metric measures the time taken to invoke appendEntries on a peer, so that it doesn’t timeout on a periodic basis. | ms | timer |
| consul.serf.snapshot.appendLine | This metric measures the time taken by the Consul agent to append an entry into the existing log. | ms | timer |
| consul.serf.snapshot.compact | This metric measures the time taken by the Consul agent to compact a log. This operation occurs only when the snapshot becomes large enough to justify the compaction . | ms | timer |
| consul.raft.state.leader | This increments whenever a Consul server becomes a leader. If there are frequent leadership changes this may be indication that the servers are overloaded and aren't meeting the soft real-time requirements for Raft, or that there are networking problems between the servers. | leadership transitions / interval | counter |
| consul.raft.state.candidate | This increments whenever a Consul server starts an election. If this increments without a leadership change occurring it could indicate that a single server is overloaded or is experiencing network connectivity issues. | election attempts / interval | counter |
| consul.raft.apply | This counts the number of Raft transactions occurring over the interval, which is a general indicator of the write load on the Consul servers. | raft transactions / interval | counter |
| consul.raft.barrier | This metric counts the number of times the agent has started the barrier i.e the number of times it has issued a blocking call, to ensure that the agent has all the pending operations that were queued, to be applied to the agent's FSM. | blocks / interval | counter |
| consul.raft.verify\_leader | This metric counts the number of times an agent checks whether it is still the leader or not | checks / interval | Counter |
| consul.raft.restore | This metric counts the number of times the restore operation has been performed by the agent. Here, restore refers to the action of raft consuming an external snapshot to restore its state. | operation invoked / interval | counter |
| consul.raft.commitTime | This measures the time it takes to commit a new entry to the Raft log on the leader. | ms | timer |
| consul.raft.leader.dispatchLog | This measures the time it takes for the leader to write log entries to disk. | ms | timer |
| consul.raft.replication.appendEntries | This measures the time it takes to replicate log entries to followers. This is a general indicator of the load pressure on the Consul servers, as well as the performance of the communication between the servers. | ms | timer |
| consul.raft.state.follower | This metric counts the number of times an agent has entered the follower mode. This happens when a new agent joins the cluster or after the end of a leader election. | follower state entered / interval | counter |
| consul.raft.transistion.heartbeat\_timeout | This metric gives the number of times an agent has transitioned to the Candidate state, after receive no heartbeat messages from the last known leader. | timeouts / interval | counter |
| consul.raft.restoreUserSnapshot | This metric measures the time taken by the agent to restore the FSM state from a user's snapshot | ms | timer |
| consul.raft.rpc.processHeartBeat | This metric measures the time taken to process a heartbeat request. | ms | timer |
| consul.raft.rpc.appendEntries | This metric measures the time taken to process an append entries RPC call from an agent. | ms | timer |
| consul.raft.rpc.appendEntries.storeLogs | This metric measures the time taken to add any outstanding logs for an agent, since the last appendEntries was invoked | ms | timer |
| consul.raft.rpc.appendEntries.processLogs | This metric measures the time taken to process the outstanding log entries of an agent. | ms | timer |
| consul.raft.rpc.requestVote | This metric measures the time taken to process the request vote RPC call. | ms | timer |
| consul.raft.rpc.installSnapshot | This metric measures the time taken to process the installSnapshot RPC call. This metric should only be seen on agents which are currently in the follower state. | ms | timer |
| consul.raft.replication.appendEntries.rpc | This metric measures the time taken by the append entries RFC, to replicate the log entries of a leader agent onto its follower agent(s) | ms | timer |
| consul.raft.replication.appendEntries.logs | This metric measures the number of logs replicated to an agent, to bring it upto speed with the leader's logs. | logs appended/ interval | counter |
| consul.raft.leader.lastContact | This will only be emitted by the Raft leader and measures the time since the leader was last able to contact the follower nodes when checking its leader lease. It can be used as a measure for how stable the Raft timing is and how close the leader is to timing out its lease.  The lease timeout is 500 ms times the [raft\_multiplier configuration](https://www.consul.io/docs/agent/options.html#raft_multiplier), so this telemetry value should not be getting close to that configured value, otherwise the Raft timing is marginal and might need to be tuned, or more powerful servers might be needed. See the [Server Performance](https://www.consul.io/docs/guides/performance.html) guide for more details. | ms | timer |
| consul.acl.apply | This measures the time it takes to complete an update to the ACL store. | ms | timer |
| consul.acl.fault | This measures the time it takes to fault in the rules for an ACL during a cache miss. | ms | timer |
| consul.acl.fetchRemoteACLs | This measures the time it takes to fetch remote ACLs during replication. | ms | timer |
| consul.acl.updateLocalACLs | This measures the time it takes to apply replication changes to the local ACL store. | ms | timer |
| consul.acl.replicateACLs | This measures the time it takes to do one pass of the ACL replication algorithm. | ms | timer |
| consul.acl.resolveToken | This measures the time it takes to resolve an ACL token. | ms | timer |
| consul.rpc.accept\_conn | This increments when a server accepts an RPC connection. | connections | counter |
| consul.catalog.register | This measures the time it takes to complete a catalog register operation. | ms | timer |
| consul.catalog.deregister | This measures the time it takes to complete a catalog deregister operation. | ms | timer |
| consul.fsm.register | This measures the time it takes to apply a catalog register operation to the FSM. | ms | timer |
| consul.fsm.deregister | This measures the time it takes to apply a catalog deregister operation to the FSM. | ms | timer |
| consul.fsm.acl.<op> | This measures the time it takes to apply the given ACL operation to the FSM. | ms | timer |
| consul.fsm.session.<op> | This measures the time it takes to apply the given session operation to the FSM. | ms | timer |
| consul.fsm.kvs.<op> | This measures the time it takes to apply the given KV operation to the FSM. | ms | timer |
| consul.fsm.tombstone.<op> | This measures the time it takes to apply the given tombstone operation to the FSM. | ms | timer |
| consul.fsm.coordinate.batch-update | This measures the time it takes to apply the given batch coordinate update to the FSM. | ms | timer |
| consul.fsm.prepared-query.<op> | This measures the time it takes to apply the given prepared query update operation to the FSM. | ms | timer |
| consul.fsm.txn | This measures the time it takes to apply the given transaction update to the FSM. | ms | timer |
| consul.fsm.autopilot | This measures the time it takes to apply the given autopilot update to the FSM. | ms | timer |
| consul.fsm.persist | This measures the time it takes to persist the FSM to a raft snapshot. | ms | timer |
| consul.kvs.apply | This measures the time it takes to complete an update to the KV store. | ms | timer |
| consul.leader.barrier | This measures the time spent waiting for the raft barrier upon gaining leadership. | ms | timer |
| consul.leader.reconcile | This measures the time spent updating the raft store from the serf member information. | ms | timer |
| consul.leader.reconcileMember | This measures the time spent updating the raft store for a single serf member's information. | ms | timer |
| consul.leader.reapTombstones | This measures the time spent clearing tombstones. | ms | timer |
| consul.prepared-query.apply | This measures the time it takes to apply a prepared query update. | ms | timer |
| consul.prepared-query.explain | This measures the time it takes to process a prepared query explain request. | ms | timer |
| consul.prepared-query.execute | This measures the time it takes to process a prepared query execute request. | ms | timer |
| consul.prepared-query.execute\_remote | This measures the time it takes to process a prepared query execute request that was forwarded to another datacenter. | ms | timer |
| consul.rpc.raft\_handoff | This increments when a server accepts a Raft-related RPC connection. | connections | counter |
| consul.rpc.request\_error | This increments when a server returns an error from an RPC request. | errors | counter |
| consul.rpc.request | This increments when a server receives a Consul-related RPC request. | requests | counter |
| consul.rpc.query | This increments when a server sends a (potentially blocking) RPC query. | queries | counter |
| consul.rpc.cross-dc | This increments when a server sends a (potentially blocking) cross datacenter RPC query. | queries | counter |
| consul.rpc.consistentRead | This measures the time spent confirming that a consistent read can be performed. | ms | timer |
| consul.session.apply | This measures the time spent applying a session update. | ms | timer |
| consul.session.renew | This measures the time spent renewing a session. | ms | timer |
| consul.session\_ttl.invalidate | This measures the time spent invalidating an expired session. | ms | timer |
| consul.txn.apply | This measures the time spent applying a transaction operation. | ms | timer |
| consul.txn.read | This measures the time spent returning a read transaction. | ms | timer |

## [»](https://www.consul.io/docs/agent/telemetry.html" \l "cluster-health)Cluster Health

These metrics give insight into the health of the cluster as a whole.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Description** | **Unit** | **Type** |
| consul.memberlist.degraded.probe | This metric counts the number of times the agent has performed failure detection on an other agent at a slower probe rate. The agent uses its own health metric as an indicator to perform this action. (If its health score is low, means that the node is healthy, and vice versa.) | probes / interval | counter |
| consul.memberlist.degraded.timeout | This metric counts the number of times an agent was marked as a dead node, whilst not getting enough confirmations from a randomly selected list of agent nodes in an agent's membership. | occurrence / interval | counter |
| consul.memberlist.msg.dead | This metric counts the number of times an agent has marked another agent to be a dead node. | messages / interval | counter |
| consul.memberlist.health.score | This metric describes a node's perception of its own health based on how well it is meeting the soft real-time requirements of the protocol. This metric ranges from 0 to 8, where 0 indicates "totally healthy". This health score is used to scale the time between outgoing probes, and higher scores translate into longer probing intervals. For more details see section IV of the Lifeguard paper: <https://arxiv.org/pdf/1707.00788.pdf> | score | gauge |
| consul.memberlist.msg.suspect | This increments when an agent suspects another as failed when executing random probes as part of the gossip protocol. These can be an indicator of overloaded agents, network problems, or configuration errors where agents can not connect to each other on the [required ports](https://www.consul.io/docs/agent/options.html#ports). | suspect messages received / interval | counter |
| consul.memberlist.tcp.accept | This metric counts the number of times an agent has accepted an incoming TCP stream connection. | connections accepted / interval | counter |
| consul.memberlist.udp.sent/received | This metric measures the total number of bytes sent/received by an agent through the UDP protocol. | bytes sent or bytes received / interval | counter |
| consul.memberlist.tcp.connect | This metric counts the number of times an agent has initiated a push/pull sync with an other agent. | push/pull initiated / interval | counter |
| consul.memberlist.tcp.sent | This metric measures the total number of bytes sent by an agent through the TCP protocol | bytes sent / interval | counter |
| consul.memberlist.gossip | This metric gives the number of gossips (messages) broadcasted to a set of randomly selected nodes. | messages / Interval | counter |
| consul.memberlist.msg\_alive | This metric counts the number of alive agents, that the agent has mapped out so far, based on the message information given by the network layer. | nodes / Interval | counter |
| consul.memberlist.msg\_dead | This metric gives the number of dead agents, that the agent has mapped out so far, based on the message information given by the network layer. | nodes / Interval | counter |
| consul.memberlist.msg\_suspect | This metric gives the number of suspect nodes, that the agent has mapped out so far, based on the message information given by the network layer. | nodes / Interval | counter |
| consul.memberlist.probeNode | This metric measures the time taken to perform a single round of failure detection on a select agent. | nodes / Interval | counter |
| consul.memberlist.pushPullNode | This metric measures the number of agents that have exchanged state with this agent. | nodes / Interval | counter |
| consul.serf.member.flap | Available in Consul 0.7 and later, this increments when an agent is marked dead and then recovers within a short time period. This can be an indicator of overloaded agents, network problems, or configuration errors where agents can not connect to each other on the [required ports](https://www.consul.io/docs/agent/options.html#ports). | flaps / interval | counter |
| consul.serf.events | This increments when an agent processes an [event](https://www.consul.io/docs/commands/event.html). Consul uses events internally so there may be additional events showing in telemetry. There are also a per-event counters emitted as consul.serf.events.<event name>. | events / interval | counter |
| consul.autopilot.failure\_tolerance | This tracks the number of voting servers that the cluster can lose while continuing to function. | servers | gauge |
| consul.autopilot.healthy | This tracks the overall health of the local server cluster. If all servers are considered healthy by Autopilot, this will be set to 1. If any are unhealthy, this will be 0. | boolean | gauge |
| consul.session\_ttl.active | This tracks the active number of sessions being tracked. | sessions | gauge |
| consul.catalog.service.query.<service> | This increments for each catalog query for the given service. | queries | counter |
| consul.catalog.service.query-tag.<service>.<tag> | This increments for each catalog query for the given service with the given tag. | queries | counter |
| consul.catalog.service.query-tags.<service>.<tags> | This increments for each catalog query for the given service with the given tags. | queries | counter |
| consul.catalog.service.not-found.<service> | This increments for each catalog query where the given service could not be found. | queries | counter |
| consul.health.service.query.<service> | This increments for each health query for the given service. | queries | counter |
| consul.health.service.query-tag.<service>.<tag> | This increments for each health query for the given service with the given tag. | queries | counter |
| consul.health.service.query-tags.<service>.<tags> | This increments for each health query for the given service with the given tags. | queries | counter |
| consul.health.service.not-found.<service> | This increments for each health query where the given service could not be found. | queries | counter |

## [»](https://www.consul.io/docs/agent/telemetry.html" \l "connect-built-in-proxy-metrics)Connect Built-in Proxy Metrics

Consul Connect's built-in proxy is by default configured to log metrics to the same sink as the agent that starts it when running as a [managed proxy](https://www.consul.io/docs/connect/proxies.html#managed-proxies).

When running in this mode it emits some basic metrics. These will be expanded upon in the future.

All metrics are prefixed with consul.proxy.<proxied-service-id> to distinguish between multiple proxies on a given host. The table below use web as an example service name for brevity.

### [»](https://www.consul.io/docs/agent/telemetry.html" \l "labels)Labels

Most labels have a dst label and some have a src label. When using metrics sinks and timeseries stores that support labels or tags, these allow aggregating the connections by service name.

Assuming all services are using a managed built-in proxy, you can get a complete overview of both number of open connections and bytes sent and recieved between all services by aggregating over these metrics.

For example aggregating over all upstream (i.e. outbound) connections which have both src and dst labels, you can get a sum of all the bandwidth in and out of a given service or the total number of connections between two services.

### [»](https://www.consul.io/docs/agent/telemetry.html" \l "metrics-reference-1)Metrics Reference

The standard go runtime metrics are exported by go-metrics as with Consul agent. The table below describes the additional metrics exported by the proxy.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Description** | **Unit** | **Type** |
| consul.proxy.web.runtime.\* | The same go runtime metrics as documented for the agent above. | mixed | mixed |
| consul.proxy.web.inbound.conns | Shows the current number of connections open from inbound requests to the proxy. Where supported a dst label is added indicating the service name the proxy represents. | connections | gauge |
| consul.proxy.web.inbound.rx\_bytes | This increments by the number of bytes received from an inbound client connection. Where supported a dst label is added indicating the service name the proxy represents. | bytes | counter |
| consul.proxy.web.inbound.tx\_bytes | This increments by the number of bytes transfered to an inbound client connection. Where supported a dst label is added indicating the service name the proxy represents. | bytes | counter |
| consul.proxy.web.upstream.conns | Shows the current number of connections open from a proxy instance to an upstream. Where supported a src label is added indicating the service name the proxy represents, and a dst label is added indicating the service name the upstream is connecting to. | connections | gauge |
| consul.proxy.web.inbound.rx\_bytes | This increments by the number of bytes received from an upstream connection. Where supported a src label is added indicating the service name the proxy represents, and a dst label is added indicating the service name the upstream is connecting to. | bytes | counter |
| consul.proxy.web.inbound.tx\_bytes | This increments by the number of bytes transfered to an upstream connection. Where supported a srclabel is added indicating the service name the proxy represents, and a dst label is added indicating the service name the upstream is connecting to. | bytes | counter |

# Watches

Watches are a way of specifying a view of data (e.g. list of nodes, KV pairs, health checks) which is monitored for updates. When an update is detected, an external handler is invoked. A handler can be any executable or HTTP endpoint. As an example, you could watch the status of health checks and notify an external system when a check is critical.

Watches are implemented using blocking queries in the [HTTP API](https://www.consul.io/api/index.html). Agents automatically make the proper API calls to watch for changes and inform a handler when the data view has updated.

Watches can be configured as part of the [agent's configuration](https://www.consul.io/docs/agent/options.html#watches), causing them to run once the agent is initialized. Reloading the agent configuration allows for adding or removing watches dynamically.

Alternatively, the [watch command](https://www.consul.io/docs/commands/watch.html) enables a watch to be started outside of the agent. This can be used by an operator to inspect data in Consul or to easily pipe data into processes without being tied to the agent lifecycle.

In either case, the type of the watch must be specified. Each type of watch supports different parameters, some required and some optional. These options are specified in a JSON body when using agent configuration or as CLI flags for the watch command.

## [»](https://www.consul.io/docs/agent/watches.html" \l "handlers)Handlers

The watch configuration specifies the view of data to be monitored. Once that view is updated, the specified handler is invoked. Handlers can be either an executable or an HTTP endpoint. A handler receives JSON formatted data with invocation info, following a format that depends on the type of the watch. Each watch type documents the format type. Because they map directly to an HTTP API, handlers should expect the input to match the format of the API. A Consul index is also given, corresponding to the responses from the [HTTP API](https://www.consul.io/api/index.html).

### [»](https://www.consul.io/docs/agent/watches.html" \l "executable)Executable

An executable handler reads the JSON invocation info from stdin. Additionally, the CONSUL\_INDEX environment variable will be set to the Consul index Anything written to stdout is logged.

Here is an example configuration, where handler\_type is optionally set to script:

{

"type": "key",

"key": "foo/bar/baz",

"handler\_type": "script",

"args": ["/usr/bin/my-service-handler.sh", "-redis"]

}

Prior to Consul 1.0, watches used a single handler field to define the command to run, and would always run in a shell. In Consul 1.0, the args array was added so that handlers can be run without a shell. The handler field is deprecated, and you should include the shell in the args to run under a shell, eg. "args": ["sh", "-c", "..."].

### [»](https://www.consul.io/docs/agent/watches.html" \l "http-endpoint)HTTP endpoint

An HTTP handler sends an HTTP request when a watch is invoked. The JSON invocation info is sent as a payload along the request. The response also contains the Consul index as a header named X-Consul-Index.

The HTTP handler can be configured by setting handler\_type to http. Additional handler options are set using http\_handler\_config. The only required parameter is the path field which specifies the URL to the HTTP endpoint. Consul uses POST as the default HTTP method, but this is also configurable. Other optional fields are header, timeout and tls\_skip\_verify. The watch invocation data is always sent as a JSON payload.

Here is an example configuration:

{

"type": "key",

"key": "foo/bar/baz",

"handler\_type": "http",

"http\_handler\_config": {

"path":"https://localhost:8000/watch",

"method": "POST",

"header": {"x-foo":["bar", "baz"]},

"timeout": "10s",

"tls\_skip\_verify": **false**

}

}

## [»](https://www.consul.io/docs/agent/watches.html" \l "global-parameters)Global Parameters

In addition to the parameters supported by each option type, there are a few global parameters that all watches support:

* [datacenter](https://www.consul.io/docs/agent/watches.html#datacenter) - Can be provided to override the agent's default datacenter.
* [token](https://www.consul.io/docs/agent/watches.html#token) - Can be provided to override the agent's default ACL token.

* [args](https://www.consul.io/docs/agent/watches.html" \l "args) - The handler subprocess and arguments to invoke when the data view updates.

* [handler](https://www.consul.io/docs/agent/watches.html" \l "handler) - The handler shell command to invoke when the data view updates.

## [»](https://www.consul.io/docs/agent/watches.html" \l "watch-types)Watch Types

The following types are supported. Detailed documentation on each is below:

* [key](https://www.consul.io/docs/agent/watches.html#key) - Watch a specific KV pair
* [keyprefix](https://www.consul.io/docs/agent/watches.html#keyprefix) - Watch a prefix in the KV store
* [services](https://www.consul.io/docs/agent/watches.html#services) - Watch the list of available services
* [nodes](https://www.consul.io/docs/agent/watches.html#nodes) - Watch the list of nodes
* [service](https://www.consul.io/docs/agent/watches.html#service)- Watch the instances of a service
* [checks](https://www.consul.io/docs/agent/watches.html#checks) - Watch the value of health checks
* [event](https://www.consul.io/docs/agent/watches.html#event) - Watch for custom user events

### [»](https://www.consul.io/docs/agent/watches.html" \l "type-key)Type: key

The "key" watch type is used to watch a specific key in the KV store. It requires that the "key" parameter be specified.

This maps to the /v1/kv/ API internally.

Here is an example configuration:

{

"type": "key",

"key": "foo/bar/baz",

"args": ["/usr/bin/my-service-handler.sh", "-redis"]

}

Or, using the watch command:

$ consul watch -type=key -key=foo/bar/baz /usr/bin/my-key-handler.sh

An example of the output of this command:

{

"Key": "foo/bar/baz",

"CreateIndex": 1793,

"ModifyIndex": 1793,

"LockIndex": 0,

"Flags": 0,

"Value": "aGV5",

"Session": ""

}

### [»](https://www.consul.io/docs/agent/watches.html" \l "type-keyprefix)Type: keyprefix

The "keyprefix" watch type is used to watch a prefix of keys in the KV store. It requires that the "prefix" parameter be specified. This watch returns all keys matching the prefix whenever any key matching the prefix changes.

This maps to the /v1/kv/ API internally.

Here is an example configuration:

{

"type": "keyprefix",

"prefix": "foo/",

"args": ["/usr/bin/my-service-handler.sh", "-redis"]

}

Or, using the watch command:

$ consul watch -type=keyprefix -prefix=foo/ /usr/bin/my-prefix-handler.sh

An example of the output of this command:

[

{

"Key": "foo/bar",

"CreateIndex": 1796,

"ModifyIndex": 1796,

"LockIndex": 0,

"Flags": 0,

"Value": "TU9BUg==",

"Session": ""

},

{

"Key": "foo/baz",

"CreateIndex": 1795,

"ModifyIndex": 1795,

"LockIndex": 0,

"Flags": 0,

"Value": "YXNkZg==",

"Session": ""

},

{

"Key": "foo/test",

"CreateIndex": 1793,

"ModifyIndex": 1793,

"LockIndex": 0,

"Flags": 0,

"Value": "aGV5",

"Session": ""

}

]

### [»](https://www.consul.io/docs/agent/watches.html" \l "type-services)Type: services

The "services" watch type is used to watch the list of available services. It has no parameters.

This maps to the /v1/catalog/services API internally.

An example of the output of this command:

{

"consul": [],

"redis": [],

"web": []

}

### [»](https://www.consul.io/docs/agent/watches.html" \l "type-nodes)Type: nodes

The "nodes" watch type is used to watch the list of available nodes. It has no parameters.

This maps to the /v1/catalog/nodes API internally.

An example of the output of this command:

[

{

"Node": "nyc1-consul-1",

"Address": "192.241.159.115"

},

{

"Node": "nyc1-consul-2",

"Address": "192.241.158.205"

},

{

"Node": "nyc1-consul-3",

"Address": "198.199.77.133"

},

{

"Node": "nyc1-worker-1",

"Address": "162.243.162.228"

},

{

"Node": "nyc1-worker-2",

"Address": "162.243.162.226"

},

{

"Node": "nyc1-worker-3",

"Address": "162.243.162.229"

}

]

### [»](https://www.consul.io/docs/agent/watches.html" \l "type-service)Type: service

The "service" watch type is used to monitor the providers of a single service. It requires the "service" parameter and optionally takes the parameters "tag" and "passingonly". The "tag" parameter will filter by tag, and "passingonly" is a boolean that will filter to only the instances passing all health checks.

This maps to the /v1/health/service API internally.

Here is an example configuration:

{

"type": "service",

"service": "redis",

"args": ["/usr/bin/my-service-handler.sh", "-redis"]

}

Or, using the watch command:

$ consul watch -type=service -service=redis /usr/bin/my-service-handler.sh

An example of the output of this command:

[

{

"Node": {

"Node": "foobar",

"Address": "10.1.10.12"

},

"Service": {

"ID": "redis",

"Service": "redis",

"Tags": **null**,

"Port": 8000

},

"Checks": [

{

"Node": "foobar",

"CheckID": "service:redis",

"Name": "Service 'redis' check",

"Status": "passing",

"Notes": "",

"Output": "",

"ServiceID": "redis",

"ServiceName": "redis"

},

{

"Node": "foobar",

"CheckID": "serfHealth",

"Name": "Serf Health Status",

"Status": "passing",

"Notes": "",

"Output": "",

"ServiceID": "",

"ServiceName": ""

}

]

}

]

### [»](https://www.consul.io/docs/agent/watches.html" \l "type-checks)Type: checks

The "checks" watch type is used to monitor the checks of a given service or those in a specific state. It optionally takes the "service" parameter to filter to a specific service or the "state" parameter to filter to a specific state. By default, it will watch all checks.

This maps to the /v1/health/state/ API if monitoring by state or /v1/health/checks/ if monitoring by service.

An example of the output of this command:

[

{

"Node": "foobar",

"CheckID": "service:redis",

"Name": "Service 'redis' check",

"Status": "passing",

"Notes": "",

"Output": "",

"ServiceID": "redis",

"ServiceName": "redis"

}

]

### [»](https://www.consul.io/docs/agent/watches.html" \l "type-event)Type: event

The "event" watch type is used to monitor for custom user events. These are fired using the [consul event](https://www.consul.io/docs/commands/event.html) command. It takes only a single optional "name" parameter which restricts the watch to only events with the given name.

This maps to the v1/event/list API internally.

Here is an example configuration:

{

"type": "event",

"name": "web-deploy",

"args": ["/usr/bin/my-service-handler.sh", "-web-deploy"]

}

Or, using the watch command:

$ consul watch -type=event -name=web-deploy /usr/bin/my-deploy-handler.sh -web-deploy

An example of the output of this command:

[

{

"ID": "f07f3fcc-4b7d-3a7c-6d1e-cf414039fcee",

"Name": "web-deploy",

"Payload": "MTYwOTAzMA==",

"NodeFilter": "",

"ServiceFilter": "",

"TagFilter": "",

"Version": 1,

"LTime": 18

},

...

]

To fire a new web-deploy event the following could be used:

$ consul event -name=web-deploy 1609030

# Connect

Consul Connect provides service-to-service connection authorization and encryption using mutual TLS. Applications can use [sidecar proxies](https://www.consul.io/docs/connect/proxies.html) to automatically establish TLS connections for inbound and outbound connections without being aware of Connect at all. Applications may also [natively integrate with Connect](https://www.consul.io/docs/connect/native.html) for optimal performance and security.

Connect enables deployment best-practices with service-to-service encryption everywhere and identity-based authorization. Rather than authorizing host-based access with IP address access rules, Connect uses the registered service identity to enforce access control with [intentions](https://www.consul.io/docs/connect/intentions.html). This makes it much easier to reason about access control and also enables services to freely move, such as in a scheduled environment with software such as Kubernetes or Nomad. Additionally, intention enforcement can be done regardless of the underlying network, so Connect works with physical networks, cloud networks, software-defined networks, cross-cloud, and more.

## [»](https://www.consul.io/docs/connect/index.html" \l "how-it-works)How it Works

The core of Connect is based on [mutual TLS](https://en.wikipedia.org/wiki/Mutual_authentication).

Connect provides each service with an identity encoded as a TLS certificate. This certificate is used to establish and accept connections to and from other services. The identity is encoded in the TLS certificate in compliance with the [SPIFFE X.509 Identity Document](https://github.com/spiffe/spiffe/blob/master/standards/X509-SVID.md). This enables Connect services to establish and accept connections with other SPIFFE-compliant systems.

The client service verifies the destination service certificate against the [public CA bundle](https://www.consul.io/api/connect/ca.html#list-ca-root-certificates). This is very similar to a typical HTTPS web browser connection. In addition to this, the client provides its own client certificate to show its identity to the destination service. If the connection handshake succeeds, the connection is encrypted and authorized.

The destination service verifies the client certificate against the [public CA bundle](https://www.consul.io/api/connect/ca.html#list-ca-root-certificates). After verifying the certificate, it must also call the [authorization API](https://www.consul.io/api/agent/connect.html#authorize) to authorize the connection against the configured set of Consul intentions. If the authorization API responds successfully, the connection is established. Otherwise, the connection is rejected.

To generate and distribute certificates, Consul has a built-in CA that requires no other dependencies, and also ships with built-in support for [Vault](https://www.consul.io/docs/connect/ca/vault.html). The PKI system is designed to be pluggable and can be extended to support any system by adding additional CA providers.

All APIs required for Connect typically respond in microseconds and impose minimal overhead to existing services. This is because the Connect-related APIs are all made to the local Consul agent over a loopback interface, and all [agent Connect endpoints](https://www.consul.io/api/agent/connect.html) implement local caching, background updating, and support blocking queries. As a result, most API calls operate on purely local in-memory data and can respond in microseconds.

## [»](https://www.consul.io/docs/connect/index.html" \l "getting-started-with-connect)Getting Started With Connect

There are several ways to try Connect in different environments.

* The [Connect introduction](https://learn.hashicorp.com/consul/getting-started/connect) in the Getting Started guide provides a simple walk through of getting two services to communicate via Connect using only Consul directly on your local machine.
* The [Envoy guide](https://www.consul.io/docs/guides/connect-envoy.html) walks through getting started with Envoy as a proxy, and uses Docker to run components locally without installing anything else.
* The [Kubernetes documentation](https://www.consul.io/docs/platform/k8s/run.html) shows how to get from an empty Kubernetes cluster to having Consul installed and Envoy configured to proxy application traffic automatically using the official helm chart.

## [»](https://www.consul.io/docs/connect/index.html" \l "agent-caching-and-performance)Agent Caching and Performance

To enable microsecond-speed responses on [agent Connect API endpoints](https://www.consul.io/api/agent/connect.html), the Consul agent locally caches most Connect-related data and sets up background [blocking queries](https://www.consul.io/api/index.html#blocking-queries) against the server to update the cache in the background. This allows most API calls such as retrieving certificates or authorizing connections to use in-memory data and respond very quickly.

All data cached locally by the agent is populated on demand. Therefore, if Connect is not used at all, the cache does not store any data. On first request, the data is loaded from the server and cached. The set of data cached is: public CA root certificates, leaf certificates, and intentions. For leaf certificates and intentions, only data related to the service requested is cached, not the full set of data.

Further, the cache is partitioned by ACL token and datacenters. This is done to minimize the complexity of the cache and prevent bugs where an ACL token may see data it shouldn't from the cache. This results in higher memory usage for cached data since it is duplicated per ACL token, but with the benefit of simplicity and security.

With Connect enabled, you'll likely see increased memory usage by the local Consul agent. The total memory is dependent on the number of intentions related to the services registered with the agent accepting Connect-based connections. The other data (leaf certificates and public CA certificates) is a relatively fixed size per service. In most cases, the overhead per service should be relatively small: single digit kilobytes at most.

The cache does not evict entries due to memory pressure. If memory capacity is reached, the process will attempt to swap. If swap is disabled, the Consul agent may begin failing and eventually crash. Cache entries do have TTLs associated with them and will evict their entries if they're not used. Given a long period of inactivity (3 days by default), the cache will empty itself.

## [»](https://www.consul.io/docs/connect/index.html" \l "multi-datacenter)Multi-Datacenter

Using Connect for service-to-service communications across multiple datacenters requires Consul Enterprise.

With Open Source Consul, Connect may be enabled on multiple Consul datacenters, but only services within the same datacenter can establish Connect-based, Authenticated and Authorized connections. In this version, Certificate Authority configurations and intentions are both local to their respective datacenters; they are not replicated across datacenters.

Full multi-datacenter support for Connect is available in [Consul Enterprise](https://www.consul.io/docs/enterprise/connect-multi-datacenter/index.html).

# Connect Configuration

There are many configuration options exposed for Connect. The only option that must be set is the "enabled" option on Consul Servers to enable Connect. All other configurations are optional and have reasonable defaults.

## [»](https://www.consul.io/docs/connect/configuration.html" \l "enable-connect-on-the-cluster)Enable Connect on the Cluster

The first step to use Connect is to enable Connect for your Consul cluster. By default, Connect is disabled. Enabling Connect requires changing the configuration of only your Consul servers (not client agents). To enable Connect, add the following to a new or existing [server configuration file](https://www.consul.io/docs/agent/options.html). In HCL:

connect {

enabled **=** **true**

}

This will enable Connect and configure your Consul cluster to use the built-in certificate authority for creating and managing certificates. You may also configure Consul to use an external [certificate management system](https://www.consul.io/docs/connect/ca.html), such as [Vault](https://vaultproject.io/).

No agent-wide configuration is necessary for non-server agents. Services and proxies may always register with Connect settings, but they will fail to retrieve or verify any TLS certificates. This causes all Connect-based connection attempts to fail until Connect is enabled on the server agents.

**Note:** Connect is enabled by default when running Consul in dev mode with consul agent -dev.

**Security note:** Enabling Connect is enough to try the feature but doesn't automatically ensure complete security. Please read the [Connect production guide](https://www.consul.io/docs/guides/connect-production.html) to understand the additional steps needed for a secure deployment.

## [»](https://www.consul.io/docs/connect/configuration.html" \l "built-in-proxy-options)Built-In Proxy Options

This is a complete example of all the configuration options available for the built-in proxy. Note that only the service.connect.proxy.config and service.connect.proxy.upsteams[].config maps are being described here, the rest of the service definition is shown for context but is [described elsewhere](https://www.consul.io/docs/connect/proxies.html#managed-proxies).

{

"service": {

...

"connect": {

"proxy": {

"config": {

"bind\_address": "0.0.0.0",

"bind\_port": 20000,

"tcp\_check\_address": "192.168.0.1",

"disable\_tcp\_check": **false**,

"local\_service\_address": "127.0.0.1:1234",

"local\_connect\_timeout\_ms": 1000,

"handshake\_timeout\_ms": 10000,

"upstreams": [...]

},

"upstreams": [

{

...

"config": {

"connect\_timeout\_ms": 1000

}

}

]

}

}

}

}

#### [»](https://www.consul.io/docs/connect/configuration.html" \l "proxy-config-key-reference)Proxy Config Key Reference

All fields are optional with a sane default.

* [bind\_address](https://www.consul.io/docs/connect/configuration.html" \l "bind_address) - The address the proxy will bind it's public mTLS listener to. It defaults to the same address the agent binds to.

* [bind\_port](https://www.consul.io/docs/connect/configuration.html" \l "bind_port) - The port the proxy will bind it's public mTLS listener to. If not provided, the agent will attempt to assign one from its [configured proxy port range](https://www.consul.io/docs/agent/options.html#proxy_min_port) if available. By default the range is [20000, 20255] and the port is selected at random from that range.

* [tcp\_check\_address](https://www.consul.io/docs/connect/configuration.html" \l "tcp_check_address) - The address the agent will run a [TCP health check](https://www.consul.io/docs/agent/checks.html) against. By default this is the same as the proxy's [bind address](https://www.consul.io/docs/connect/configuration.html#bind_address) except if the bind\_address is 0.0.0.0 or [::] in which case this defaults to 127.0.0.1 and assumes the agent can dial the proxy over loopback. For more complex configurations where agent and proxy communicate over a bridge for example, this configuration can be used to specify a different address (but not port) for the agent to use for health checks if it can't talk to the proxy over localhost or it's publicly advertised port. The check always uses the same port that the proxy is bound to.

* [disable\_tcp\_check](https://www.consul.io/docs/connect/configuration.html" \l "disable_tcp_check) - If true, this disables a TCP check being setup for the proxy. Default is false.

* [local\_service\_address](https://www.consul.io/docs/connect/configuration.html" \l "local_service_address) - The [address]:port that the proxy should use to connect to the local application instance. By default it assumes 127.0.0.1 as the address and takes the port from the service definition's port field. Note that allowing the application to listen on any non-loopback address may expose it externally and bypass Connect's access enforcement. It may be useful though to allow non-standard loopback addresses or where an alternative known-private IP is available for example when using internal networking between containers.

* [local\_connect\_timeout\_ms](https://www.consul.io/docs/connect/configuration.html" \l "local_connect_timeout_ms) - The number of milliseconds the proxy will wait to establish a connection to the local application before giving up. Defaults to 1000 or 1 second.

* [handshake\_timeout\_ms](https://www.consul.io/docs/connect/configuration.html" \l "handshake_timeout_ms) - The number of milliseconds the proxy will wait for incoming mTLS connections to complete the TLS handshake. Defaults to 10000 or 10 seconds.

* [upstreams](https://www.consul.io/docs/connect/configuration.html" \l "upstreams) - **Deprecated** Upstreams are now specified in the connect.proxy definition. Upstreams specified in the opaque config map here will continue to work for compatibility but it's strongly recommended that you move to using the higher level [upstream configuration](https://www.consul.io/docs/connect/proxies.html#upstream-configuration).

#### [»](https://www.consul.io/docs/connect/configuration.html" \l "proxy-upstream-config-key-reference)Proxy Upstream Config Key Reference

All fields are optional with a sane default.

* [connect\_timeout\_ms](https://www.consul.io/docs/connect/configuration.html" \l "connect_timeout_ms) - The number of milliseconds the proxy will wait to establish a TLS connection to the discovered upstream instance before giving up. Defaults to 10000 or 10 seconds.

# Connect Proxies

A Connect-aware proxy enables unmodified applications to use Connect. A per-service proxy sidecar transparently handles inbound and outbound service connections, automatically wrapping and verifying TLS connections.

When a proxy is used, the actual service being proxied should only accept connections on a loopback address. This requires all external connections to be established via the Connect protocol to provide authentication and authorization.

**Deprecation Note:** Managed Proxies are deprecated as of Consul 1.3. See [managed proxy deprecation](https://www.consul.io/docs/connect/proxies/managed-deprecated.html) for more information. It's strongly recommended to switch to one of the approaches listed on this page as soon as possible.

## [»](https://www.consul.io/docs/connect/proxies.html" \l "proxy-service-definitions)Proxy Service Definitions

Connect proxies are registered using regular [service definitions](https://www.consul.io/docs/agent/services.html). They can be registered both in config files or via the API just like any other service.

Additionally, to reduce the amount of boilerplate needed for a sidecar proxy, application service definitions may define inline [sidecar service registrations](https://www.consul.io/docs/connect/proxies/sidecar-service.html) which are an opinionated shorthand for a separate full proxy registration as described here.

To function as a Connect proxy, they must be declared as a proxy type and provide information about the service they represent.

To declare a service as a proxy, the service definition must contain the following fields:

* [kind](https://www.consul.io/docs/connect/proxies.html" \l "kind) (string) must be set to connect-proxy. This declares that the service is a proxy type.

* [proxy.destination\_service\_name](https://www.consul.io/docs/connect/proxies.html" \l "proxy-destination_service_name) (string) must be set to the service that this proxy is representing. Note that this replaces proxy\_destination in versions 1.2.0 to 1.3.0.
* [port](https://www.consul.io/docs/connect/proxies.html#port) (int) must be set so that other Connect services can discover the exact address for connections. address is optional if the service is being registered against an agent, since it'll inherit the node address.

Minimal Example:

{

"name": "redis-proxy",

"kind": "connect-proxy",

"proxy": {

"destination\_service\_name": "redis"

},

"port": 8181

}

With this service registered, any Connect clients searching for a Connect-capable endpoint for "redis" will find this proxy.

### [»](https://www.consul.io/docs/connect/proxies.html" \l "sidecar-proxy-fields)Sidecar Proxy Fields

Most Connect proxies are deployed as "sidecars" which means they are co-located with a single service instance which they represent and proxy all inbound traffic to. In this case the following fields should also be set:

* [proxy.destination\_service\_id](https://www.consul.io/docs/connect/proxies.html" \l "proxy-destination_service_id) (string: <required>) is set to the id (and not the name if they are different) of the specific service instance that is being proxied. The proxied service is assumed to be registered on the same agent although it's not strictly validated to allow for un-coordinated registrations.

* [proxy.local\_service\_port](https://www.consul.io/docs/connect/proxies.html" \l "proxy-local_service_port) (int: <required>) must specify the port the proxy should use to connect to the localservice instance.

* [proxy.local\_service\_address](https://www.consul.io/docs/connect/proxies.html" \l "proxy-local_service_address) (string: "") can be set to override the IP or hostname the proxy should use to connect to the local service. Defaults to 127.0.0.1.

### [»](https://www.consul.io/docs/connect/proxies.html" \l "complete-configuration-example)Complete Configuration Example

The following is a complete example showing all the options available when registering a proxy instance.

{

"name": "redis-proxy",

"kind": "connect-proxy",

"proxy": {

"destination\_service\_name": "redis",

"destination\_service\_id": "redis1",

"local\_service\_address": "127.0.0.1",

"local\_service\_port": 9090,

"config": {},

"upstreams": []

},

"port": 8181

}

**Deprecation Notice:** From version 1.2.0 to 1.3.0, proxy destination was specified using proxy\_destination at the top level. This will continue to work until at least 1.5.0 but it's highly recommended to switch to usingproxy.destination\_service\_name.

#### [»](https://www.consul.io/docs/connect/proxies.html" \l "proxy-parameters)Proxy Parameters

* [destination\_service\_name](https://www.consul.io/docs/connect/proxies.html" \l "destination_service_name) (string: <required>) - Specifies the name of the service this instance is proxying. Both side-car and centralized load-balancing proxies must specify this. It is used during service discovery to find the correct proxy instances to route to for a given service name.

* [destination\_service\_id](https://www.consul.io/docs/connect/proxies.html" \l "destination_service_id) (string: "") - Specifies the ID of a single specific service instance that this proxy is representing. This is only valid for side-car style proxies that run on the same node. It is assumed that the service instance is registered via the same Consul agent so the ID is unique and has no node qualifier. This is useful to show in tooling which proxy instance is a side-car for which application instance and will enable fine-grained analysis of the metrics coming from the proxy.
* [local\_service\_address](https://www.consul.io/docs/connect/proxies.html#local_service_address) (string: "") - Specifies the address a side-car proxy should attempt to connect to the local application instance on. Defaults to 127.0.0.1.

* [local\_service\_port](https://www.consul.io/docs/connect/proxies.html" \l "local_service_port) (int: <optional>) - Specifies the port a side-car proxy should attempt to connect to the local application instance on. Defaults to the port advertised by the service instance identified bydestination\_service\_id if it exists otherwise it may be empty in responses.

* [config](https://www.consul.io/docs/connect/proxies.html" \l "config) (object: {}) - Specifies opaque config JSON that will be stored and returned along with the service instance from future API calls.
* [upstreams](https://www.consul.io/docs/connect/proxies.html#upstreams) (array<Upstream>: []) - Specifies the upstream services this proxy should create listeners for. The format is defined in [Upstream Configuration Reference](https://www.consul.io/docs/connect/proxies.html#upstream-configuration-reference).

### [»](https://www.consul.io/docs/connect/proxies.html" \l "upstream-configuration-reference)Upstream Configuration Reference

The following examples show all possible upstream configuration parameters.

Note that in versions 1.2.0 to 1.3.0, managed proxy upstreams were specified inside the opaque connect.proxy.configmap. The format is almost unchanged however managed proxy upstreams are now defined a level up in theconnect.proxy.upstreams. The old location is deprecated and will be automatically converted into the new for an interim period before support is dropped in a future major release. The only difference in format between the upstream defintions is that the field destination\_datacenter has been renamed to datacenter to reflect that it's the discovery target and not necessarily the same as the instance that will be returned in the case of a prepared query that fails over to another datacenter.

Note that snake\_case is used here as it works in both [config file and API registrations](https://www.consul.io/docs/agent/services.html#service-definition-parameter-case).

Upstreams support multiple destination types. Both examples are shown below followed by documentation for each attribute.

#### [»](https://www.consul.io/docs/connect/proxies.html" \l "service-destination)Service Destination

{

"destination\_type": "service",

"destination\_name": "redis",

"datacenter": "dc1",

"local\_bind\_address": "127.0.0.1",

"local\_bind\_port": 1234,

"config": {}

},

#### [»](https://www.consul.io/docs/connect/proxies.html" \l "prepared-query-destination)Prepared Query Destination

{

"destination\_type": "prepared\_query",

"destination\_name": "database",

"local\_bind\_address": "127.0.0.1",

"local\_bind\_port": 1234,

"config": {}

},

* [destination\_name](https://www.consul.io/docs/connect/proxies.html" \l "destination_name) (string: <required>) - Specifies the name of the service or prepared query to route connect to. The prepared query should be the name or the ID of the prepared query.

* [local\_bind\_port](https://www.consul.io/docs/connect/proxies.html" \l "local_bind_port) (int: <required>) - Specifies the port to bind a local listener to for the application to make outbound connections to this upstream.

* [local\_bind\_address](https://www.consul.io/docs/connect/proxies.html" \l "local_bind_address) (string: "") - Specifies the address to bind a local listener to for the application to make outbound connections to this upstream. Defaults to 127.0.0.1.

* [destination\_type](https://www.consul.io/docs/connect/proxies.html" \l "destination_type) (string: "") - Speficied the type of discovery query to use to find an instance to connect to. Valid values are service or prepared\_query. Defaults to service.
* [datacenter](https://www.consul.io/docs/connect/proxies.html#datacenter) (string: "") - Specifies the datacenter to issue the discovery query too. Defaults to the local datacenter.

* [config](https://www.consul.io/docs/connect/proxies.html" \l "config-1) (object: {}) - Specifies opaque configuration options that will be provided to the proxy instance for this specific upstream. Can contain any valid JSON object. This might be used to configure proxy-specific features like timeouts or retries for the given upstream. See the [built-in proxy configuration reference](https://www.consul.io/docs/connect/configuration.html#built-in-proxy-options) for options available when using the built-in proxy. If using Envoy as a proxy, see [Envoy configuration reference](https://www.consul.io/docs/connect/configuration.html#envoy-options)

### [»](https://www.consul.io/docs/connect/proxies.html" \l "dynamic-upstreams)Dynamic Upstreams

If an application requires dynamic dependencies that are only available at runtime, it must currently [natively integrate](https://www.consul.io/docs/connect/native.html)with Connect. After natively integrating, the HTTP API or [DNS interface](https://www.consul.io/docs/agent/dns.html#connect-capable-service-lookups) can be used.

# Intentions

Intentions define access control for services via Connect and are used to control which services may establish connections. Intentions can be managed via the API, CLI, or UI.

Intentions are enforced by the [proxy](https://www.consul.io/docs/connect/proxies.html) or [natively integrated application](https://www.consul.io/docs/connect/native.html) on inbound connections. After verifying the TLS client certificate, the [authorize API endpoint](https://www.consul.io/docs/connect/intentions.html) is called which verifies the connection is allowed by testing the intentions. If authorize returns false the connection must be terminated.

The default intention behavior is defined by the default [ACL policy](https://www.consul.io/docs/guides/acl.html). If the default ACL policy is "allow all", then all Connect connections are allowed by default. If the default ACL policy is "deny all", then all Connect connections are denied by default.

## [»](https://www.consul.io/docs/connect/intentions.html" \l "intention-basics)Intention Basics

Intentions can be managed via the [API](https://www.consul.io/docs/connect/intentions.html), [CLI](https://www.consul.io/docs/connect/intentions.html), or UI. Please see the respective documentation for each for full details on options, flags, etc. Below is an example of a basic intention to show the basic attributes of an intention. The full data model of an intention can be found in the [API documentation](https://www.consul.io/docs/connect/intentions.html).

$ consul intention create -deny web db

Created: web => db (deny)

The intention above is a deny intention with a source of "web" and destination of "db". This says that connections from web to db are not allowed and the connection will be rejected.

When an intention is modified, existing connections will not be affected. This means that changing a connection from "allow" to "deny" today will not kill the connection. Addressing this shortcoming is on the near term roadmap for Consul.

### [»](https://www.consul.io/docs/connect/intentions.html" \l "wildcard-intentions)Wildcard Intentions

An intention source or destination may also be the special wildcard value \*. This matches any value and is used as a catch-all. Example:

$ consul intention create -deny web '\*'

Created: web => \* (deny)

This example says that the "web" service cannot connect to any service.

### [»](https://www.consul.io/docs/connect/intentions.html" \l "metadata)Metadata

Arbitrary string key/value data may be associated with intentions. This is unused by Consul but can be used by external systems or for visibility in the UI.

$ consul intention create \

-deny \

-meta description='Hello there' \

web db

...

$ consul intention get web db

Source: web

Destination: db

Action: deny

ID: 31449e02-c787-f7f4-aa92-72b5d9b0d9ec

Meta[description]: Hello there

Created At: Friday, 25-May-18 02:07:51 CEST

## [»](https://www.consul.io/docs/connect/intentions.html" \l "precedence-and-match-order)Precedence and Match Order

Intentions are matched in an implicit order based on specificity, preferring deny over allow. Specificity is determined by whether a value is an exact specified value or is the wildcard value \*. The full precedence table is shown below and is evaluated top to bottom, with larger numbers being evaluated first.

| **Source Name** | **Destination Name** | **Precedence** |
| --- | --- | --- |
| Exact | Exact | 9 |
| \* | Exact | 8 |
| Exact | \* | 6 |
| \* | \* | 5 |

The precedence value can be read from the [API](https://www.consul.io/api/connect/intentions.html) after an intention is created. Precedence cannot be manually overridden today. This is a feature that will be added in a later version of Consul.

In the case the two precedence values match, Consul will evaluate intentions based on lexographical ordering of the destination then source name. In practice, this is a moot point since authorizing a connection has an exact source and destination value so its impossible for two valid non-wildcard intentions to match.

The numbers in the table above are not stable. Their ordering will remain fixed but the actual number values may change in the future. The numbers are non-contiguous because there are some unused values in the middle in preparation for a future version of Consul supporting namespaces.

## [»](https://www.consul.io/docs/connect/intentions.html" \l "intention-management-permissions)Intention Management Permissions

Intention management can be protected by [ACLs](https://www.consul.io/docs/guides/acl.html). Permissions for intentions are destination-oriented, meaning the ACLs for managing intentions are looked up based on the destination value of the intention, not the source.

Intention permissions are by default implicitly granted at read level when granting service:read or service:write. This is because a service registered that wants to use Connect needs intentions:read for its own service name in order to know whether or not to authorize connections. The following ACL policy will implicitly grant intentions:read (note read) for service web.

service "web" {

policy **=** "write"

}

It is possible to explicitly specify intention permissions. For example, the following policy will allow a service to be discovered without granting access to read intentions for it.

service "web" {

policy **=** "read"

intentions **=** "deny"

}

Note that intentions:read is required for a token that a Connect-enabled service uses to register itself or it's proxy. If the token used does not have intentions:read then the agent will be unable to resolve intentions for the service and so will not be able to authorize any incoming connections.

**Security Note:** Explicitly allowing intentions:write on the token you provide to a service instance at registration time opens up a significant additional vulnerability. Although you may trust the service team to define which inbound connections they accept, using a combined token for registration allows a compromised instance to to redefine the intentions which allows many additional attack vectors and may be hard to detect. We strongly recommend only delegating intentions:write using tokens that are used by operations teams or orchestrators rather than spread via application config, or only manage intentions with management tokens.

## [»](https://www.consul.io/docs/connect/intentions.html" \l "performance-and-intention-updates)Performance and Intention Updates

The intentions for services registered with a Consul agent are cached locally on that agent. They are then updated via a background blocking query against the Consul servers.

Connect connection attempts require only local agent communication for authorization and generally impose only impose microseconds of latency to the connection. All actions in the data path of connections require only local data to ensure minimal performance overhead.

Updates to intentions are propagated nearly instantly to agents since agents maintain a continuous blocking query in the background for intention updates for registered services.

Because all the intention data is cached locally, the agents can fail static. Even if the agents are severed completely from the Consul servers, inbound connection authorization continues to work for a configured amount of time. Changes to intentions will not be picked up until the partition heals, but will then automatically take effect when connectivity is restored.

# Connect Certificate Management

Certificate management in Connect is done centrally through the Consul servers using the configured CA (Certificate Authority) provider. A CA provider manages root and intermediate certificates and performs certificate signing operations. The Consul leader orchestrates CA provider operations as necessary, such as when a service needs a new certificate or during CA rotation events.

The CA provider abstraction enables Consul to support multiple systems for storing and signing certificates. Consul ships with a [built-in CA](https://www.consul.io/docs/connect/ca/consul.html) which generates and stores the root certificate and private key on the Consul servers. Consul also has built-in support for [Vault as a CA](https://www.consul.io/docs/connect/ca/vault.html). With Vault, the root certificate and private key material remain with the Vault cluster. A future version of Consul will support pluggable CA systems using external binaries.

## [»](https://www.consul.io/docs/connect/ca.html" \l "ca-bootstrapping)CA Bootstrapping

CA initialization happens automatically when a new Consul leader is elected as long as [Connect is enabled](https://www.consul.io/docs/connect/configuration.html#enable-connect-on-the-cluster) and the CA system hasn't already been initialized. This initialization process will generate the initial root certificates and setup the internal Consul server state.

For the initial bootstrap, the CA provider can be configured through the [Agent configuration](https://www.consul.io/docs/agent/options.html#connect_ca_config). After initialization, the CA can only be updated through the [Update CA Configuration API endpoint](https://www.consul.io/api/connect/ca.html#update-ca-configuration). If a CA is already initialized, any changes to the CA configuration in the agent configuration file (including removing the configuration completely) will have no effect.

If no specific provider is configured when Connect is enabled, the built-in Consul CA provider will be used and a private key and root certificate will be generated automatically.

## [»](https://www.consul.io/docs/connect/ca.html" \l "viewing-root-certificates)Viewing Root Certificates

Root certificates can be queried with the [list CA Roots endpoint](https://www.consul.io/api/connect/ca.html#list-ca-root-certificates). With this endpoint, you can see the list of currently trusted root certificates. When a cluster first initializes, this will only list one trusted root. Multiple roots may appear as part of [rotation](https://www.consul.io/docs/connect/ca.html).

$ curl http://localhost:8500/v1/connect/ca/roots

**{**

"ActiveRootID": "31:6c:06:fb:49:94:42:d5:e4:55:cc:2e:27:b3:b2:2e:96:67:3e:7e",

"TrustDomain": "36cb52cd-4058-f811-0432-6798a240c5d3.consul",

"Roots": **[**

**{**

"ID": "31:6c:06:fb:49:94:42:d5:e4:55:cc:2e:27:b3:b2:2e:96:67:3e:7e",

"Name": "Consul CA Root Cert",

"SerialNumber": 7,

"SigningKeyID": "31:39:3a:34:35:3a:38:62:3a:33:30:3a:61:31:3a:34:35:3a:38:34:3a:61:65:3a:32:33:3a:35:32:3a:64:62:3a:38:64:3a:31:62:3a:66:66:3a:61:39:3a:30:39:3a:64:62:3a:66:63:3a:32:61:3a:37:32:3a:33:39:3a:61:65:3a:64:61:3a:31:31:3a:35:33:3a:66:34:3a:33:37:3a:35:63:3a:64:65:3a:64:31:3a:36:38:3a:64:38",

"NotBefore": "2018-06-06T17:35:25Z",

"NotAfter": "2028-06-03T17:35:25Z",

"RootCert": "-----BEGIN CERTIFICATE-----\nMIICmDCCAj6gAwIBAgIBBzAKBggqhkjOPQQDAjAWMRQwEgYDVQQDEwtDb25zdWwg\nQ0EgNzAeFw0xODA2MDYxNzM1MjVaFw0yODA2MDMxNzM1MjVaMBYxFDASBgNVBAMT\nC0NvbnN1bCBDQSA3MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEgo09lpx63bHw\ncSXeeoSpHpHgyzX1Q8ewJ3RUg6Ie8Howbs/QBz1y/kGxsF35HXij3YrqhgQyPPx4\nbQ8FH2YR4aOCAXswggF3MA4GA1UdDwEB/wQEAwIBhjAPBgNVHRMBAf8EBTADAQH/\nMGgGA1UdDgRhBF8xOTo0NTo4YjozMDphMTo0NTo4NDphZToyMzo1MjpkYjo4ZDox\nYjpmZjphOTowOTpkYjpmYzoyYTo3MjozOTphZTpkYToxMTo1MzpmNDozNzo1Yzpk\nZTpkMTo2ODpkODBqBgNVHSMEYzBhgF8xOTo0NTo4YjozMDphMTo0NTo4NDphZToy\nMzo1MjpkYjo4ZDoxYjpmZjphOTowOTpkYjpmYzoyYTo3MjozOTphZTpkYToxMTo1\nMzpmNDozNzo1YzpkZTpkMTo2ODpkODA/BgNVHREEODA2hjRzcGlmZmU6Ly8zNmNi\nNTJjZC00MDU4LWY4MTEtMDQzMi02Nzk4YTI0MGM1ZDMuY29uc3VsMD0GA1UdHgEB\n/wQzMDGgLzAtgiszNmNiNTJjZC00MDU4LWY4MTEtMDQzMi02Nzk4YTI0MGM1ZDMu\nY29uc3VsMAoGCCqGSM49BAMCA0gAMEUCIHl6UDdouw8Fzn/oDHputAxt3UFbVg/U\nvC6jWPuqqMwmAiEAkvMadtwjtNU7m/AQRJrj1LeG3eXw7dWO8SlI2fEs0yY=\n-----END CERTIFICATE-----\n",

"IntermediateCerts": null,

"Active": true,

"CreateIndex": 8,

"ModifyIndex": 8

**}**

**]**

**}**

## [»](https://www.consul.io/docs/connect/ca.html" \l "ca-configuration)CA Configuration

After initialization, the CA provider configuration can be viewed with the [Get CA Configuration API endpoint](https://www.consul.io/api/connect/ca.html#get-ca-configuration). Consul will filter sensitive values from this endpoint depending on the provider in use, so the configuration may not be complete.

$ curl http://localhost:8500/v1/connect/ca/configuration

**{**

"Provider": "consul",

"Config": **{**

"LeafCertTTL": "72h",

"RotationPeriod": "2160h"

**}**,

"CreateIndex": 5,

"ModifyIndex": 5

**}**

The CA provider can be reconfigured using the [Update CA Configuration API endpoint](https://www.consul.io/api/connect/ca.html#update-ca-configuration). Specific options for reconfiguration can be found in the specific CA provider documentation in the sidebar to the left.

## [»](https://www.consul.io/docs/connect/ca.html" \l "root-certificate-rotation)Root Certificate Rotation

Whenever the CA's configuration is updated in a way that causes the root key to change, a special rotation process will be triggered in order to smoothly transition to the new certificate. This rotation is automatically orchestrated by Consul.

This also automatically occurs when a completely different CA provider is configured (since this changes the root key). Therefore, this automatic rotation process can also be used to cleanly transition between CA providers. For example, updating Connect to use Vault instead of the built-in CA.

During rotation, an intermediate CA certificate is requested from the new root, which is then cross-signed by the old root. This cross-signed certificate is then distributed alongside any newly-generated leaf certificates used by the proxies once the new root becomes active, and provides a chain of trust back to the old root certificate in the event that a certificate signed by the new root is presented to a proxy that has not yet updated its bundle of trusted root CA certificates to include the new root.

After the cross-signed certificate has been successfully generated and the new root certificate or CA provider has been set up, the new root becomes the active one and is immediately used for signing any new incoming certificate requests.

If we check the [list CA roots endpoint](https://www.consul.io/api/connect/ca.html#list-ca-root-certificates) after updating the configuration with a new root certificate, we can see both the old and new root certificates are present, and the currently active root has an intermediate certificate which has been generated and cross-signed automatically by the old root during the rotation process:

$ curl localhost:8500/v1/connect/ca/roots

**{**

"ActiveRootID": "d2:2c:41:94:1e:50:04:ea:86:fc:08:d6:b0:45:a4:af:8a:eb:76:a0",

"TrustDomain": "36cb52cd-4058-f811-0432-6798a240c5d3.consul",

"Roots": **[**

**{**

"ID": "31:6c:06:fb:49:94:42:d5:e4:55:cc:2e:27:b3:b2:2e:96:67:3e:7e",

"Name": "Consul CA Root Cert",

"SerialNumber": 7,

"SigningKeyID": "31:39:3a:34:35:3a:38:62:3a:33:30:3a:61:31:3a:34:35:3a:38:34:3a:61:65:3a:32:33:3a:35:32:3a:64:62:3a:38:64:3a:31:62:3a:66:66:3a:61:39:3a:30:39:3a:64:62:3a:66:63:3a:32:61:3a:37:32:3a:33:39:3a:61:65:3a:64:61:3a:31:31:3a:35:33:3a:66:34:3a:33:37:3a:35:63:3a:64:65:3a:64:31:3a:36:38:3a:64:38",

"NotBefore": "2018-06-06T17:35:25Z",

"NotAfter": "2028-06-03T17:35:25Z",

"RootCert": "-----BEGIN CERTIFICATE-----\nMIICmDCCAj6gAwIBAgIBBzAKBggqhkjOPQQDAjAWMRQwEgYDVQQDEwtDb25zdWwg\nQ0EgNzAeFw0xODA2MDYxNzM1MjVaFw0yODA2MDMxNzM1MjVaMBYxFDASBgNVBAMT\nC0NvbnN1bCBDQSA3MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEgo09lpx63bHw\ncSXeeoSpHpHgyzX1Q8ewJ3RUg6Ie8Howbs/QBz1y/kGxsF35HXij3YrqhgQyPPx4\nbQ8FH2YR4aOCAXswggF3MA4GA1UdDwEB/wQEAwIBhjAPBgNVHRMBAf8EBTADAQH/\nMGgGA1UdDgRhBF8xOTo0NTo4YjozMDphMTo0NTo4NDphZToyMzo1MjpkYjo4ZDox\nYjpmZjphOTowOTpkYjpmYzoyYTo3MjozOTphZTpkYToxMTo1MzpmNDozNzo1Yzpk\nZTpkMTo2ODpkODBqBgNVHSMEYzBhgF8xOTo0NTo4YjozMDphMTo0NTo4NDphZToy\nMzo1MjpkYjo4ZDoxYjpmZjphOTowOTpkYjpmYzoyYTo3MjozOTphZTpkYToxMTo1\nMzpmNDozNzo1YzpkZTpkMTo2ODpkODA/BgNVHREEODA2hjRzcGlmZmU6Ly8zNmNi\nNTJjZC00MDU4LWY4MTEtMDQzMi02Nzk4YTI0MGM1ZDMuY29uc3VsMD0GA1UdHgEB\n/wQzMDGgLzAtgiszNmNiNTJjZC00MDU4LWY4MTEtMDQzMi02Nzk4YTI0MGM1ZDMu\nY29uc3VsMAoGCCqGSM49BAMCA0gAMEUCIHl6UDdouw8Fzn/oDHputAxt3UFbVg/U\nvC6jWPuqqMwmAiEAkvMadtwjtNU7m/AQRJrj1LeG3eXw7dWO8SlI2fEs0yY=\n-----END CERTIFICATE-----\n",

"IntermediateCerts": null,

"Active": false,

"CreateIndex": 8,

"ModifyIndex": 24

**}**,

**{**

"ID": "d2:2c:41:94:1e:50:04:ea:86:fc:08:d6:b0:45:a4:af:8a:eb:76:a0",

"Name": "Consul CA Root Cert",

"SerialNumber": 16238269036752183483,

"SigningKeyID": "",

"NotBefore": "2018-06-06T17:37:03Z",

"NotAfter": "2028-06-03T17:37:03Z",

"RootCert": "-----BEGIN CERTIFICATE-----\nMIIDijCCAnKgAwIBAgIJAOFZ66em1qC7MA0GCSqGSIb3DQEBCwUAMGIxCzAJBgNV\nBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRYwFAYDVQQHDA1TYW4gRnJhbmNp\nc2NvMRIwEAYDVQQKDAlIYXNoaUNvcnAxEjAQBgNVBAMMCWxvY2FsaG9zdDAeFw0x\nODA2MDYxNzM3MDNaFw0yODA2MDMxNzM3MDNaMGIxCzAJBgNVBAYTAlVTMRMwEQYD\nVQQIDApDYWxpZm9ybmlhMRYwFAYDVQQHDA1TYW4gRnJhbmNpc2NvMRIwEAYDVQQK\nDAlIYXNoaUNvcnAxEjAQBgNVBAMMCWxvY2FsaG9zdDCCASIwDQYJKoZIhvcNAQEB\nBQADggEPADCCAQoCggEBAK6ostXN6W093EpI3RDNQDwC1Gq3lPNoodL5XRaVVIBU\n3X5iC+Ttk02p67cHUguh4ZrWr3o3Dzxm+gKK0lfZLW0nNYNPAIGZWQD9zVSx1Lqt\n8X0pd+fhMV5coQrh3YIG/vy17IBTSBuRUX0mXOKjOeJJlrw1HQZ8pfm7WX6LFul2\nXszvgn5K1XR+9nhPy6K2bv99qsY0sm7AqCS2BjYBW8QmNngJOdLPdhyFh7invyXe\nPqgujc/KoA3P6e3/G7bJZ9+qoQMK8uwD7PxtA2hdQ9t0JGPsyWgzhwfBxWdBWRzV\nRvVi6Yu2tvw3QrjdeKQ5Ouw9FUb46VnTU7jTO974HjkCAwEAAaNDMEEwPwYDVR0R\nBDgwNoY0c3BpZmZlOi8vMzZjYjUyY2QtNDA1OC1mODExLTA0MzItNjc5OGEyNDBj\nNWQzLmNvbnN1bDANBgkqhkiG9w0BAQsFAAOCAQEATHgCro9VXj7JbH/tlB6f/KWf\n7r98+rlUE684ZRW9XcA9uUA6y265VPnemsC/EykPsririoh8My1jVPuEfgMksR39\n9eMDJKfutvSpLD1uQqZE8hu/hcYyrmQTFKjW71CfGIl/FKiAg7wXEw2ljLN9bxNv\nGG118wrJyMZrRvFjC2QKY025QQSJ6joNLFMpftsZrJlELtRV+nx3gMabpiDRXhIw\nJM6ti26P1PyVgGRPCOG10v+OuUtwe0IZoOqWpPJN8jzSuqZWf99uolkG0xuqLNz6\nd8qvTp1YF9tTmysgvdeGALez/02HTF035RVTsQfH9tM/+4yG1UnmjLpz3p4Fow==\n-----END CERTIFICATE-----",

"IntermediateCerts": **[**

"-----BEGIN CERTIFICATE-----\nMIIDTzCCAvWgAwIBAgIBFzAKBggqhkjOPQQDAjAWMRQwEgYDVQQDEwtDb25zdWwg\nQ0EgNzAeFw0xODA2MDYxNzM3MDNaFw0yODA2MDMxNzM3MDNaMGIxCzAJBgNVBAYT\nAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRYwFAYDVQQHDA1TYW4gRnJhbmNpc2Nv\nMRIwEAYDVQQKDAlIYXNoaUNvcnAxEjAQBgNVBAMMCWxvY2FsaG9zdDCCASIwDQYJ\nKoZIhvcNAQEBBQADggEPADCCAQoCggEBAK6ostXN6W093EpI3RDNQDwC1Gq3lPNo\nodL5XRaVVIBU3X5iC+Ttk02p67cHUguh4ZrWr3o3Dzxm+gKK0lfZLW0nNYNPAIGZ\nWQD9zVSx1Lqt8X0pd+fhMV5coQrh3YIG/vy17IBTSBuRUX0mXOKjOeJJlrw1HQZ8\npfm7WX6LFul2Xszvgn5K1XR+9nhPy6K2bv99qsY0sm7AqCS2BjYBW8QmNngJOdLP\ndhyFh7invyXePqgujc/KoA3P6e3/G7bJZ9+qoQMK8uwD7PxtA2hdQ9t0JGPsyWgz\nhwfBxWdBWRzVRvVi6Yu2tvw3QrjdeKQ5Ouw9FUb46VnTU7jTO974HjkCAwEAAaOC\nARswggEXMGgGA1UdDgRhBF8xOTo0NTo4YjozMDphMTo0NTo4NDphZToyMzo1Mjpk\nYjo4ZDoxYjpmZjphOTowOTpkYjpmYzoyYTo3MjozOTphZTpkYToxMTo1MzpmNDoz\nNzo1YzpkZTpkMTo2ODpkODBqBgNVHSMEYzBhgF8xOTo0NTo4YjozMDphMTo0NTo4\nNDphZToyMzo1MjpkYjo4ZDoxYjpmZjphOTowOTpkYjpmYzoyYTo3MjozOTphZTpk\nYToxMTo1MzpmNDozNzo1YzpkZTpkMTo2ODpkODA/BgNVHREEODA2hjRzcGlmZmU6\nLy8zNmNiNTJjZC00MDU4LWY4MTEtMDQzMi02Nzk4YTI0MGM1ZDMuY29uc3VsMAoG\nCCqGSM49BAMCA0gAMEUCIBp46tRDot7GFyDXu7egq7lXBvn+UUHD5MmlFvdWmtnm\nAiEAwKBzEMcLd5kCBgFHNGyksRAMh/AGdEW859aL6z0u4gM=\n-----END CERTIFICATE-----\n"

**]**,

"Active": true,

"CreateIndex": 24,

"ModifyIndex": 24

**}**

**]**

**}**

The old root certificate will be automatically removed once enough time has elapsed for any leaf certificates signed by it to expire.

# Connect-Native App Integration

Applications can natively integrate with the Connect API to support accepting and establishing connections to other Connect services without the overhead of a [proxy sidecar](https://www.consul.io/docs/connect/proxies.html). This option is especially useful for applications that may be experiencing performance issues with the proxy sidecar deployment. This page will cover the high-level overview of integration, registering the service, etc. For language-specific examples, see the sidebar navigation to the left.

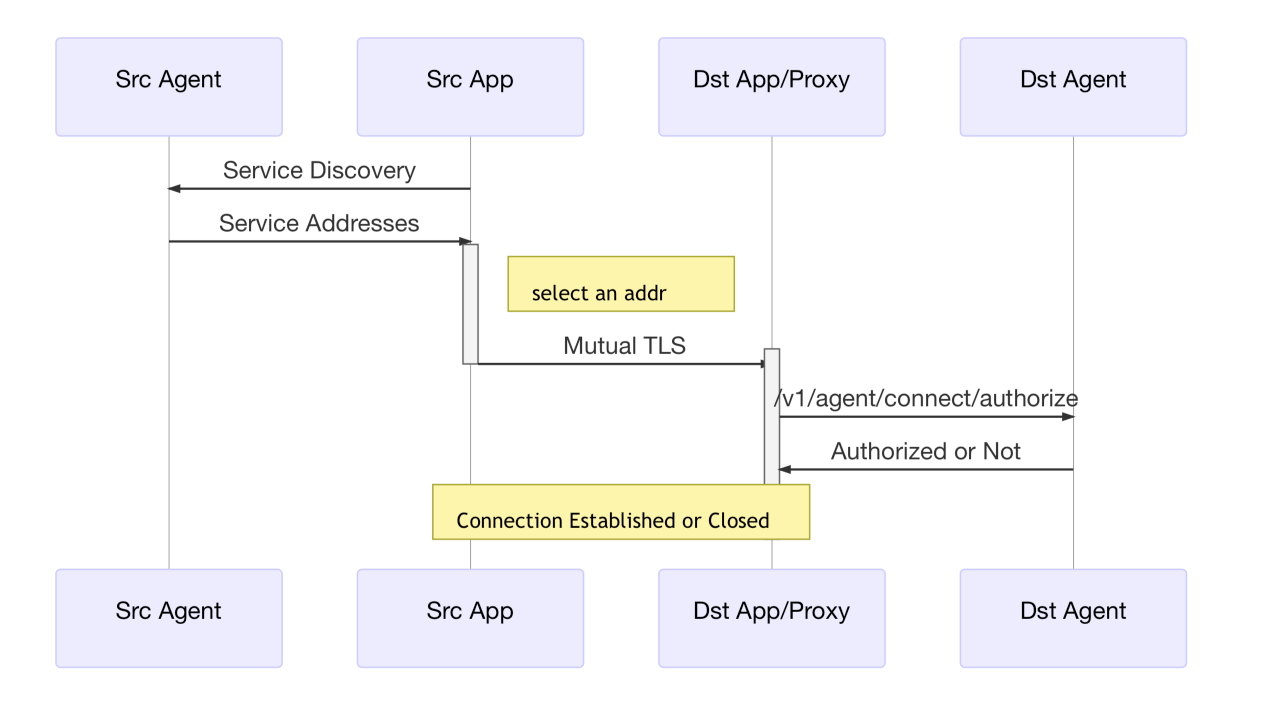
Connect is just basic mutual TLS. This means that almost any application can easily integrate with Connect. There is no custom protocol in use; any language that supports TLS can accept and establish Connect-based connections.

We currently provide an easy-to-use [Go integration](https://www.consul.io/docs/connect/native/go.html) to assist with the getting the proper certificates, verifying connections, etc. We plan to add helper libraries for other languages in the future. However, without library support, it is still possible for any major language to integrate with Connect.

## [»](https://www.consul.io/docs/connect/native.html" \l "overview)Overview

The primary work involved in natively integrating with Connect is [acquiring the proper TLS certificate](https://www.consul.io/api/agent/connect.html#service-leaf-certificate), [verifying TLS certificates](https://www.consul.io/api/agent/connect.html#certificate-authority-ca-roots), and [authorizing inbound connections](https://www.consul.io/api/agent/connect.html#authorize). All of this is done using the Consul HTTP APIs linked above.

An overview of the sequence is shown below. The diagram and the following details may seem complex, but this is a regular mutual TLS connection with an API call to verify the incoming client certificate.



Details on the steps are below:

* **Service discovery** - This is normal service discovery using Consul, a static IP, or any other mechanism. If you're using Consul DNS, the [<service>.connect](https://www.consul.io/docs/agent/dns.html#connect-capable-service-lookups) syntax to find Connect-capable endpoints for a service. After service discovery, choose one address from the list of **service addresses**.
* **Mutual TLS** - As a client, connect to the discovered service address over normal TLS. As part of the TLS connection, provide the [service certificate](https://www.consul.io/api/agent/connect.html#service-leaf-certificate) as the client certificate. Verify the remote certificate against the [public CA roots](https://www.consul.io/api/agent/connect.html#certificate-authority-ca-roots). As a client, if the connection is established then you've established a Connect-based connection and there are no further steps!
* **Authorization** - As a server accepting connections, verify the client certificate against the [public CA roots](https://www.consul.io/api/agent/connect.html#certificate-authority-ca-roots). After verifying the certificate, parse some basic fields from it and call the [authorizing API](https://www.consul.io/api/agent/connect.html#authorize) against the local agent. If this returns successfully, complete the TLS handshake and establish the connection. If authorization fails, close the connection.

**A note on performance:** The only API call in the connection path is the [authorization API](https://www.consul.io/api/agent/connect.html#authorize). The other API calls to acquire the leaf certificate and CA roots are expected to be done out of band and reused. The authorize API call should be called against the local Consul agent. The agent uses locally cached data to authorize the connection and typically responds in microseconds. Therefore, the impact to the TLS handshake is typically microseconds.

## [»](https://www.consul.io/docs/connect/native.html" \l "updating-certificates-and-certificate-roots)Updating Certificates and Certificate Roots

The leaf certificate and CA roots can be updated at any time and the natively integrated application must react to this relatively quickly so that new connections are not disrupted. This can be done through Consul blocking queries (HTTP long polling) or through periodic polling.

The API calls for [acquiring a leaf TLS certificate](https://www.consul.io/api/agent/connect.html#service-leaf-certificate) and [reading CA roots](https://www.consul.io/api/agent/connect.html#certificate-authority-ca-roots) both support [blocking queries](https://www.consul.io/api/index.html#blocking-queries). By using blocking queries, an application can efficiently wait for an updated value. For example, the leaf certificate API will block until the certificate is near expiration or the signing certificates have changed and will issue and return a new certificate.

In some languages, using blocking queries may not be simple. In that case, we still recommend using the blocking query parameters but with a very short timeout value set. Doing this is documented with [blocking queries](https://www.consul.io/api/index.html#blocking-queries). The low timeout will ensure the API responds quickly. We recommend that applications poll the certificate endpoints frequently, such as multiple times per minute.

The overhead for the blocking queries (long or periodic polling) is minimal. The API calls are to the local agent and the local agent uses locally cached data multiplexed over a single TCP connection to the Consul leader. Even if a single machine has 1,000 Connect-enabled services all blocking on certificate updates, this translates to only one TCP connection to the Consul server.

Some language libraries such as the [Go library](https://www.consul.io/docs/connect/native/go.html) automatically handle updating and locally caching the certificates.

## [»](https://www.consul.io/docs/connect/native.html" \l "service-registration)Service Registration

Connect-native applications must tell Consul that they support Connect natively. This enables the service to be returned as part of service discovery for Connect-capable services, used by other Connect-native applications and client [proxies](https://www.consul.io/docs/connect/proxies.html).

This can be specified directly in the [service definition](https://www.consul.io/docs/agent/services.html):

{

"service": {

"name": "redis",

"port": 8000,

"connect": {

"native": **true**

}

}

}

Services that support Connect natively are still returned through the standard service discovery mechanisms in addition to the Connect-only service discovery mechanisms.

# Developing and Debugging Connect Services

It is often necessary to connect to a service for development or debugging. If a service only exposes a Connect listener, then we need a way to establish a mutual TLS connection to the service. The [consul connect proxy command](https://www.consul.io/docs/commands/connect/proxy.html) can be used for this task on any machine with access to a Consul agent (local or remote).

Restricting access to services only via Connect ensures that the only way to connect to a service is through valid authorization of the [intentions](https://www.consul.io/docs/connect/intentions.html). This can extend to developers and operators, too.

## [»](https://www.consul.io/docs/connect/dev.html" \l "connecting-to-connect-only-services)Connecting to Connect-only Services

As an example, let's assume that we have a PostgreSQL database running that we want to connect to via psql, but the only non-loopback listener is via Connect. Let's also assume that we have an ACL token to identify as operator-mitchellh. We can start a local proxy:

$ consul connect proxy \

-service operator-mitchellh \

-upstream postgresql:8181

This works because the source -service does not need to be registered in the local Consul catalog. However, to retrieve a valid identifying certificate, the ACL token must have service:write permissions. This can be used as a sort of "virtual service" to represent people, too. In the example above, the proxy is identifying as operator-mitchellh.

With the proxy running, we can now use psql like normal:

$ psql -h 127.0.0.1 -p 8181 -U mitchellh mydb

>

This psql session is now happening through our local proxy via an authorized mutual TLS connection to the PostgreSQL service in our Consul catalog.

### [»](https://www.consul.io/docs/connect/dev.html" \l "masquerading-as-a-service)Masquerading as a Service

You can also easily masquerade as any source service by setting the -service value to any service. Note that the proper ACL permissions are required to perform this task.

For example, if you have an ACL token that allows service:write for web and you want to connect to the postgresqlservice as "web", you can start a proxy like so:

$ consul connect proxy \

-service web \

-upstream postgresql:8181

# Connect on Nomad

Connect can be used with [Nomad](https://www.nomadproject.io/) to provide secure service-to-service communication between Nomad jobs and task groups. The ability to use the [dynamic port](https://www.nomadproject.io/docs/job-specification/network.html#dynamic-ports) feature of Nomad makes Connect particularly easy to use.

Using Connect with Nomad today requires manually specifying the Connect sidecar proxy and managing intentions directly via Consul (outside of Nomad). The Consul and Nomad teams are working together towards a more automatic and unified solution in an upcoming Nomad release.

**Important security note:** As of Nomad 0.8.4, Nomad doesn't yet support network namespacing for tasks in a task group. As a result, running Connect with Nomad should assume the same [security checklist](https://www.consul.io/docs/connect/security.html#prevent-non-connect-traffic-to-services) as running directly on a machine without namespacing.

**Envoy Support:** Using the [Envoy Connect integration](https://www.consul.io/docs/connect/proxies/envoy.html) with Nomad is not currently possible following this guide. This guide relies on some features of the built-in proxy that are not possible via Envoy.

## [»](https://www.consul.io/docs/connect/platform/nomad.html" \l "requirements)Requirements

To use Connect with Nomad, the following requirements must be first be satisfied:

* **Nomad 0.8.3 or later.** - The server and clients of the Nomad cluster must be running version 0.8.3 or later. This is the earliest version that was verified to work with Connect. It is possible to work with earlier versions but it is untested.
* **Consul 1.2.0 or later.** - A Consul cluster must be setup and running with version 1.2.0 or later. Nomad must be [configured to use this Consul cluster](https://www.nomadproject.io/docs/service-discovery/index.html).

## [»](https://www.consul.io/docs/connect/platform/nomad.html" \l "accepting-connect-for-an-existing-service)Accepting Connect for an Existing Service

The job specification below shows a job that is configured with Connect. The example uses raw\_exec for now just to show how it can be used locally but the Docker driver or any other driver could easily be used. The example will be updated to use the official Consul Docker image following release.

The example below shows a hypothetical database being configured to listen with Connect only. Explanations of the various sections follow the example.

job "db" {

datacenters **=** ["dc1"]

group "db" {

task "db" {

driver **=** "raw\_exec"

config {

command **=** "/usr/local/bin/my-database"

args **=** ["-bind", "127.0.0.1:${NOMAD\_PORT\_tcp}"]

}

resources {

network {

port "tcp" {}

}

}

}

task "connect-proxy" {

driver **=** "raw\_exec"

config {

command **=** "/usr/local/bin/consul"

args **=** [

"connect", "proxy",

"-service", "db",

"-service-addr", "${NOMAD\_ADDR\_db\_tcp}",

"-listen", ":${NOMAD\_PORT\_tcp}",

"-register",

]

}

resources {

network {

port "tcp" {}

}

}

}

}

}

The job specification contains a single task group "db" with two tasks. By placing the two tasks in the same group, the Connect proxy will always be colocated directly next to the database, and has access to information such as the dynamic port it is running on.

For the "db" task, there are a few important configurations:

* The -bind address for the database is loopback only and listening on a dynamic port. This prevents non-Connect connections from outside of the node that the database is running on.
* The tcp port is dynamic. This removes any static constraints on the port, allowing Nomad to allocate any available port for any allocation.
* The database is not registered with Consul using a service block. This isn't strictly necessary, but since we won't be connecting directly to this service, we also don't need to register it. We recommend registering the source service as well since Consul can then know the health of the target service, which is used in determining if the proxy should receive requests.

Next, the "connect-proxy" task is colocated next to the "db" task. This is using "raw\_exec" executing Consul directly. In the future this example will be updated to use the official Consul Docker image.

The important configuration for this proxy:

* The -service and -service-addr flag specify the name of the service the proxy is representing. The address is set to the interpolation ${NOMAD\_ADDR\_db\_tcp} which allows the database to listen on any dynamic address and the proxy can still find it.
* The -listen flag sets up a public listener (TLS) to accept connections on behalf of the "db" service. The port this is listening on is dynamic, since service discovery can be used to find the service ports.
* The -register flag tells the proxy to self-register with Consul. Nomad doesn't currently know how to register Connect proxies with the service stanza, and this causes the proxy to register itself so it is discoverable.

Following running this job specification, the DB will be started with a Connect proxy. The only public listener from the job is the proxy. This means that only Connect connections can access the database from an external node.

## [»](https://www.consul.io/docs/connect/platform/nomad.html" \l "connecting-to-upstream-dependencies)Connecting to Upstream Dependencies

In addition to accepting Connect-based connections, services often need to connect to upstream dependencies that are listening via Connect. For example, a "web" application may need to connect to the "db" exposed in the example above.

The job specification below shows an example of this scenario:

job "web" {

datacenters **=** ["dc1"]

group "web" {

task "web" {

# ... typical configuration.

env {

DATABASE\_URL **=** "postgresql://${NOMAD\_ADDR\_proxy\_tcp}/db"

}

}

task "proxy" {

driver **=** "raw\_exec"

config {

command **=** "/usr/local/bin/consul"

args **=** [

"connect", "proxy",

"-service", "web",

"-upstream", "db:${NOMAD\_PORT\_tcp}",

]

}

resources {

network {

port "tcp" {}

}

}

}

}

}

Starting with the "proxy" task, the primary difference to accepting connections is that the service address, -listen, and -register flag are not specified. This prevents the proxy from registering itself as a valid listener for the given service.

The -upstream flag is specified to configure a private listener to connect to the service "db" as "web". The port is dynamic. The listener will bind to a loopback address only.

Finally, the "web" task is configured to use the localhost address to connect to the database. This will establish a connection to the remote DB using Connect. Interpolation is used to retrieve the address dynamically since the port is dynamic.

**Both -listen and -upstream can be specified** for services that both accept Connect connections as well as have upstream dependencies. Additionally, multiple -upstream flags can be specified for multiple upstream dependencies. This can be done on a single proxy instance rather than having multiple.

# Connect Sidecar on Kubernetes

[Connect](https://www.consul.io/docs/connect/index.html) is a feature built into to Consul that enables automatic service-to-service authorization and connection encryption across your Consul services. Connect can be used with Kubernetes to secure pod communication with other pods and external Kubernetes services.

The Connect sidecar running Envoy can be automatically injected into pods in your cluster, making configuration for Kubernetes automatic. This functionality is provided by the [consul-k8s project](https://github.com/hashicorp/consul-k8s) and can be automatically installed and configured using the [Consul Helm chart](https://www.consul.io/docs/platform/k8s/helm.html).

## [»](https://www.consul.io/docs/platform/k8s/connect.html#usage)Usage

When the [Connect injector is installed](https://www.consul.io/docs/platform/k8s/connect.html#installation-and-configuration), the Connect sidecar is automatically added to all pods. This sidecar can both accept and establish connections using Connect, enabling the pod to communicate to clients and dependencies exclusively over authorized and encrypted connections.

**Note:** The pod specifications in this section are valid and use publicly available images. If you've installed the Connect injector, feel free to run the pod specifications in this section to try Connect with Kubernetes. Please note the documentation below this section on how to properly install and configure the Connect injector.

### [»](https://www.consul.io/docs/platform/k8s/connect.html" \l "accepting-inbound-connections)Accepting Inbound Connections

An example pod is shown below with Connect enabled to accept inbound connections. Notice that the pod would still be fully functional without Connect. Minimal to zero modifications are required to pod specifications to enable Connect in Kubernetes.

This pod specification starts a server that responds to any HTTP request with the static text "hello world".

apiVersion: v1

kind: Pod

metadata:

name: static-server

annotations:

"consul.hashicorp.com/connect-inject": "true"

spec:

containers:

- name: static-server

image: hashicorp/http-echo:latest

args:

- -text="hello world"

- -listen=:8080

ports:

- containerPort: 8080

name: http

The only change for Connect is the addition of the consul.hashicorp.com/connect-inject annotation. This enables injection for this pod. The injector can also be [configured](https://www.consul.io/docs/platform/k8s/connect.html#installation-and-configuration) to automatically inject unless explicitly disabled, but the default installation requires opt-in using the annotation shown above.

This will start a Connect sidecar that listens on a random port registered with Consul and proxies valid inbound connections to port 8080 in the pod. To establish a connection to the pod using Connect, a client must use another Connect proxy. The client Connect proxy will use Consul service discovery to find all available upstream proxies and their public ports.

In the example above, the server is listening on :8080. This means the server will still bind to the pod IP and allow external connections. This is useful to transition to Connect by allowing both Connect and non-Connect connections. To restrict access to only Connect-authorized clients, any listeners should bind to localhost only (such as 127.0.0.1).

### [»](https://www.consul.io/docs/platform/k8s/connect.html" \l "connecting-to-connect-enabled-services)Connecting to Connect-Enabled Services

The example pod specification below configures a pod that is capable of establishing connections to our previous example "static-server" service. The connection to this static text service happens over an authorized and encrypted connection via Connect.

apiVersion: v1

kind: Pod

metadata:

name: static-client

annotations:

"consul.hashicorp.com/connect-inject": "true"

"consul.hashicorp.com/connect-service-upstreams": "static-server:1234"

spec:

containers:

- name: static-client

image: tutum/curl:latest

# Just spin & wait forever, we'll use `kubectl exec` to demo

command: [ "/bin/sh", "-c", "--" ]

args: [ "while true; do sleep 30; done;" ]

Pods must specify upstream dependencies with the [consul.hashicorp.com/connect-service-upstreams annotation](https://www.consul.io/docs/platform/k8s/connect.html#consul-hashicorp-com-connect-service-upstreams). This annotation declares the names of any upstream dependencies and a local port for the proxy to listen on. When a connection is established to that local port, the proxy establishes a connection to the target service ("static-server" in this example) using mutual TLS and identifying as the source service ("static-client" in this example).

The injector will also set environment variables <NAME>\_CONNECT\_SERVICE\_HOST and <NAME>\_CONNECT\_SERVICE\_PORT in every container in the pod for every defined upstream. This is analogous to the standard Kubernetes service environment variables, but point instead to the correct local proxy port to establish connections via Connect.

Any containers running in the pod that need to establish connections to dependencies must be reconfigured to use the local upstream address either directly or using the environment variables set by the injector (defined above). This means pods should not use Kubernetes service DNS or environment variables for these connections.

We can verify access to the static text server using kubectl exec. Notice that we use the local address and port from the upstream annotation (1234) for this verification.

$ kubectl exec static-client -- curl -s http://127.0.0.1:1234/

"hello world"

We can control access to the server using [intentions](https://www.consul.io/docs/connect/intentions.html). If you use the Consul UI or [CLI](https://www.consul.io/docs/commands/intention/create.html) to create a deny [intention](https://www.consul.io/docs/connect/intentions.html) between "static-client" and "static-server", connections are immediately rejected without updating either of the running pods. You can then remove this intention to allow connections again.

$ kubectl exec static-client -- curl -s http://127.0.0.1:1234/

command terminated with exit code 52

### [»](https://www.consul.io/docs/platform/k8s/connect.html" \l "available-annotations)Available Annotations

Annotations can be used to configure the injection behavior.

* [consul.hashicorp.com/connect-inject](https://www.consul.io/docs/platform/k8s/connect.html" \l "consul-hashicorp-com-connect-inject) - If this is "true" then injection is enabled. If this is "false" then injection is explicitly disabled. The default injector behavior requires pods to opt-in to injection by specifying this value as "true". This default can be changed in the injector's configuration if desired.

* [consul.hashicorp.com/connect-service](https://www.consul.io/docs/platform/k8s/connect.html" \l "consul-hashicorp-com-connect-service) - For pods that accept inbound connections, this specifies the name of the service that is being served. This defaults to the name of the first container in the pod.

* [consul.hashicorp.com/connect-service-port](https://www.consul.io/docs/platform/k8s/connect.html" \l "consul-hashicorp-com-connect-service-port) - For pods that accept inbound connections, this specifies the port to route inbound connections to. This is the port that the service is listening on. The service port defaults to the first exposed port on any container in the pod. If specified, the value can be the name of a configured port, such as "http" or it can be a direct port value such as "8080". This is the port of the service, the proxy public listener will listen on a dynamic port.

* [consul.hashicorp.com/connect-service-upstreams](https://www.consul.io/docs/platform/k8s/connect.html" \l "consul-hashicorp-com-connect-service-upstreams) - The list of upstream services that this pod needs to connect to via Connect along with a static local port to listen for those connections. Example: db:1234,auth:6789 will start two local listeners for db on port 1234 and auth on port 6789, respectively. The name of the service is the name of the service registered with Consul. This value defaults to no upstreams.

### [»](https://www.consul.io/docs/platform/k8s/connect.html" \l "deployments-statefulsets-etc-)Deployments, StatefulSets, etc.

The annotations for configuring Connect must be on the pod specification. Since higher level resources such as Deployments wrap pod specification templates, Connect can be used with all of these higher level constructs, too.

An example Deployment below shows how to enable Connect injection:

apiVersion: apps/v1

kind: Deployment

metadata:

name: consul-example-deployment

spec:

replicas: 1

selector:

matchLabels:

app: consul-example

template:

metadata:

labels:

app: consul-example

annotations:

"consul.hashicorp.com/connect-inject": "true"

spec:

containers:

- name: example

image: "nginx"

**A common mistake** is to set the annotation on the Deployment or other resource. Ensure that the injector annotations are specified on the pod specification template as shown above.

## [»](https://www.consul.io/docs/platform/k8s/connect.html" \l "installation-and-configuration)Installation and Configuration

The Connect sidecar proxy is injected via a [mutating admission webhook](https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#admission-webhooks) provided by the [consul-k8s project](https://github.com/hashicorp/consul-k8s). This enables the automatic pod mutation shown in the usage section above. Installation of the mutating admission webhook is automated using the [Helm chart](https://www.consul.io/docs/platform/k8s/helm.html).

To install the Connect injector, enable the Connect injection feature using [Helm values](https://www.consul.io/docs/platform/k8s/helm.html#configuration-values-) and upgrade the installation using helm upgrade for existing installs or helm install for a fresh install. The Connect injector **also requires** [client agents](https://www.consul.io/docs/platform/k8s/helm.html#v-client) are enabled on the node with pods that are using Connect and that [gRPC is enabled](https://www.consul.io/docs/platform/k8s/helm.html#v-client-grpc).

connectInject:

enabled: true

client:

enabled: true

grpc: true

This will configure the injector to inject when the [injection annotation](https://www.consul.io/docs/platform/k8s/connect.html) is present. Other values in the Helm chart can be used to limit the namespaces the injector runs in, enable injection by default, and more.

As noted above, the Connect auto-injection requires that local client agents are configured. These client agents must be successfully joined to a Consul cluster. The Consul server cluster can run either in or out of a Kubernetes cluster.

### [»](https://www.consul.io/docs/platform/k8s/connect.html#verifying-the-installation)Verifying the Installation

To verify the installation, run the ["Accepting Inbound Connections"](https://www.consul.io/docs/platform/k8s/connect.html#accepting-inbound-connections) example from the "Usage" section above. After running this example, run kubectl get pod static-server -o yaml. In the raw YAML output, you should see injected Connect containers and an annotation consul.hashicorp.com/connect-inject-status set to injected. This confirms that injection is working properly.

If you do not see this, then use kubectl logs against the injector pod and note any errors.

# Connect Security

Connect enables secure service-to-service communication over mutual TLS. This provides both in-transit data encryption as well as authorization. This page will document how to secure Connect. For a full security model reference, see the dedicated [Consul security model](https://www.consul.io/docs/internals/security.html) page.

Connect will function in any Consul configuration. However, unless the checklist below is satisfied, Connect is not providing the security guarantees it was built for. The checklist below can be incrementally adopted towards full security if you prefer to operate in less secure models initially.

**Warning**: The checklist below should not be considered exhaustive. Please read and understand the [Consul security model](https://www.consul.io/docs/internals/security.html) in depth to assess whether your deployment satisfies the security requirements of Consul.

## [»](https://www.consul.io/docs/connect/security.html" \l "checklist)Checklist

### [»](https://www.consul.io/docs/connect/security.html" \l "acls-enabled-with-default-deny)ACLs Enabled with Default Deny

Consul must be configured to use ACLs with a default deny policy. This forces all requests to have explicit anonymous access or provide an ACL token. The configuration also forces all service-to-service communication to be explicitly whitelisted via an allow [intention](https://www.consul.io/docs/connect/intentions.html).

To learn how to enable ACLs, please see the [guide on ACLs](https://www.consul.io/docs/guides/acl.html).

**If ACLs are enabled but are in default allow mode**, then services will be able to communicate by default. Additionally, if a proper anonymous token is not configured, this may allow anyone to edit intentions. We do not recommend this. **If ACLs are not enabled**, deny intentions will still be enforced, but anyone may edit intentions. This renders the security of the created intentions effectively useless.

### [»](https://www.consul.io/docs/connect/security.html" \l "tcp-and-udp-encryption-enabled)TCP and UDP Encryption Enabled

TCP and UDP encryption must be enabled to prevent plaintext communication between Consul agents. At a minimum, verify\_outgoing should be enabled to verify server authenticity with each server having a unique TLS certificate.verify\_incoming provides additional agent verification, but doesn't directly affect Connect since requests must also always contain a valid ACL token. Clients calling Consul APIs should be forced over encrypted connections.

See the [Consul agent encryption page](https://www.consul.io/docs/agent/encryption.html) to learn more about configuring agent encryption.

**If encryption is not enabled**, a malicious actor can sniff network traffic or perform a man-in-the-middle attack to steal ACL tokens, always authorize connections, etc.

### [»](https://www.consul.io/docs/connect/security.html" \l "prevent-unauthorized-access-to-the-config-and-data-directories)Prevent Unauthorized Access to the Config and Data Directories

The configuration and data directories of the Consul agent on both clients and servers should be protected from unauthorized access. This protection must be done outside of Consul via access control systems provided by your target operating system.

The [full Consul security model](https://www.consul.io/docs/internals/security.html) explains the risk of unauthorized access for both client agents and server agents. In general, the blast radius of unauthorized access for client agent directories is much smaller than servers. However, both must be protected against unauthorized access.

### [»](https://www.consul.io/docs/connect/security.html" \l "prevent-non-connect-traffic-to-services)Prevent Non-Connect Traffic to Services

For services that are using [proxies](https://www.consul.io/docs/connect/proxies.html) (are not [natively integrated](https://www.consul.io/docs/connect/native.html)), network access via their unencrypted listeners must be restricted to only the proxy. This requires at a minimum restricting the listener to bind to loopback only. More complex solutions may involve using network namespacing techniques provided by the underlying operating system.

For scenarios where multiple services are running on the same machine without isolation, these services must all be trusted. We call this the **trusted multi-tenancy** deployment model. Any service could theoretically connect to any other service via the loopback listener, bypassing Connect completely. In this scenario, all services must be trusted or isolation mechanisms must be used.

For developer or operator access to a service, we recommend using a local Connect proxy. This is documented in the[development and debugging guide](https://www.consul.io/docs/connect/dev.html).

**If non-proxy traffic can communicate with the service**, this traffic will not be encrypted or authorized via Connect.

# Kubernetes

Consul has many integrations with Kubernetes. You can deploy Consul to Kubernetes using the Helm chart, sync services between Consul and Kubernetes, automatically secure Pod communication with Connect, and more. This section documents the official integrations between Consul and Kubernetes.

A step-by-step beginner tutorial and accompanying video can be found at the [Minikube with Consul guide](https://www.consul.io/docs/guides/minikube.html)

## [»](https://www.consul.io/docs/platform/k8s/index.html" \l "use-cases)Use Cases

**Running a Consul server cluster:** The Consul server cluster can run directly on Kubernetes. This can be used by both nodes within Kubernetes as well as nodes external to Kubernetes, as long as they can communicate to the server nodes via the network.

**Running Consul clients:** Consul clients can run as pods on every node and expose the Consul API to running pods. This enables many Consul tools such as envconsul, consul-template, and more to work on Kubernetes since a local agent is available. This will also register each Kubernetes node with the Consul catalog for full visibility into your infrastructure

**Service sync to enable Kubernetes and non-Kubernetes services to communicate:** Consul can sync Kubernetes services with its own service registry. This allows Kubernetes services to use native Kubernetes service discovery to discover and connect to external services, and for external services to use Consul service discovery to discover and connect to Kubernetes services.

**Automatic encryption and authorization for pod network connections:** Consul can automatically inject the [Connect](https://www.consul.io/docs/connect/index.html)sidecar into pods so that they can accept and establish encrypted and authorized network connections via mutual TLS. And because Connect can run anywhere, pods can also communicate with external services (and vice versa) over a fully encrypted connection.

**And more!** Consul can run directly on Kubernetes, so in addition to the native integrations provided by Consul itself, any other tool built for Kubernetes can choose to leverage Consul.

## [»](https://www.consul.io/docs/platform/k8s/index.html" \l "quot-consul-k8s-quot-project)"consul-k8s" Project

The dedicated [consul-k8s project](https://github.com/hashicorp/consul-k8s) contains the integration functionality between Consul and Kubernetes. You generally will not need to invoke this project directly since the [Helm chart](https://www.consul.io/docs/platform/k8s/helm.html) automates the installation and configuration of the project when necessary.

We may integrate this functionality directly into Consul in the future, but the separate project allows us to iterate and version the Kubernetes functionality separately. Additionally, a lot of the functionality works across multiple Consul versions, so you're able to update and resolve any Kubernetes integration issues without also upgrading Consul itself which can be more difficult.

# Running Consul on Kubernetes

Consul can run directly on Kubernetes, both in server or client mode. For pure-Kubernetes workloads, this enables Consul to also exist purely within Kubernetes. For heterogeneous workloads, Consul agents can join a server running inside or outside of Kubernetes.

This page starts with a large how-to section for various specific tasks. To learn more about the general architecture of Consul on Kubernetes, scroll down to the [architecture](https://www.consul.io/docs/platform/k8s/run.html#architecture) section.

## [»](https://www.consul.io/docs/platform/k8s/run.html" \l "helm-chart)Helm Chart

The recommended way to run Consul on Kubernetes is via the [Helm chart](https://www.consul.io/docs/platform/k8s/helm.html). This will install and configure all the necessary components to run Consul. The configuration enables you to run just a server cluster, just a client cluster, or both. Using the Helm chart, you can have a full Consul deployment up and running in seconds.

While the Helm chart exposes dozens of useful configurations and automatically sets up complex resources, it **does not automatically operate Consul.** You are still responsible for learning how to monitor, backup, upgrade, etc. the Consul cluster.

The Helm chart has no required configuration and will install a Consul cluster with sane defaults out of the box. Prior to going to production, it is highly recommended that you [learn about the configuration options](https://www.consul.io/docs/platform/k8s/helm.html#configuration-values-).

**Security Warning:** By default, the chart will install an insecure configuration of Consul. This provides a less complicated out-of-box experience for new users, but is not appropriate for a production setup. It is highly recommended to use a properly secured Kubernetes cluster or make sure that you understand and enable the [recommended security features](https://www.consul.io/docs/internals/security.html). Currently, some of these features are not supported in the Helm chart and require additional manual configuration.

## [»](https://www.consul.io/docs/platform/k8s/run.html" \l "how-to)How-To

### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "installing-consul)Installing Consul

To install Consul, clone the consul-helm repository, checkout the latest release, and install Consul. You can run helm install with the --dry-run flag to see the resources it would configure. In a production environment, you should always use the --dry-run flag prior to making any changes to the Consul cluster via Helm.

*# Clone the chart repo*

$ git clone https://github.com/hashicorp/consul-helm.git

$ cd consul-helm

*# Checkout a tagged version*

$ git checkout v0.1.0

*# Run Helm*

$ helm install --name consul ./

...

That's it. The Helm chart does everything to setup a recommended Consul-on-Kubernetes deployment. In a couple minutes, a Consul cluster will be formed and a leader elected and every node will have a running Consul agent.

The defaults will install both server and client agents. To install only one or the other, see the [chart configuration values](https://www.consul.io/docs/platform/k8s/helm.html#configuration-values-).

### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "viewing-the-consul-ui)Viewing the Consul UI

The Consul UI is enabled by default when using the Helm chart. For security reasons, it isn't exposed via a Service by default so you must use kubectl port-forward to visit the UI. Once the port is forwarded as shown below, navigate your browser to http://localhost:8500.

$ kubectl port-forward consul-server-0 8500:8500

...

The UI can also be exposed via a Kubernetes Service. To do this, configure the [ui.service chart values](https://www.consul.io/docs/platform/k8s/helm.html#v-ui-service).

### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "joining-an-existing-consul-cluster)Joining an Existing Consul Cluster

If you have a Consul cluster already running, you can configure your Kubernetes nodes to join this existing cluster.

global:

enabled: false

client:

enabled: true

join:

- "provider=my-cloud config=val ..."

The values.yaml file to configure the Helm chart sets the proper configuration to join an existing cluster.

The global.enabled value first disables all chart components by default so that each component is opt-in. This allows us to only setup the client agents. We then opt-in to the client agents by setting client.enabled to true.

Next, client.join is set to an array of valid [-retry-join values](https://www.consul.io/docs/agent/options.html#retry-join). In the example above, a fake [cloud auto-join](https://www.consul.io/docs/agent/cloud-auto-join.html) value is specified. This should be set to resolve to the proper addresses of your existing Consul cluster.

**Networking:** Note that for the Kubernetes nodes to join an existing cluster, the nodes (and specifically the agent pods) must be able to connect to all other server and client agents inside and outside of Kubernetes. If this isn't possible, consider running the Kubernetes agents as a separate DC or adopting Enterprise for [network segments](https://www.consul.io/docs/enterprise/network-segments/index.html).

### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "accessing-the-consul-http-api)Accessing the Consul HTTP API

The Consul HTTP API should be accessed by communicating to the local agent running on the same node. While technically any listening agent (client or server) can respond to the HTTP API, communicating with the local agent has important caching behavior, and allows you to use the simpler [/agent endpoints for services and checks](https://www.consul.io/api/agent.html).

For Consul installed via the Helm chart, a client agent is installed on each Kubernetes node. This is explained in the [architecture](https://www.consul.io/docs/platform/k8s/run.html#client-agents) section. To access the agent, you may use the [downward API](https://kubernetes.io/docs/tasks/inject-data-application/downward-api-volume-expose-pod-information/).

An example pod specification is shown below. In addition to pods, anything with a pod template can also access the downward API and can therefore also access Consul: StatefulSets, Deployments, Jobs, etc.

apiVersion: v1

kind: Pod

metadata:

name: consul-example

spec:

containers:

- name: example

image: "consul:latest"

env:

- name: HOST\_IP

valueFrom:

fieldRef:

fieldPath: status.hostIP

command:

- "/bin/sh"

- "-ec"

- |

export CONSUL\_HTTP\_ADDR="${HOST\_IP}:8500"

consul kv put hello world

restartPolicy: Never

An example Deployment is also shown below to show how the host IP can be accessed from nested pod specifications:

apiVersion: apps/v1

kind: Deployment

metadata:

name: consul-example-deployment

spec:

replicas: 1

selector:

matchLabels:

app: consul-example

template:

metadata:

labels:

app: consul-example

spec:

containers:

- name: example

image: "consul:latest"

env:

- name: HOST\_IP

valueFrom:

fieldRef:

fieldPath: status.hostIP

command:

- "/bin/sh"

- "-ec"

- |

export CONSUL\_HTTP\_ADDR="${HOST\_IP}:8500"

consul kv put hello world

### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "upgrading-consul-on-kubernetes)Upgrading Consul on Kubernetes

To upgrade Consul on Kubernetes, we follow the same pattern as [generally upgrading Consul](https://www.consul.io/docs/upgrading.html), except we can use the Helm chart to step through a rolling deploy. It is important to understand how to [generally upgrade Consul](https://www.consul.io/docs/upgrading.html) before reading this section.

Upgrading Consul on Kubernetes will follow the same pattern: each server will be updated one-by-one. After that is successful, the clients will be updated in batches.

#### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "upgrading-consul-servers)Upgrading Consul Servers

To initiate the upgrade, change the server.image value to the desired Consul version. For illustrative purposes, the example below will use consul:123.456. Also set the server.updatePartition value equal to the number of server replicas:

server:

image: "consul:123.456"

replicas: 3

updatePartition: 3

The updatePartition value controls how many instances of the server cluster are updated. Only instances with an index greater than the updatePartition value are updated (zero-indexed). Therefore, by setting it equal to replicas, none should update yet.

Next, run the upgrade. You should run this with --dry-run first to verify the changes that will be sent to the Kubernetes cluster.

$ helm upgrade consul ./

...

This should cause no changes (although the resource will be updated). If everything is stable, begin by decreasing the updatePartition value by one, and running helm upgrade again. This should cause the first Consul server to be stopped and restarted with the new image.

Wait until the Consul server cluster is healthy again (30s to a few minutes) then decrease updatePartition and upgrade again. Continue until updatePartition is 0. At this point, you may remove the updatePartition configuration. Your server upgrade is complete.

#### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "upgrading-consul-clients)Upgrading Consul Clients

With the servers upgraded, it is time to upgrade the clients. To upgrade the clients, set the client.image value to the desired Consul version. Then, run helm upgrade. This will upgrade the clients in batches, waiting until the clients come up healthy before continuing.

## [»](https://www.consul.io/docs/platform/k8s/run.html" \l "architecture)Architecture

We recommend running Consul on Kubernetes with the same [general architecture](https://www.consul.io/docs/internals/architecture.html) as running it anywhere else. There are some benefits Kubernetes can provide that eases operating a Consul cluster and we document those below. The standard [production deployment guide](https://www.consul.io/docs/guides/deployment.html) is still an important read even if running Consul within Kubernetes.

Each section below will outline the different components of running Consul on Kubernetes and an overview of the resources that are used within the Kubernetes cluster.

### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "server-agents)Server Agents

The server agents are run as a **StatefulSet**, using persistent volume claims to store the server state. This also ensures that the [node ID](https://www.consul.io/docs/agent/options.html#_node_id) is persisted so that servers can be rescheduled onto new IP addresses without causing issues. The server agents are configured with [anti-affinity](https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity) rules so that they are placed on different nodes. A readiness probe is configured that marks the pod as ready only when it has established a leader.

A **Service** is registered to represent the servers and expose the various ports. The DNS address of this service is used to join the servers to each other without requiring any other access to the Kubernetes cluster. The service is configured to publish non-ready endpoints so that it can be used for joining during bootstrap and upgrades.

Additionally, a **PodDisruptionBudget** is configured so the Consul server cluster maintains quorum during voluntary operational events. The maximum unavailable is (n/2)-1 where n is the number of server agents.

**Note:** Kubernetes and Helm do not delete Persistent Volumes or Persistent Volume Claims when a [StatefulSet is deleted](https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#stable-storage), so this must done manually when removing servers.

### [»](https://www.consul.io/docs/platform/k8s/run.html" \l "client-agents)Client Agents

The client agents are run as a **DaemonSet**. This places one agent (within its own pod) on each Kubernetes node. The clients expose the Consul HTTP API via a static port (default 8500) bound to the host port. This enables all other pods on the node to connect to the node-local agent using the host IP that can be retrieved via the Kubernetes downward API. See [accessing the Consul HTTP API](https://www.consul.io/docs/platform/k8s/run.html#accessing-the-consul-http-api) for an example.

There is a major limitation to this: there is no way to bind to a local-only host port. Therefore, any other node can connect to the agent. This should be considered for security. For a properly production-secured agent with TLS and ACLs, this is safe.

Some people prefer to run **Consul agent per pod** architectures, since this makes it easy to register the pod as a service easily. However, this turns a pod into a "node" in Consul and also causes an explosion of resource usage since every pod needs a Consul agent. We recommend instead running an agent (in a dedicated pod) per node, via the DaemonSet. This maintains the node equivalence in Consul. Service registration should be handled via the catalog syncing feature with Services rather than pods.

**Note:** Due to a limitation of anti-affinity rules with DaemonSets, a client-mode agent runs alongside server-mode agents in Kubernetes. This duplication wastes some resources, but otherwise functions perfectly fine.

# Helm Chart

The [Consul Helm chart](https://github.com/hashicorp/consul-helm) is the recommended way to install and configure Consul on Kubernetes. In addition to running Consul itself, the Helm chart is the primary method for installing and configuring Consul integrations with Kubernetes such as catalog syncing, Connect injection, and more.

This page assumes general knowledge of [Helm](https://helm.sh/) and how to use it. Using Helm to install Consul will require that Helm is properly installed and configured with your Kubernetes cluster.

**Important:** The Helm chart is new and may still change significantly over time. Please always run Helm with --dry-run before any install or upgrade to verify changes.

**Security Warning:** By default, the chart will install an insecure configuration of Consul. This provides a less complicated out-of-box experience for new users, but is not appropriate for a production setup. It is highly recommended to use a properly secured Kubernetes cluster or make sure that you understand and enable the [recommended security features](https://www.consul.io/docs/internals/security.html). Currently, some of these features are not supported in the Helm chart and require additional manual configuration.

## [»](https://www.consul.io/docs/platform/k8s/helm.html" \l "using-the-helm-chart)Using the Helm Chart

To use the Helm chart, you must download or clone the [consul-helm GitHub repository](https://github.com/hashicorp/consul-helm) and run Helm against the directory. We plan to transition to using a real Helm repository soon. When running Helm, we highly recommend you always checkout a specific tagged release of the chart to avoid any instabilities from master.

Prior to this, you must have Helm installed and configured both in your Kubernetes cluster and locally on your machine. The steps to do this are out of the scope of this document, please read the [Helm documentation](https://helm.sh/) for more information.

Example chart usage:

*# Clone the chart repo*

$ git clone https://github.com/hashicorp/consul-helm.git

$ cd consul-helm

*# Checkout a tagged version*

$ git checkout v0.1.0

*# Run Helm*

$ helm install --dry-run ./

**Warning:** By default, the chart will install everything: a Consul server cluster, client agents on all nodes, feature components, etc. This provides a nice out-of-box experience for new users, but may not be appropriate for a production setup. Consider setting the global.enabled value to false and opt-in to the various components.

## [»](https://www.consul.io/docs/platform/k8s/helm.html" \l "configuration-values-)Configuration (Values)

The chart is highly customizable using [Helm configuration values](https://docs.helm.sh/using_helm/#customizing-the-chart-before-installing). Each value has a sane default tuned for an optimal getting started experience with Consul. Before going into production, please review the parameters below and consider if they're appropriate for your deployment.

* [global](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-global) - These global values affect multiple components of the chart.

* + [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-global-enabled) (boolean: true) - The master enabled/disabled configuration. If this is true, most components will be installed by default. If this is false, no components will be installed by default and manually opt-in is required, such as by setting [server.enabled](https://www.consul.io/docs/platform/k8s/helm.html#v-) to true.

* + [domain](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-global-domain) (string: "consul") - The domain Consul uses for DNS queries. This is used to configure agents both for DNS listening but also to know what domain to join the cluster. This should be consistent throughout the chart, but can be overridden per-component as well.

* + [image](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-global-image) (string: "consul:latest") - The name of the Docker image (including any tag) for the containers running Consul agents. **This should be pinned to a specific version when running in production.** Otherwise, other changes to the chart may inadvertently upgrade your Consul version.

* + [imageK8S](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-global-imagek8s) (string: "hashicorp/consul-k8s:latest") - The name of the Docker image (including any tag) for the [consul-k8s](https://github.com/hashicorp/consul-k8s) binary. This is used by components such as catalog sync. **This should be pinned to a specific version when running in production.** Otherwise, other changes to the chart may inadvertently upgrade the version.

* + [datacenter](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-global-datacenter) (string: "dc1") - The name of the datacenter that the agent cluster should register as. This may not be changed once the cluster is bootstrapped and running, since Consul doesn't yet support an automatic way to change this value.

* [server](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server) - Values that configure running a Consul server within Kubernetes.

* + [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-enabled) (boolean: global.enabled) - If true, the chart will install all the resources necessary for a Consul server cluster. If you're running Consul externally and want agents within Kubernetes to join that cluster, this should probably be false.

* + [image](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-image) (string: global.image) - The name of the Docker image (including any tag) for the containers running Consul server agents.

* + [replicas](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-replicas) (integer: 3) -The number of server agents to run. This determines the fault tolerance of the cluster. Please see the [deployment table](https://www.consul.io/docs/internals/consensus.html#deployment-table) for more information.

* + [bootstrapExpect](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-bootstrapexpect) (integer: 3) - For new clusters, this is the number of servers to wait for before performing the initial leader election and bootstrap of the cluster. This must be less than or equal to server.replicas. This value is only used when bootstrapping new clusters, it has no effect during ongoing cluster maintenance.

* + [storage](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-storage) (string: 10Gi) - This defines the disk size for configuring the servers' StatefulSet storage. For dynamically provisioned storage classes, this is the desired size. For manually defined persistent volumes, this should be set to the disk size of the attached volume.

* + [storageClass](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-storageclass) (string: null) - The StorageClass to use for the servers' StatefulSet storage. It must be able to be dynamically provisioned if you want the storage to be automatically created. For example, to use [Local](https://kubernetes.io/docs/concepts/storage/storage-classes/#local)storage classes, the PersistentVolumeClaims would need to be manually created. A null value will use the Kubernetes cluster's default StorageClass. If a default StorageClass does not exist, you will need to create one.

* + [connect](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-connect) (boolean: true) - This will enable/disable [Connect](https://www.consul.io/docs/connect/index.html). Setting this to true will not automatically secure pod communication, this setting will only enable usage of the feature. Consul will automatically initialize a new CA and set of certificates. Additional Connect settings can be configured by setting the server.extraConfig value.

* + [resources](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-resources) (string: null) - The resource requests (CPU, memory, etc.) for each of the server agents. This should be a multi-line string mapping directly to a Kubernetes [ResourceRequirements](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#resourcerequirements-v1-core) object. If this isn't specified, then the pods won't request any specific amount of resources. **Setting this is highly recommended.**
  + # Resources are defined as a formatted multi-line string:
  + resources: |
  + requests:
  + memory: "10Gi"
  + limits:
  + memory: "10Gi"

* + [updatePartition](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-updatepartition) (integer: 0) - This value is used to carefully control a rolling update of Consul server agents. This value specifies the [partition](https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#partitions) for performing a rolling update. Please read the linked Kubernetes documentation for more information.

* + [disruptionBudget](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-disruptionbudget) - This configures the [PodDisruptionBudget](https://kubernetes.io/docs/tasks/run-application/configure-pdb/) for the server cluster.

* + - [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-disruptionbudget-enabled) (boolean: true) - This will enable/disable registering a PodDisruptionBudget for the server cluster. If this is enabled, it will only register the budget so long as the server cluster is enabled.

* + - [maxUnavailable](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-disruptionbudget-maxunavailable) (integer: null) - The maximum number of unavailable pods. By default, this will be automatically computed based on the server.replicas value to be (n/2)-1. If you need to set this to 0, you will need to add a --set 'server.disruptionBudget.maxUnavailable=0' flag to the helm chart installation command because of a limitation in the Helm templating language.

* + [extraConfig](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-extraconfig) (string: "{}") - A raw string of extra JSON [configuration](https://www.consul.io/docs/agent/options.html) for Consul servers. This will be saved as-is into a ConfigMap that is read by the Consul server agents. This can be used to add additional configuration that isn't directly exposed by the chart.
  + # ExtraConfig values are formatted as a multi-line string:
  + extraConfig: |
  + {
  + "log\_level": "DEBUG"
  + }

This can also be set using Helm's --set flag (consul-helm v0.7.0 and later), using the following syntax:

--set 'server.extraConfig="{"log\_level": "DEBUG"}"'

* + [extraVolumes](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-extravolumes) (array: []) - A list of extra volumes to mount for server agents. This is useful for bringing in extra data that can be referenced by other configurations at a well known path, such as TLS certificates or Gossip encryption keys. The value of this should be a list of objects. Each object supports the following keys:

* + - [type](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-extravolumes-type) (string: required) - Type of the volume, must be one of "configMap" or "secret". Case sensitive.

* + - [name](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-extravolumes-name) (string: required) - Name of the configMap or secret to be mounted. This also controls the path that it is mounted to. The volume will be mounted to /config/userconfig/<name>.

* + - [load](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-extravolumes-load) (boolean: false) - If true, then the agent will be configured to automatically load HCL/JSON configuration files from this volume with -config-dir. This defaults to false.

* + [affinity](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-affinity) (string) - This value defines the [affinity](https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity) for server pods. It defaults to allowing only a single pod on each node, which minimizes risk of the cluster becoming unusable if a node is lost. If you need to run more pods per node (for example, testing on Minikube), set this value to null.
  + # Recommended default server affinity:
  + affinity: |
  + podAntiAffinity:
  + requiredDuringSchedulingIgnoredDuringExecution:
  + - labelSelector:
  + matchLabels:
  + app: {{ template "consul.name" . }}
  + release: "{{ .Release.Name }}"
  + component: server
  + topologyKey: kubernetes.io/hostname

* + [priorityClassName](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-priorityclassname) (string) - This value references an existing Kubernetes [priorityClassName](https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/#pod-priority) that can be assigned to server pods.

* + [annotations](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-server-annotations) (string) - This value defines additional annotations for server pods. This should be a formatted as a multi-line string.
  + annotations: |
  + "sample/annotation1": "foo"
  + "sample/annotation2": "bar"

* [client](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client) - Values that configure running a Consul client on Kubernetes nodes.

* + [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-enabled) (boolean: global.enabled) - If true, the chart will install all the resources necessary for a Consul client on every Kubernetes node. This does not require server.enabled, since the agents can be configured to join an external cluster.

* + [image](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-image) (string: global.image) - The name of the Docker image (including any tag) for the containers running Consul client agents.

* + [join](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-join) (array<string>: null) - A list of valid [-retry-join values](https://www.consul.io/docs/agent/options.html#retry-join). If this is null (default), then the clients will attempt to automatically join the server cluster running within Kubernetes. This means that with server.enabled set to true, clients will automatically join that cluster. If server.enabled is not true, then a value must be specified so the clients can join a valid cluster.

* + [grpc](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-grpc) (boolean: false) - If true, agents will enable their GRPC listener on port 8502 and expose it to the host. This will use slightly more resources, but is required for [Connect](https://www.consul.io/docs/platform/k8s/connect.html).

* + [resources](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-resources) (string: null) - The resource requests (CPU, memory, etc.) for each of the client agents. This should be a multi-line string mapping directly to a Kubernetes [ResourceRequirements](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#resourcerequirements-v1-core) object. If this isn't specified, then the pods won't request any specific amount of resources.
  + # Resources are defined as a formatted multi-line string:
  + resources: |
  + requests:
  + memory: "10Gi"
  + limits:
  + memory: "10Gi"

* + [extraConfig](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-extraconfig) (string: "{}") - A raw string of extra JSON [configuration](https://www.consul.io/docs/agent/options.html) for Consul clients. This will be saved as-is into a ConfigMap that is read by the Consul agents. This can be used to add additional configuration that isn't directly exposed by the chart.
  + # ExtraConfig values are formatted as a multi-line string:
  + extraConfig: |
  + {
  + "log\_level": "DEBUG"
  + }

This can also be set using Helm's --set flag (consul-helm v0.7.0 and later), using the following syntax:

--set 'client.extraConfig="{"log\_level": "DEBUG"}"'

* + [extraVolumes](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-extravolumes) (array: []) - A list of extra volumes to mount for client agents. This is useful for bringing in extra data that can be referenced by other configurations at a well known path, such as TLS certificates or Gossip encryption keys. The value of this should be a list of objects. Each object supports the following keys:

* + - [type](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-extravolumes-type) (string: required) - Type of the volume, must be one of "configMap" or "secret". Case sensitive.

* + - [name](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-extravolumes-name) (string: required) - Name of the configMap or secret to be mounted. This also controls the path that it is mounted to. The volume will be mounted to /config/userconfig/<name>.

* + - [load](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-extravolumes-load) (boolean: false) - If true, then the agent will be configured to automatically load HCL/JSON configuration files from this volume with -config-dir. This defaults to false.

* + [priorityClassName](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-priorityclassname) (string) - This value references an existing Kubernetes [priorityClassName](https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/#pod-priority) that can be assigned to client pods.

* + [annotations](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-client-annotations) (string) - This value defines additional annotations for client pods. This should be a formatted as a multi-line string.
  + annotations: |
  + "sample/annotation1": "foo"
  + "sample/annotation2": "bar"

* [dns](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-dns) - Values that configure Consul DNS service.

* + [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-dns-enabled) (boolean: global.enabled) - If true, a consul-dns service will be created that exposes port 53 for TCP and UDP to the running Consul agents (servers and clients). This can then be used to [configure kube-dns](https://www.consul.io/docs/platform/k8s/dns.html). The Helm chart does not automatically configure kube-dns.

* [syncCatalog](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog) - Values that configure the [service sync](https://www.consul.io/docs/platform/k8s/service-sync.html) process.

* + [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-enabled) (boolean: false) - If true, the chart will install all the resources necessary for the catalog sync process to run.

* + [image](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-image) (string: global.imageK8S) - The name of the Docker image (including any tag) for [consul-k8s](https://www.consul.io/docs/platform/k8s/index.html#quot-consul-k8s-quot-project) to run the sync program.

* + [default](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-default) (boolean: true) - If true, all valid services in K8S are synced by default. If false, the service must be [annotated](https://www.consul.io/docs/platform/k8s/service-sync.html#sync-enable-disable) properly to sync. In either case an annotation can override the default.

* + [toConsul](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-toconsul) (boolean: true) - If true, will sync Kubernetes services to Consul. This can be disabled to have a one-way sync.

* + [toK8S](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-tok8s) (boolean: true) - If true, will sync Consul services to Kubernetes. This can be disabled to have a one-way sync.

* + [k8sPrefix](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-k8sprefix) (string: "") - A prefix to prepend to all services registered in Kubernetes from Consul. This defaults to "" where no prefix is prepended; Consul services are synced with the same name to Kubernetes. (Consul -> Kubernetes sync only)

* + [k8sTag](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-k8stag) (string: null) - An optional tag that is applied to all of the Kubernetes services that are synced into Consul. If nothing is set, this defaults to "k8s". (Kubernetes -> Consul sync only)

* + [syncClusterIPServices](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-clusterip-sync) (boolean: true) - If true, will sync Kubernetes ClusterIP services to Consul. This can be disabled to have the sync ignore ClusterIP-type services.

* + [nodePortSyncType](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-nodeport-sync) (string: ExternalFirst) - Configures the type of syncing that happens for NodePort services. The only valid options are: ExternalOnly, InternalOnly, and ExternalFirst. ExternalOnly will only use a node's ExternalIP address for the sync, otherwise the service will not be synced. InternalOnly uses the node's InternalIP address. ExternalFirst will preferentially use the node's ExternalIP address, but if it doesn't exist, it will use the node's InternalIP address instead.

* + [aclSyncToken](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-acl-sync-token) - references a Kubernetes [secret](https://kubernetes.io/docs/concepts/configuration/secret/#creating-your-own-secrets) that contains an existing Consul ACL token. This will provide the sync process the correct permissions. This is only needed if ACLs are enabled on the Consul cluster.

* + [secretName](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-acl-sync-token-secret-name)(string: null) - The name of the Kubernetes secret. This defaults to null.

* + [secretKey](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-synccatalog-acl-sync-token-secret-key)(string: null) - The key for the Kubernetes secret. This defaults to null.

* [ui](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-ui) - Values that configure the Consul UI.

* + [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-ui-enabled) (boolean: global.enabled) - If true, the UI will be enabled. This will only enable the UI, it doesn't automatically register any service for external access. The UI will only be enabled on server agents. If server.enabled is false, then this setting has no effect. To expose the UI in some way, you must configure ui.service.

* + [service](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-ui-service) - This configures the Service resource registered for the Consul UI.

* + - [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-ui-service-enabled) (boolean: true) - This will enable/disable registering a Kubernetes Service for the Consul UI. This value only takes effect if ui.enabled is true and taking effect.

* + - [type](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-ui-service-type) (string: null) - The service type to register. This defaults to null which doesn't set an explicit service type, which typically is defaulted to "ClusterIP" by Kubernetes. The available service types are documented on [the Kubernetes website](https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types).

* [connectInject](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject) - Values that configure running the [Connect injector](https://www.consul.io/docs/platform/k8s/connect.html).

* + [enabled](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-enabled) (boolean: false) - If true, the chart will install all the resources necessary for the Connect injector process to run. This will enable the injector but will require pods to opt-in with an annotation by default.

* + [image](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-image) (string: global.imageK8S) - The name of the Docker image (including any tag) for the [consul-k8s](https://github.com/hashicorp/consul-k8s)binary.

* + [default](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-default) (boolean: false) - If true, the injector will inject the Connect sidecar into all pods by default. Otherwise, pods must specify the. [injection annotation](https://www.consul.io/docs/platform/k8s/connect.html#consul-hashicorp-com-connect-inject) to opt-in to Connect injection. If this is true, pods can use the same annotation to explicitly opt-out of injection.

* + [imageConsul](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-imageConsul) (string: global.image) - The name of the Docker image (including any tag) for Consul. This is used for proxy service registration, Envoy configuration, etc.

* + [imageEnvoy](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-imageEnvoy) (string: "") - The name of the Docker image (including any tag) for the Envoy sidecar. envoymust be on the executable path within this image. This Envoy version must be compatible with the Consul version used by the injector. This defaults to letting the injector choose the Envoy image, which is usually envoy/envoy-alpine.

* + [namespaceSelector](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-namespaceselector) (string: "") - A [selector](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/) for restricting injection to only matching namespaces. By default all namespaces except the system namespace will have injection enabled.

* + [certs](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-certs) - The certs section configures how the webhook TLS certs are configured. These are the TLS certs for the Kube apiserver communicating to the webhook. By default, the injector will generate and manage its own certs, but this requires the ability for the injector to update its own MutatingWebhookConfiguration. In a production environment, custom certs should probably be used. Configure the values below to enable this.

* + - [secretName](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-certs-secretname) (string: null) - secretName is the name of the Kubernetes secret that has the TLS certificate and private key to serve the injector webhook. If this is null, then the injector will default to its automatic management mode.

* + - [caBundle](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-cabundle) (string: "") - The PEM-encoded CA public certificate bundle for the TLS certificate served by the injector. This must be specified as a string and can't come from a secret because it must be statically configured on the Kubernetes MutatingAdmissionWebhook resource. This only needs to be specified if secretName is not null.

* + - [certName](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-certs-certname) (string: "tls.crt") - The name of the certificate file within the secretName secret.

* + - [keyName](https://www.consul.io/docs/platform/k8s/helm.html" \l "v-connectinject-certs-keyname) (string: "tls.key") - The name of the private key for the certificate file within the secretNamesecret.

## [»](https://www.consul.io/docs/platform/k8s/helm.html" \l "using-the-helm-chart-to-deploy-consul-enterprise)Using the Helm Chart to deploy Consul Enterprise

You can also use this Helm chart to deploy Consul Enterprise by following a few extra steps.

Find the license file that you received in your welcome email. It should have the extension .hclic. You will use the contents of this file to create a Kubernetes secret before installing the Helm chart.

**Note:** If you cannot find your .hclic file, please contact your sales team or Technical Account Manager.

You can use the following commands to create the secret:

secret**=$(**cat 1931d1f4-bdfd-6881-f3f5-19349374841f.hclic**)**

kubectl create secret generic consul-ent-license --from-literal**=**"key=**${**secret**}**"

In your values.yaml, change the value of global.image to one of the enterprise [release tags](https://hub.docker.com/r/hashicorp/consul-enterprise/tags).

global:

image: "hashicorp/consul-enterprise:1.4.3-ent"

Add the name of the secret you just created to server.enterpriseLicense.

server:

enterpriseLicense:

secretName: "consul-ent-license"

secretKey: "key"

Add the --wait option to your helm install command. This will force Helm to wait for all the pods to become ready before it applies the license to your Consul cluster.

$ helm install --wait .

Once the cluster is up, you can verify the nodes are running Consul Enterprise.

$ kubectl port-forward service/consul-server 8500 &

$ consul license get

License is valid

License ID: 1931d1f4-bdfd-6881-f3f5-19349374841f

Customer ID: b2025a4a-8fdd-f268-95ce-1704723b9996

Expires At: 2020-03-09 03:59:59.999 +0000 UTC

Datacenter: **\***

Package: premium

Licensed Features:

Automated Backups

Automated Upgrades

Enhanced Read Scalability

Network Segments

Redundancy Zone

Advanced Network Federation

$ consul members

Node Address Status Type Build Protocol DC Segment

consul-server-0 10.60.0.187:8301 alive server 1.4.3+ent 2 dc1 <all>

consul-server-1 10.60.1.229:8301 alive server 1.4.3+ent 2 dc1 <all>

consul-server-2 10.60.2.197:8301 alive server 1.4.3+ent 2 dc1 <all>

## [»](https://www.consul.io/docs/platform/k8s/helm.html" \l "helm-chart-examples)Helm Chart Examples

The below values.yaml can be used to set up a single server Consul cluster with a LoadBalancer to allow external access to the UI and API.

global:

enabled: true

image: "consul:1.4.2"

domain: consul

datacenter: dc1

server:

enabled: true

replicas: 1

bootstrapExpect: 1

storage: 10Gi

client:

enabled: true

dns:

enabled: true

ui:

enabled: true

service:

enabled: true

type: LoadBalancer

The below values.yaml can be used to set up a three server Consul Enterprise cluster with 100GB of storage and automatic Connect injection for annotated pods in the "my-app" namespace.

Note, this would require a secret that contains the enterprise license key.

global:

enabled: true

domain: consul

image: "hashicorp/consul-enterprise:1.4.2-ent"

datacenter: dc1

server:

enabled: true

replicas: 3

bootstrapExpect: 3

enterpriseLicense:

secretName: "consul-license"

secretKey: "key"

storage: 100Gi

connect: true

affinity: |

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchLabels:

app: {{ template "consul.name" . }}

release: "{{ .Release.Name }}"

component: server

topologyKey: kubernetes.io/hostname

client:

enabled: true

grpc: true

dns:

enabled: true

ui:

enabled: true

service:

enabled: true

type: NodePort

connectInject:

enabled: true

default: false

namespaceSelector: "my-app"

## [»](https://www.consul.io/docs/platform/k8s/helm.html" \l "customizing-the-helm-chart)Customizing the Helm Chart

Consul within Kubernetes is highly configurable and the Helm chart contains dozens of the most commonly used configuration options. If you need to extend the Helm chart with additional options, we recommend using a third-party tool, such as [kustomize](https://github.com/kubernetes-sigs/kustomize) or [ship](https://github.com/replicatedhq/ship).

# Out-of-Cluster Nodes

Non-Kubernetes nodes can join a Consul cluster running within Kubernetes. These are considered "out-of-cluster" nodes.

## [»](https://www.consul.io/docs/platform/k8s/out-of-cluster-nodes.html" \l "auto-join)Auto-join

The recommended way to join a cluster running within Kubernetes is to use the ["k8s" cloud auto-join provider](https://www.consul.io/docs/agent/cloud-auto-join.html#kubernetes-k8s-).

The auto-join provider dynamically discovers IP addresses to join using the Kubernetes API. It authenticates with Kubernetes using a standard kubeconfig file. This works with all major hosted Kubernetes offerings as well as self-hosted installations.

The auto-join string below will join a Consul server cluster that is started using the [official Helm chart](https://www.consul.io/docs/platform/k8s/helm.html):

$ consul agent -retry-join 'provider=k8s label\_selector="app=consul,component=server"'

By default, Consul will join the default Gossip port. Pods may set an annotation consul.hashicorp.com/auto-join-portto an integer value or a named port to specify the port for the auto-join to return. This enables different pods to have different exposed ports.

## [»](https://www.consul.io/docs/platform/k8s/out-of-cluster-nodes.html" \l "networking)Networking

Consul typically requires a fully connected network. Therefore, out-of-cluster nodes joining a cluster running within Kubernetes must be able to communicate to pod IPs or Kubernetes node IPs via the network.

**Consul Enterprise customers** may use [network segments](https://www.consul.io/docs/enterprise/network-segments/index.html) to enable non-fully-connected topologies. However, out-of-cluster nodes must still be able to communicate with the server pod or host IP addresses.

The auto-join provider discussed above will use pod IPs by default. The host\_network=true setting may be set to use host IPs, however all the ports Consul requires must be exposed via a hostPort. If no ports are exposed via hostPort, the pod will not be discovered.

# Consul DNS on Kubernetes

One of the primary query interfaces to Consul is the [DNS interface](https://www.consul.io/docs/agent/dns.html). The Consul DNS interface can be exposed for all pods in Kubernetes using a [stub-domain configuration](https://kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/#configure-stub-domain-and-upstream-dns-servers).

The stub-domain configuration must point to a static IP of a DNS resolver. The [Helm chart](https://www.consul.io/docs/platform/k8s/helm.html) creates a consul-dns service by default that exports Consul DNS. The cluster IP of this service can be used to configure a stub-domain with kube-dns. While the kube-dns configuration lives in the kube-system namepace, the IP just has to be routable so the service can live in a different namespace.

cat <<EOF | kubectl apply -f -

apiVersion: v1

kind: ConfigMap

metadata:

labels:

addonmanager.kubernetes.io/mode: EnsureExists

name: kube-dns

namespace: kube-system

data:

stubDomains: |

{"consul": ["$(kubectl get svc consul-dns -o jsonpath='{.spec.clusterIP}')"]}

EOF

**Note:** The stubDomain can only point to a static IP. If the cluster IP of the consul-dns service changes, then it must be updated in the config map to match the new service IP for this to continue working. This can happen if the service is deleted and recreated, such as in full cluster rebuilds.

## [»](https://www.consul.io/docs/platform/k8s/dns.html" \l "coredns-configuration)CoreDNS Configuration

If you are using CoreDNS instead of kube-dns in your Kubernetes cluster, you will need to update your existing corednsConfigMap in the kube-system namespace to include a proxy definition for consul that points to the cluster IP of theconsul-dns service.

apiVersion: v1

kind: ConfigMap

metadata:

labels:

addonmanager.kubernetes.io/mode: EnsureExists

name: coredns

namespace: kube-system

data:

Corefile: |

.:53 {

<Existing CoreDNS definition>

}

consul {

errors

cache 30

proxy . <consul-dns service cluster ip>

}

**Note:** The consul proxy can only point to a static IP. If the cluster IP of the consul-dns service changes, then it must be updated to the new IP to continue working. This can happen if the service is deleted and recreated, such as in full cluster rebuilds.

## [»](https://www.consul.io/docs/platform/k8s/dns.html" \l "verifying-dns-works)Verifying DNS Works

To verify DNS works, run a simple job to query DNS. Save the following job to the file job.yaml and run it:

apiVersion: batch/v1

kind: Job

metadata:

name: dns

spec:

template:

spec:

containers:

- name: dns

image: anubhavmishra/tiny-tools

command: ["dig", "consul.service.consul"]

restartPolicy: Never

backoffLimit: 4

$ kubectl apply -f job.yaml

Then query the pod name for the job and check the logs. You should see output similar to the following showing a successful DNS query. If you see any errors, then DNS is not configured properly.

$ kubectl get pods --show-all | grep dns

dns-lkgzl 0/1 Completed 0 6m

$ kubectl logs dns-lkgzl

; <<>> DiG 9.11.2-P1 <<>> consul.service.consul

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4489

;; flags: qr aa rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 4

;; OPT PSEUDOSECTION:

; EDNS: version: 0, flags:; udp: 4096

;; QUESTION SECTION:

;consul.service.consul. IN A

;; ANSWER SECTION:

consul.service.consul. 0 IN A 10.36.2.23

consul.service.consul. 0 IN A 10.36.4.12

consul.service.consul. 0 IN A 10.36.0.11

;; ADDITIONAL SECTION:

consul.service.consul. 0 IN TXT "consul-network-segment="

consul.service.consul. 0 IN TXT "consul-network-segment="

consul.service.consul. 0 IN TXT "consul-network-segment="

;; Query time: 5 msec

;; SERVER: 10.39.240.10#53(10.39.240.10)

;; WHEN: Wed Sep 12 02:12:30 UTC 2018

;; MSG SIZE rcvd: 206

# Syncing Kubernetes and Consul Services

The services in Kubernetes and Consul can be automatically synced so that Kubernetes services are available to Consul agents and services in Consul can be available as first-class Kubernetes services. This functionality is provided by the[consul-k8s project](https://github.com/hashicorp/consul-k8s) and can be automatically installed and configured using the [Consul Helm chart](https://www.consul.io/docs/platform/k8s/helm.html).

**Why sync Kubernetes services to Consul?** Kubernetes services synced to the Consul catalog enable Kubernetes services to be accessed by any node that is part of the Consul cluster, including other distinct Kubernetes clusters. For non-Kubernetes nodes, they can access services using the standard [Consul DNS](https://www.consul.io/docs/agent/dns.html) or HTTP API.

**Why sync Consul services to Kubernetes?** Syncing Consul services to Kubernetes services enables non-Kubernetes services (such as external to the cluster) to be accessed in a native Kubernetes way: using kube-dns, environment variables, etc. This makes it very easy to automate external service discovery, including hosted services like databases.

## [»](https://www.consul.io/docs/platform/k8s/service-sync.html#installation-and-configuration)Installation and Configuration

The service sync is done using an external long-running process in the [consul-k8s project](https://github.com/hashicorp/consul-k8s). This process can run either in or out of a Kubernetes cluster. However, running this within the Kubernetes cluster is generally easier since it is automated using the [Helm chart](https://www.consul.io/docs/platform/k8s/helm.html).

The Consul server cluster can run either in or out of a Kubernetes cluster. The Consul server cluster does not need to be running on the same machine or same platform as the sync process. The sync process needs to be configured with the address to the Consul cluster as well as any additional access information such as ACL tokens.

To install the sync, enable the catalog sync feature using [Helm values](https://www.consul.io/docs/platform/k8s/helm.html#configuration-values-) and upgrade the installation using helm upgradefor existing installs or helm install for a fresh install.

syncCatalog:

enabled: true

This will enable services to sync in both directions. You can also choose to only sync Kubernetes services to Consul or vice versa by disabling a direction. See the [Helm configuration](https://www.consul.io/docs/platform/k8s/helm.html#configuration-values-) for more information.

**Before installing,** please read the introduction paragraphs for the reference documentation below for both[Kubernetes to Consul](https://www.consul.io/docs/platform/k8s/service-sync.html#kubernetes-to-consul) and [Consul to Kubernetes](https://www.consul.io/docs/platform/k8s/service-sync.html#consul-to-kubernetes) sync to understand how the syncing works.

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "authentication)Authentication

The sync process must authenticate to both Kubernetes and Consul to read and write services.

For Kubernetes, a valid kubeconfig file must be provided with cluster and authentication information. The sync process will look into the default locations for both in-cluster and out-of-cluster authentication. If kubectl works, then the sync program should work.

For Consul, if ACLs are configured on the cluster, a Consul [ACL token](https://learn.hashicorp.com/consul/advanced/day-1-operations/acl-guide) will need to be provided. Review the [ACL rules](https://www.consul.io/docs/agent/acl-rules.html) when creating this token so that it only allows the necessary privileges. The catalog sync process accepts this token by using the [CONSUL\_HTTP\_TOKEN](https://www.consul.io/docs/commands/index.html#consul_http_token) environment variable. This token should be set as a [Kubernetes secret](https://kubernetes.io/docs/concepts/configuration/secret/#creating-your-own-secrets) and referenced in the Helm chart.

## [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "kubernetes-to-consul)Kubernetes to Consul

This sync registers Kubernetes services to the Consul catalog automatically.

This enables discovery and connection to Kubernetes services using native Consul service discovery such as DNS or HTTP. This is particularly useful for non-Kubernetes nodes. This also causes all discoverable services to be part of a central service catalog in Consul for further syncing into alternate Kubernetes clusters or other platforms.

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "kubernetes-service-types)Kubernetes Service Types

Not all Kubernetes services are externally accessible. The sync program by default will only sync services of the following types or configurations. If a service type is not listed below, then the sync program will ignore that service type.

#### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "nodeport)NodePort

[NodePort services](https://kubernetes.io/docs/concepts/services-networking/service/#nodeport) register a static port that every node in the K8S cluster listens on.

For NodePort services, a Consul service instance will be created for each node that has the representative pod running. While Kubernetes configures a static port on all nodes in the cluster, this limits the number of service instances to be equal to the nodes running the target pods.

The service instances will be registered to the Kubernetes node name that each instance lives on. This is guaranteed unique by Kubernetes. An existing node entry will be used if it is already part of the Consul cluster (for example if you're running a client agent on all Kubernetes nodes). This allows the normal agent health checks for that node to continue working.

#### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "loadbalancer)LoadBalancer

For LoadBalancer services, a single service instance will be registered with the external IP of the created load balancer. Because this is already a load balancer, only one service instance will be registered with Consul rather than registering each individual pod endpoint.

#### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "external-ips)External IPs

Any service type may specify an "[external IP](https://kubernetes.io/docs/concepts/services-networking/service/#external-ips)" configuration. The external IP must be configured by some other system, but any service discovery will resolve to this set of IP addresses rather than a virtual IP.

If an external IP list is present, a service instance in Consul will be created for each external IP. It is assumed that if an external IP is present that it is routable and configured by some other system.

#### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "clusterip)ClusterIP

ClusterIP services are synced by default as of consul-k8s version 0.3.0. In many Kubernetes clusters, ClusterIPs may not be accessible outside of the cluster, so you may end up with services registered in Consul that are not routeable. To skip syncing ClusterIP services, set [syncClusterIPServices](https://www.consul.io/docs/platform/k8s/helm.html#v-synccatalog-clusterip-sync) to false in the Helm chart values file.

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "sync-enable-disable)Sync Enable/Disable

By default, all valid services (as explained above) are synced. This default can be changed using the [configuration](https://www.consul.io/docs/platform/k8s/helm.html#v-synccatalog-default). Syncing can also be explicitly enabled or disabled using an annotation:

kind: Service

apiVersion: v1

metadata:

name: my-service

annotations:

"consul.hashicorp.com/service-sync": "false"

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "service-name)Service Name

When a Kubernetes service is synced to Consul, the name of the service in Consul by default will be the value of the "name" metadata on that Kubernetes service. This makes it so that service sync works with zero configuration changes. This can be overridden using an annotation to specify the Consul service name:

kind: Service

apiVersion: v1

metadata:

name: my-service

annotations:

"consul.hashicorp.com/service-name": my-consul-service

**If a conflicting service name exists in Consul,** the sync program will register additional instances to that same service. Therefore, services inside and outside of Kubernetes should have different names unless you want either side to potentially connect. This default behavior also enables gracefully transitioning a service from outside of K8S to inside, and vice versa.

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "service-ports)Service Ports

When syncing the Kubernetes service to Consul, the Consul service port will be the first defined port in the service. Additionally, all ports will be registered in the service instance metadata with the key "port-X" where X is the name of the port and the value is the externally accessible port.

The default service port can be overridden using an annotation:

kind: Service

apiVersion: v1

metadata:

name: my-service

annotations:

"consul.hashicorp.com/service-port": "http"

The annotation value may a name of a port (recommended) or an exact port value.

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "service-tags)Service Tags

A service registered in Consul from Kubernetes will always have the tag "k8s" added to it. Additional tags can be specified with a comma-separated annotation value as shown below. This will also automatically include the "k8s" tag which can't be disabled. The values should be specified comma-separated without any additional whitespace.

kind: Service

apiVersion: v1

metadata:

name: my-service

annotations:

"consul.hashicorp.com/service-tags": "primary,foo"

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "service-meta)Service Meta

A service registered in Consul from Kubernetes will set the external-source key to "kubernetes". This can be used by API consumers, the UI, CLI, etc. to filter service instances that are set in k8s. The Consul UI (in Consul 1.2.3 and later) will read this value to show a Kubernetes icon next to all externally registered services from Kubernetes.

Additional metadata can be specified using annotations. The "KEY" below can be set to any key. This allows setting multiple meta values:

kind: Service

apiVersion: v1

metadata:

name: my-service

annotations:

"consul.hashicorp.com/service-meta-KEY": "value"

## [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "consul-to-kubernetes)Consul to Kubernetes

This syncs Consul services into first-class Kubernetes services. Each Consul service is synced to an [ExternalName](https://kubernetes.io/docs/concepts/services-networking/service/#externalname) service in Kubernetes. The external name is configured to be the Consul DNS entry.

This enables external services to be discovered using native Kubernetes tooling. This can be used to ease software migration into or out of Kubernetes, across platforms, to and from hosted services, and more.

**Requires Consul DNS via CoreDNS in Kubernetes:** This feature requires that [Consul DNS](https://www.consul.io/docs/platform/k8s/dns.html) is configured within Kubernetes. Additionally, [**CoreDNS**](https://kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/#config-coredns)**is required (instead of kube-dns)** to resolve an issue with resolving externalName services pointing to custom domains. In the future we hope to remove this requirement by syncing the instance addresses directly into service endpoints.

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "sync-enable-disable-1)Sync Enable/Disable

All Consul services visible to the sync process based on its given ACL token will be synced to Kubernetes.

There is no way to change this behavior per service. For the opposite sync direction (Kubernetes to Consul), you can use Kubernetes annotations to disable a sync per service. This is not currently possible for Consul to Kubernetes sync and the ACL token must be used to limit what services are synced.

In the future, we hope to support per-service configuration.

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "service-name-1)Service Name

When a Consul service is synced to Kubernetes, the name of the Kubernetes service will exactly match the name of the Consul service.

To change this default exact match behavior, it is possible to specify a prefix to be added to service names within Kubernetes by using the -k8s-service-prefix flag. This can also be specified in the Helm configuration.

**If a conflicting service is found,** the service will not be synced. This does not match the Kubernetes to Consul behavior, but given the current implementation we must do this because Kubernetes can't mix both CNAME and Endpoint-based services.

### [»](https://www.consul.io/docs/platform/k8s/service-sync.html" \l "kubernetes-service-labels-and-annotations)Kubernetes Service Labels and Annotations

Any Consul services synced to Kubernetes will be labeled and annotated. An annotation consul.hashicorp.com/syncedwill be set to "true" to note that this is a synced service from Consul.

Additionally, a label consul=true will be specified so that label selectors can be used with kubectl and other tooling to easily filter all Consul-synced services.

# Connect Sidecar on Kubernetes

[Connect](https://www.consul.io/docs/connect/index.html) is a feature built into to Consul that enables automatic service-to-service authorization and connection encryption across your Consul services. Connect can be used with Kubernetes to secure pod communication with other pods and external Kubernetes services.

The Connect sidecar running Envoy can be automatically injected into pods in your cluster, making configuration for Kubernetes automatic. This functionality is provided by the [consul-k8s project](https://github.com/hashicorp/consul-k8s) and can be automatically installed and configured using the [Consul Helm chart](https://www.consul.io/docs/platform/k8s/helm.html).

## [»](https://www.consul.io/docs/platform/k8s/connect.html#usage)Usage

When the [Connect injector is installed](https://www.consul.io/docs/platform/k8s/connect.html#installation-and-configuration), the Connect sidecar is automatically added to all pods. This sidecar can both accept and establish connections using Connect, enabling the pod to communicate to clients and dependencies exclusively over authorized and encrypted connections.

**Note:** The pod specifications in this section are valid and use publicly available images. If you've installed the Connect injector, feel free to run the pod specifications in this section to try Connect with Kubernetes. Please note the documentation below this section on how to properly install and configure the Connect injector.

### [»](https://www.consul.io/docs/platform/k8s/connect.html#accepting-inbound-connections)Accepting Inbound Connections

An example pod is shown below with Connect enabled to accept inbound connections. Notice that the pod would still be fully functional without Connect. Minimal to zero modifications are required to pod specifications to enable Connect in Kubernetes.

This pod specification starts a server that responds to any HTTP request with the static text "hello world".

apiVersion: v1

kind: Pod

metadata:

name: static-server

annotations:

"consul.hashicorp.com/connect-inject": "true"

spec:

containers:

- name: static-server

image: hashicorp/http-echo:latest

args:

- -text="hello world"

- -listen=:8080

ports:

- containerPort: 8080

name: http

The only change for Connect is the addition of the consul.hashicorp.com/connect-inject annotation. This enables injection for this pod. The injector can also be [configured](https://www.consul.io/docs/platform/k8s/connect.html#installation-and-configuration) to automatically inject unless explicitly disabled, but the default installation requires opt-in using the annotation shown above.

This will start a Connect sidecar that listens on a random port registered with Consul and proxies valid inbound connections to port 8080 in the pod. To establish a connection to the pod using Connect, a client must use another Connect proxy. The client Connect proxy will use Consul service discovery to find all available upstream proxies and their public ports.

In the example above, the server is listening on :8080. This means the server will still bind to the pod IP and allow external connections. This is useful to transition to Connect by allowing both Connect and non-Connect connections. To restrict access to only Connect-authorized clients, any listeners should bind to localhost only (such as 127.0.0.1).

### [»](https://www.consul.io/docs/platform/k8s/connect.html#connecting-to-connect-enabled-services)Connecting to Connect-Enabled Services

The example pod specification below configures a pod that is capable of establishing connections to our previous example "static-server" service. The connection to this static text service happens over an authorized and encrypted connection via Connect.

apiVersion: v1

kind: Pod

metadata:

name: static-client

annotations:

"consul.hashicorp.com/connect-inject": "true"

"consul.hashicorp.com/connect-service-upstreams": "static-server:1234"

spec:

containers:

- name: static-client

image: tutum/curl:latest

# Just spin & wait forever, we'll use `kubectl exec` to demo

command: [ "/bin/sh", "-c", "--" ]

args: [ "while true; do sleep 30; done;" ]

Pods must specify upstream dependencies with the [consul.hashicorp.com/connect-service-upstreams annotation](https://www.consul.io/docs/platform/k8s/connect.html#consul-hashicorp-com-connect-service-upstreams). This annotation declares the names of any upstream dependencies and a local port for the proxy to listen on. When a connection is established to that local port, the proxy establishes a connection to the target service ("static-server" in this example) using mutual TLS and identifying as the source service ("static-client" in this example).

The injector will also set environment variables <NAME>\_CONNECT\_SERVICE\_HOST and <NAME>\_CONNECT\_SERVICE\_PORT in every container in the pod for every defined upstream. This is analogous to the standard Kubernetes service environment variables, but point instead to the correct local proxy port to establish connections via Connect.

Any containers running in the pod that need to establish connections to dependencies must be reconfigured to use the local upstream address either directly or using the environment variables set by the injector (defined above). This means pods should not use Kubernetes service DNS or environment variables for these connections.

We can verify access to the static text server using kubectl exec. Notice that we use the local address and port from the upstream annotation (1234) for this verification.

$ kubectl exec static-client -- curl -s http://127.0.0.1:1234/

"hello world"

We can control access to the server using [intentions](https://www.consul.io/docs/connect/intentions.html). If you use the Consul UI or [CLI](https://www.consul.io/docs/commands/intention/create.html) to create a deny [intention](https://www.consul.io/docs/connect/intentions.html) between "static-client" and "static-server", connections are immediately rejected without updating either of the running pods. You can then remove this intention to allow connections again.

$ kubectl exec static-client -- curl -s http://127.0.0.1:1234/

command terminated with exit code 52

### [»](https://www.consul.io/docs/platform/k8s/connect.html#available-annotations)Available Annotations

Annotations can be used to configure the injection behavior.

* [consul.hashicorp.com/connect-inject](https://www.consul.io/docs/platform/k8s/connect.html#consul-hashicorp-com-connect-inject) - If this is "true" then injection is enabled. If this is "false" then injection is explicitly disabled. The default injector behavior requires pods to opt-in to injection by specifying this value as "true". This default can be changed in the injector's configuration if desired.
* [consul.hashicorp.com/connect-service](https://www.consul.io/docs/platform/k8s/connect.html#consul-hashicorp-com-connect-service) - For pods that accept inbound connections, this specifies the name of the service that is being served. This defaults to the name of the first container in the pod.
* [consul.hashicorp.com/connect-service-port](https://www.consul.io/docs/platform/k8s/connect.html#consul-hashicorp-com-connect-service-port) - For pods that accept inbound connections, this specifies the port to route inbound connections to. This is the port that the service is listening on. The service port defaults to the first exposed port on any container in the pod. If specified, the value can be the name of a configured port, such as "http" or it can be a direct port value such as "8080". This is the port of the service, the proxy public listener will listen on a dynamic port.
* [consul.hashicorp.com/connect-service-upstreams](https://www.consul.io/docs/platform/k8s/connect.html#consul-hashicorp-com-connect-service-upstreams) - The list of upstream services that this pod needs to connect to via Connect along with a static local port to listen for those connections. Example: db:1234,auth:6789 will start two local listeners for db on port 1234 and auth on port 6789, respectively. The name of the service is the name of the service registered with Consul. This value defaults to no upstreams.

### [»](https://www.consul.io/docs/platform/k8s/connect.html#deployments-statefulsets-etc-)Deployments, StatefulSets, etc.

The annotations for configuring Connect must be on the pod specification. Since higher level resources such as Deployments wrap pod specification templates, Connect can be used with all of these higher level constructs, too.

An example Deployment below shows how to enable Connect injection:

apiVersion: apps/v1

kind: Deployment

metadata:

name: consul-example-deployment

spec:

replicas: 1

selector:

matchLabels:

app: consul-example

template:

metadata:

labels:

app: consul-example

annotations:

"consul.hashicorp.com/connect-inject": "true"

spec:

containers:

- name: example

image: "nginx"

**A common mistake** is to set the annotation on the Deployment or other resource. Ensure that the injector annotations are specified on the pod specification template as shown above.

## [»](https://www.consul.io/docs/platform/k8s/connect.html#installation-and-configuration)Installation and Configuration

The Connect sidecar proxy is injected via a [mutating admission webhook](https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#admission-webhooks) provided by the [consul-k8s project](https://github.com/hashicorp/consul-k8s). This enables the automatic pod mutation shown in the usage section above. Installation of the mutating admission webhook is automated using the [Helm chart](https://www.consul.io/docs/platform/k8s/helm.html).

To install the Connect injector, enable the Connect injection feature using [Helm values](https://www.consul.io/docs/platform/k8s/helm.html#configuration-values-) and upgrade the installation using helm upgrade for existing installs or helm install for a fresh install. The Connect injector **also requires** [client agents](https://www.consul.io/docs/platform/k8s/helm.html#v-client) are enabled on the node with pods that are using Connect and that [gRPC is enabled](https://www.consul.io/docs/platform/k8s/helm.html#v-client-grpc).

connectInject:

enabled: true

client:

enabled: true

grpc: true

This will configure the injector to inject when the [injection annotation](https://www.consul.io/docs/platform/k8s/connect.html) is present. Other values in the Helm chart can be used to limit the namespaces the injector runs in, enable injection by default, and more.

As noted above, the Connect auto-injection requires that local client agents are configured. These client agents must be successfully joined to a Consul cluster. The Consul server cluster can run either in or out of a Kubernetes cluster.

### [»](https://www.consul.io/docs/platform/k8s/connect.html#verifying-the-installation)Verifying the Installation

To verify the installation, run the ["Accepting Inbound Connections"](https://www.consul.io/docs/platform/k8s/connect.html#accepting-inbound-connections) example from the "Usage" section above. After running this example, run kubectl get pod static-server -o yaml. In the raw YAML output, you should see injected Connect containers and an annotation consul.hashicorp.com/connect-inject-status set to injected. This confirms that injection is working properly.

If you do not see this, then use kubectl logs against the injector pod and note any errors.

# Consul Guides

This section provides various guides for common actions. Due to the nature of Consul, some of these procedures can be complex, so our goal is to provide guidance to do them safely.

The following guides are available:

* [ACLs](https://www.consul.io/docs/guides/acl-index.html) - This set of guides covers Consul's Access Control List (ACL) capability, which can be used to control access to Consul resources.
* [Adding/Removing Servers](https://learn.hashicorp.com/consul/day-2-operations/advanced-operations/servers) - This guide covers how to safely add and remove Consul servers from the cluster. This should be done carefully to avoid availability outages.
* [Agent Communication Encryption](https://learn.hashicorp.com/consul/advanced/day-1-operations/agent-encryption) - This guide covers how to encrypt both gossip and RPC communication.
* [Autopilot](https://learn.hashicorp.com/consul/day-2-operations/advanced-operations/autopilot) - This guide covers Autopilot, which provides automatic operator-friendly management of Consul servers.
* [Connect in Production](https://www.consul.io/docs/guides/connect-production.html) - This guide describes critical aspects of operating Consul Connect in Production.
* [Connect with Envoy](https://www.consul.io/docs/guides/connect-envoy.html) - This guide will describe how to setup a development-mode Consul server and two Envoy proxies on a single machine using Docker.
* [Consul Cluster Monitoring & Metrics](https://learn.hashicorp.com/consul/advanced/day-1-operations/monitoring) - After setting up your first datacenter, it is an ideal time to make sure your cluster is healthy and establish a baseline. This guide will cover several types of metrics in two sections: Consul health and server health.
* [Consul with Containers](https://www.consul.io/docs/guides/consul-containers.html) - This guide describes critical aspects of operating a Consul cluster that's run inside containers.
* [Consul Template](https://www.consul.io/docs/guides/consul-template.html) - This guide covers the Consul template tool, which provides a programmatic method for populating values into the file system.
* [Consul-AWS](https://www.consul.io/docs/guides/consul-aws.html) - This guide covers the Consul-AWS tool, which syncs Consul's and AWS Cloud Map's service catalog.
* [Creating Certificates](https://learn.hashicorp.com/consul/advanced/day-1-operations/certificates) - This guide describes how to setup CA and certificates to secure a Consul cluster with TLS.
* [Datacenter Backups](https://learn.hashicorp.com/consul/advanced/day-1-operations/backup) - Consul provide the snapshot tool for backing up and restoring data. In this guide you will learn how to use both.
* [Deployment Guide](https://learn.hashicorp.com/consul/advanced/day-1-operations/deployment-guide) - This deployment guide covers the steps required to install and configure a single HashiCorp Consul cluster as defined in the Consul Reference Architecture.
* [DNS Caching](https://learn.hashicorp.com/consul/day-2-operations/advanced-operations/dns-caching) - Enabling TTLs for DNS query caching
* [DNS Forwarding](https://www.consul.io/docs/guides/forwarding.html) - Forward DNS queries from Bind to Consul
* [External Services](https://www.consul.io/docs/guides/external.html) - This guide covers registering an external service. This allows using 3rd party services within the Consul framework.
* Federation ([Basic](https://www.consul.io/docs/guides/datacenters.html) and [Advanced](https://www.consul.io/docs/guides/areas.html)) - Configuring Consul to support multiple datacenters.
* [Geo Failover](https://www.consul.io/docs/guides/geo-failover.html) - This guide covers using [prepared queries](https://www.consul.io/api/query.html) to implement geo failover for services.
* [Minikube with Consul](https://www.consul.io/docs/guides/minikube.html) - In this guide, you'll start a local Kubernetes cluster with minikube, install Consul,and then deploy two custom services.
* [Leader Election](https://www.consul.io/docs/guides/leader-election.html) - The goal of this guide is to cover how to build client-side leader election using Consul.
* [Monitoring Consul with Telegraf](https://www.consul.io/docs/guides/monitoring-telegraf.html) - This guide demonstrates how to setup Consul for monitoring with Telegraf.
* [Network Segments](https://www.consul.io/docs/guides/segments.html) - Configuring Consul to support partial LAN connectivity using Network Segments.
* [Outage Recovery](https://learn.hashicorp.com/consul/day-2-operations/advanced-operations/outage) - This guide covers recovering a cluster that has become unavailable due to server failures.
* [Consul Reference Architecture](https://learn.hashicorp.com/consul/advanced/day-1-operations/reference-architecture) - This document provides recommended practices and a reference architecture, including system requirements, datacenter design, networking, and performance optimizations for Consul production deployments.
* [Semaphore](https://www.consul.io/docs/guides/semaphore.html) - This guide covers using the KV store to implement a semaphore.
* [Server Performance](https://www.consul.io/docs/guides/performance.html) - This guide covers minimum requirements for Consul servers as well as guidelines for running Consul servers in production.
* [Windows Service](https://www.consul.io/docs/guides/windows-guide.html) - This guide covers how to run Consul as a service on Windows.

# ACL Guides

We have several guides for setting up and configuring Consul's ACL system. They include how to bootstrap the ACL system in Consul version 1.4.0 and newer, how to bootstrap the ACL system in older versions of Consul, and how to migrate tokens from the legacy system to the new system in Consul 1.4.0.

Please select one of the following guides to get started.

## [»](https://www.consul.io/docs/guides/acl-index.html" \l "bootstrapping-guide)Bootstrapping Guide

Learn how to control access to Consul resources with this step-by-step [guide](https://learn.hashicorp.com/consul/advanced/day-1-operations/acl-guide) on bootstrapping the ACL system in Consul 1.4.0 and newer. This guide also includes additional steps for configuring the anonymous token, setting up agent-specific default tokens, and creating tokens for Consul UI use.

## [»](https://www.consul.io/docs/guides/acl-index.html" \l "legacy-acl-system)Legacy ACL System

The ACL system in Consul 1.3.1 and older is now called legacy. For information on bootstrapping the legacy system, ACL rules, and a general ACL system overview, read the legacy [guide](https://www.consul.io/docs/guides/acl-legacy.html).

## [»](https://www.consul.io/docs/guides/acl-index.html" \l "migrating-tokens)Migrating Tokens

[This guide](https://www.consul.io/docs/guides/acl-migrate-tokens.html) documents how to upgrade existing legacy tokens after upgrading to 1.4.0. It will briefly describe what changed, and then walk through the high-level migration process options, finally giving some specific examples of migration strategies. The new ACL system has improvements for the security and management of ACL tokens and policies.

# Running Connect in Production

Consul Connect can secure all inter-service communication via mutual TLS. It's designed to work with [minimal configuration out of the box](https://learn.hashicorp.com/consul/getting-started/connect), but completing the [security checklist](https://www.consul.io/docs/connect/security.html) and understanding the [Consul security model](https://www.consul.io/docs/internals/security.html) are prerequisites for production deployments.

This guide aims to walk through the steps required to ensure the security guarantees hold.

We assume a cluster is already running with an appropriate number of servers and clients and that other reference material like the [deployment](https://www.consul.io/docs/guides/deployment.html) and [performance](https://www.consul.io/docs/guides/performance.html) guides have been followed.

In practical deployments it may be necessary to incrementally adopt Connect service-by-service. In this case some or all of the advice below may not apply during the transition but should give a good understanding on which security properties have been sacrificed in the interim. The final deployment goal should be to end up compliant with all the advice below.

The steps we need to get to a secure Connect cluster are:

1. [Configure ACLs](https://www.consul.io/docs/guides/connect-production.html#configure-acls)
2. [Configure Agent Transport Encryption](https://www.consul.io/docs/guides/connect-production.html#configure-agent-transport-encryption)
3. [Bootstrap Certificate Authority](https://www.consul.io/docs/guides/connect-production.html#bootstrap-certificate-authority)
4. [Setup Host Firewall](https://www.consul.io/docs/guides/connect-production.html#setup-host-firewall)
5. [Configure Service Instances](https://www.consul.io/docs/guides/connect-production.html#configure-service-instances)

## [»](https://www.consul.io/docs/guides/connect-production.html" \l "configure-acls)Configure ACLs

Consul Connect's security is based on service identity. In practice the identity of the service is only enforcible with sufficiently restrictive ACLs.

This section will not replace reading the full [ACL guide](https://www.consul.io/docs/guides/acl.html) but will highlight the specific requirements Connect relies on to ensure it's security properties.

A service's identity, in the form of an x.509 certificate, will only be issued to an API client that has service:writepermission for that service. In other words, any client that has permission to register an instance of a service will be able to identify as that service and access all of the resources that that service is allowed to access.

A secure ACL setup must meet these criteria:

1. [**ACL default policy**](https://www.consul.io/docs/agent/options.html#acl_default_policy)**must be deny.** It is technically sufficient to keep the default policy of allow but add an explicit ACL denying anonymous service:write. Note however that in this case the Connect intention graph will also default to allow and explicit deny intentions will be needed to restrict service access. Also note that explicit rules to limit who can manage intentions are necessary in this case. It is assumed for the remainder of this guide that ACL policy defaults to deny.
2. **Each service must have a distinct ACL token** that is restricted to service:write only for the named service. Current Consul ACLs only support prefix matching but in a near-future release we will allow exact name matching. It is possible for all instances of the service to share the same token although best practices is for each instance to get a unique token as described below.

### [»](https://www.consul.io/docs/guides/connect-production.html" \l "fine-grained-enforcement)Fine Grained Enforcement

Connect intentions manage access based only on service identity so it is sufficient for ACL tokens to only be unique per service and shared between instances.

It is much better though if ACL tokens are unique per service instance because it limits the blast radius of a compromise.

A future release of Connect will support revoking specific certificates that have been issued. For example if a single node in a datacenter has been compromised, it will be possible to find all certificates issued to the agent on that node and revoke them. This will block all access to the intruder without taking instances of the service(s) on other nodes offline too.

While this will work with service-unique tokens, there is nothing stopping an attacker from obtaining certificates while spoofing the agent ID or other identifier – these certificates will not appear to have been issued to the compromised agent and so will not be revoked.

If every service instance has a unique token however, it will be possible to revoke all certificates that were requested under that token. Assuming the attacker can only access the tokens present on the compromised host, this guarantees that any certificate they might have access to or requested directly will be revoked.

In practice, managing per-instance tokens requires automated ACL provisioning, for example using [HashiCorp's Vault](https://www.vaultproject.io/docs/secrets/consul/index.html).

## [»](https://www.consul.io/docs/guides/connect-production.html" \l "configure-agent-transport-encryption)Configure Agent Transport Encryption

Consul's gossip (UDP) and RPC (TCP) communications need to be encrypted otherwise attackers may be able to see ACL tokens while in flight between the server and client agents (RPC) or between client agent and application (HTTP). Certificate private keys never leave the host they are used on but are delivered to the application or proxy over local HTTP so local agent traffic should be encrypted where potentially untrusted parties might be able to observe localhost agent API traffic.

Follow the [encryption documentation](https://www.consul.io/docs/agent/encryption.html) to ensure both gossip encryption and RPC/HTTP TLS are configured securely.

For now client and server TLS certificates are still managed by manual configuration. In the future we plan to automate more of that with the same mechanisms Connect offers to user applications.

## [»](https://www.consul.io/docs/guides/connect-production.html" \l "bootstrap-certificate-authority)Bootstrap Certificate Authority

Consul Connect comes with a built in Certificate Authority (CA) that will bootstrap by default when you first enable Connect on your servers.

To use the built-in CA, enable it in the server's configuration.

connect {

enabled = true

}

This config change requires a restart which you can perform one server at a time to maintain availability in an existing cluster.

As soon as a server that has Connect enabled becomes the leader, it will bootstrap a new CA and generate it's own private key which is written to the Raft state.

Alternatively, an external private key can be provided via the [CA configuration](https://www.consul.io/docs/connect/ca.html#specifying-a-private-key-and-root-certificate).

### [»](https://www.consul.io/docs/guides/connect-production.html" \l "external-cas)External CAs

Connect has been designed with a pluggable CA component so external CAs can be integrated. We will expand the external CA systems that are supported in the future and will allow seamless online migration to a different CA or bootstrapping with an external CA.

For production workloads we recommend using [Vault or another external CA](https://www.consul.io/docs/connect/ca.html#external-ca-certificate-authority-providers) once available such that the root key is not stored within Consul state at all.

## [»](https://www.consul.io/docs/guides/connect-production.html" \l "setup-host-firewall)Setup Host Firewall

In order to enable inbound connections to connect proxies, you may need to configure host or network firewalls to allow incoming connections to proxy ports.

In addition to Consul agent's [communication ports](https://www.consul.io/docs/agent/options.html#ports) any [proxies](https://www.consul.io/docs/connect/proxies.html) will need to have ports open to accept incoming connections.

If using [sidecar service registration](https://www.consul.io/docs/connect/proxies/sidecar-service.html) Consul will by default assign ports from [a configurable range](https://www.consul.io/docs/agent/options.html#sidecar_min_port) the default range is 21000 - 21255. If this feature is used, the agent assumes all ports in that range are both free to use (no other processes listening on them) and are exposed in the firewall to accept connections from other service hosts.

It is possible to prevent automated port selection by [configuring sidecar\_min\_port and sidecar\_max\_port](https://www.consul.io/docs/agent/options.html#sidecar_min_port) to both be 0, forcing any sidecar service registrations to need an explicit port configured.

It then becomes the same problem as opening ports necessary for any other application and might be managed by configuration management or a scheduler.

## [»](https://www.consul.io/docs/guides/connect-production.html" \l "configure-service-instances)Configure Service Instances

With [necessary ACL tokens](https://www.consul.io/docs/guides/connect-production.html#configure-acls) in place, all service registrations need to have an appropriate ACL token present.

For on-disk configuration the token parameter of the service definition must be set.

For registration via the API [the token is passed in the request header](https://www.consul.io/api/index.html#acls) or by using the [Go client configuration](https://godoc.org/github.com/hashicorp/consul/api#Config).

For examples of proxy service definitions see the [proxy documentation](https://www.consul.io/docs/connect/proxies.html).

To avoid the overhead of a proxy, applications may [natively integrate](https://www.consul.io/docs/connect/native.html) with connect.

### [»](https://www.consul.io/docs/guides/connect-production.html" \l "protect-application-listener)Protect Application Listener

If using any kind of proxy for connect, the application must ensure no untrusted connections can be made to it's unprotected listening port. This is typically done by binding to localhost and only allowing loopback traffic, but may also be achieved using firewall rules or network namespacing.

# Using Connect with Envoy Proxy

Consul Connect has first class support for using [Envoy](https://www.envoyproxy.io/) as a proxy. This guide will walk through a working example on a local development machine that shows the moving parts.

For reference documentation on how the integration works and is configured, please see [Envoy](https://www.consul.io/docs/connect/proxies/envoy.html).

## [»](https://www.consul.io/docs/guides/connect-envoy.html" \l "setup-overview)Setup Overview

This guide will describe how to setup a development-mode Consul server and two Envoy proxies on a single machine using [Docker](https://www.docker.com/). The aim is to demonstrate a minimal working setup and the moving parts involved.

We choose to run in Docker since Envoy is only distributed as a Docker image so it's the quickest way to get a demo running. The same commands used here will work in just the same way outside of Docker if you build an Envoy binary yourself.

We'll start all containers using Docker's host network mode which is not a realistic simulation of a production setup, but makes the following steps much simpler.

We should end up with five containers running:

1. The Consul agent
2. An example TCP echo service as a destination
3. An Envoy sidecar proxy for the echo service
4. An Envoy sidecar proxy for the client service
5. An example client service (netcat)

## [»](https://www.consul.io/docs/guides/connect-envoy.html" \l "building-an-envoy-image)Building an Envoy Image

Starting Envoy requires a bootstrap configuration file that points Envoy to the local agent for discovering the rest of it's configuration. The Consul binary includes the [consul connect envoy command](https://www.consul.io/docs/commands/connect/envoy.html) which can generate the bootstrap configuration for Envoy and optionally run it directly.

Envoy's official Docker image can be used with Connect directly however it requires some additional steps to generate bootstrap configuration and inject it into the container.

Instead, we'll use Docker multi-stage builds (added in version 17.05) to make a local image that has both envoy and consul binaries.

We'll create a local Docker image to use that contains both binaries. First create a Dockerfile containing the following:

FROM consul:latest

FROM envoyproxy/envoy:v1.8.0

COPY --from**=**0 /bin/consul /bin/consul

ENTRYPOINT **[**"dumb-init", "consul", "connect", "envoy"**]**

This takes the Consul binary from the latest release image and copies it into a new image based on the official Envoy image.

This can be built locally with:

docker build -t consul-envoy .

We will use the consul-envoy image we just made to configure and run Envoy processes later.

## [»](https://www.consul.io/docs/guides/connect-envoy.html" \l "consul-agent-setup)Consul Agent Setup

Next we need a Consul agent. We'll work with a single Consul agent in -dev mode for simplicity.

**Note:** -dev mode enables the gRPC server on port 8502 by default. For a production agent you'll need to [explicitly configure the gRPC port](https://www.consul.io/docs/agent/options.html#grpc_port).

In order to start a proxy instance, a [proxy service definition](https://www.consul.io/docs/connect/proxies.html) must exist on the local agent. We'll create one using the [sidecar service registration](https://www.consul.io/docs/connect/proxies/sidecar-service.html) syntax.

Create a config file called envoy\_demo.hcl containing the following service definitions.

services {

name **=** "client"

port **=** 8080

connect {

sidecar\_service {

proxy {

upstreams {

destination\_name **=** "echo"

local\_bind\_port **=** 9191

}

}

}

}

}

services {

name **=** "echo"

port **=** 9090

connect {

sidecar\_service {}

}

}

The Consul agent container can now be started with that configuration.

$ docker run --rm -d -v**$(**pwd**)**/envoy\_demo.hcl:/etc/consul/envoy\_demo.hcl \

--network host --name consul-agent consul:latest \

agent -dev -config-file /etc/consul/envoy\_demo.hcl

1c90f7fcc83f5390332d7a4fdda2f1bf74cf62762de9ea2f67cd5a09c0573641

Running with -d like this puts the container into the background so we can continue in the same terminal. Log output can be seen using the name we gave.

docker logs -f consul-agent

Note that the Consul agent has registered two services client and echo, but also registered two proxies client-sidecar-proxy and echo-sidecar-proxy. Next we'll need to run those services and proxies.

## [»](https://www.consul.io/docs/guides/connect-envoy.html" \l "running-the-echo-service)Running the Echo Service

Next we'll run the echo service. We can use an existing tcp echo utility image for this.

Start the echo service on port 9090 as registered before.

$ docker run -d --network host abrarov/tcp-echo --port 9090

1a0b0c569016d00aadc4fc2b2954209b32b510966083f2a9e17d3afc6d185d87

## [»](https://www.consul.io/docs/guides/connect-envoy.html" \l "running-the-proxies)Running the Proxies

We can now run "sidecar" proxy instances.

$ docker run --rm -d --network host --name echo-proxy \

consul-envoy -sidecar-for echo

3f213a3cf9b7583a194dd0507a31e0188a03fc1b6e165b7f9336b0b1bb2baccb

$ docker run --rm -d --network host --name client-proxy \

consul-envoy -sidecar-for client -admin-bind localhost:19001

d8399b54ee0c1f67d729bc4c8b6e624e86d63d2d9225935971bcb4534233012b

The -admin-bind flag on the second proxy command is needed because both proxies are running on the host network and so can't bind to the same port for their admin API (which cannot be disabled).

Again we can see the output using docker logs. To see more verbose information from Envoy you can add -- -l debugto the end of the commands above. This passes the -l (log level) option directly through to Envoy. With debug level logs you should see the config being delivered to the proxy in the output.

The [consul connect envoy command](https://www.consul.io/docs/commands/connect/envoy.html) here is connecting to the local agent, getting the proxy configuration from the proxy service registration and generating the required Envoy bootstrap configuration before execing the envoy binary directly to run it with the generated configuration.

Envoy uses the bootstrap configuration to connect to the local agent directly via gRPC and use it's xDS protocol to retrieve the actual configuration for listeners, TLS certificates, upstream service instances and so on. The xDS API allows the Envoy instance to watch for any changes so certificate rotations or changes to the upstream service instances are immediately sent to the proxy.

## [»](https://www.consul.io/docs/guides/connect-envoy.html" \l "running-the-client)Running the Client

Finally, we can see the connectivity by running a dummy "client" service. Rather than run a full service that itself can listen, we'll simulate the service with a simple netcat process that will only talk to the client-sidecar-proxy Envoy instance.

Recall that we configured the client sidecar with one declared "upstream" dependency (the echo service). In that declaration we also requested that the echo service should be exposed to the client on local port 9191.

This configuration causes the client-sidecar-proxy to start a TCP proxy listening on localhost:9191 and proxying to the echo service. Importantly, the listener will use the correct client service mTLS certificate to authorize the connection. It discovers the IP addresses of instances of the echo service via Consul service discovery.

We can now see this working if we run netcat.

$ docker run -ti --rm --network host gophernet/netcat localhost 9191

Hello World!

Hello World!

^C

## [»](https://www.consul.io/docs/guides/connect-envoy.html" \l "testing-authorization)Testing Authorization

To demonstrate that Connect is controlling authorization for the echo service, we can add an explicit deny rule.

$ docker run -ti --rm --network host consul:latest intention create -deny client echo

Created: client **=>** echo **(**deny**)**

Now, new connections will be denied. Depending on a few factors, netcat may not see the connection being closed but will not get a response from the service.

$ docker run -ti --rm --network host gophernet/netcat localhost 9191

Hello?

Anyone there?

^C

**Note:** Envoy will not currently re-authenticate already established TCP connections so if you still have the netcat terminal open from before, that will still be able to communicate with "echo". New connections should be denied though.

Removing the intention restores connectivity.

$ docker run -ti --rm --network host consul:latest intention delete client echo

Intention deleted.

$ docker run -ti --rm --network host gophernet/netcat localhost 9191

Hello?

Hello?

^C

## [»](https://www.consul.io/docs/guides/connect-envoy.html" \l "summary)Summary

In this guide we walked through getting a minimal working example of two plain TCP processes communicating over mTLS using Envoy sidecars configured by Connect.

For more details on how the Envoy integration works, please see the [Envoy reference documentation](https://www.consul.io/docs/connect/proxies/envoy.html).

To see how to get Consul Connect working in different environments like Kubernetes see the [Connect Getting Started](https://www.consul.io/docs/connect/index.html#getting-started-with-connect)overview.

# Consul with Containers

This guide describes critical aspects of operating a Consul cluster that's run inside containers. It primarily focuses on the Docker container runtime, but the principles largely apply to rkt, oci, and other container runtimes as well.

## [»](https://www.consul.io/docs/guides/consul-containers.html" \l "consul-official-docker-image)Consul Official Docker Image

Consul's official Docker images are tagged with version numbers. For example, docker pull consul:0.9.0 will pull the 0.9.0 Consul release image.

For major releases, make sure to read our [upgrade guides](https://www.consul.io/docs/upgrade-specific.html) before upgrading a cluster.

To get a development mode Consul instance running the latest version, run docker run consul.

More instructions on how to get started using this image are available at the [official Docker repository page](https://store.docker.com/images/consul)

## [»](https://www.consul.io/docs/guides/consul-containers.html" \l "data-directory-persistence)Data Directory Persistence

The container exposes its data directory, /consul/data, as a [volume](https://docs.docker.com/engine/tutorials/dockervolumes/). This is where Consul will store its persisted state.

For clients, this stores some information about the cluster and the client's services and health checks in case the container is restarted. If the volume on a client disappears, it doesn't affect cluster operations.

For servers, this stores the client information plus snapshots and data related to the consensus algorithm and other state like Consul's key/value store and catalog. **Servers need the volume's data to be available when restarting containers to recover from outage scenarios.** Therefore, care must be taken by operators to make sure that volumes containing consul cluster data are not destroyed during container restarts.

We also recommend taking additional backups via [consul snapshot](https://www.consul.io/docs/commands/snapshot.html), and storing them externally.

## [»](https://www.consul.io/docs/guides/consul-containers.html" \l "configuration)Configuration

The container has a Consul configuration directory set up at /consul/config and the agent will load any configuration files placed here by binding a volume or by composing a new image and adding files.

Note that the configuration directory is not exposed as a volume, and will not persist. Consul uses it only during start up and does not store any state there.

Configuration can also be added by passing the configuration JSON via environment variable CONSUL\_LOCAL\_CONFIG. Example:

$ docker run \

-d \

-e CONSUL\_LOCAL\_CONFIG**=**'{

"datacenter":"us\_west",

"server":true,

"enable\_debug":true

}' \

consul agent -server -bootstrap-expect**=**3

## [»](https://www.consul.io/docs/guides/consul-containers.html#networking)Networking

When running inside a container, Consul must be configured with an appropriate cluster address and client address. In some cases, it may also require configuring an advertise address.

* **Cluster Address** - The address at which other Consul agents may contact a given agent. This is also referred to as the bind address.
* **Client Address** - The address where other processes on the host contact Consul in order to make HTTP or DNS requests. Consider setting this to localhost or 127.0.0.1 to only allow processes on the same container to make HTTP/DNS requests.
* **Advertise Address** - The advertise address is used to change the address that we advertise to other nodes in the cluster. This defaults to the bind address. Consider using this if you use NAT in your environment, or in scenarios where you have a routable address that cannot be bound.

You will need to tell Consul what its cluster address is when starting so that it binds to the correct interface and advertises a workable interface to the rest of the Consul agents. There are two ways of doing this:

1. Environment Variables: Use the CONSUL\_CLIENT\_INTERFACE and CONSUL\_BIND\_INTERFACE environment variables. In the following example eth0 is the network interface of the container.
2. $ docker run \
3. -d \
4. -e CONSUL\_CLIENT\_INTERFACE**=**'eth0' \
5. -e CONSUL\_BIND\_INTERFACE**=**'eth0' \
6. consul agent -server -bootstrap-expect**=**3
7. Address Templates: You can declaratively specify the client and cluster addresses using the formats described in the [go-socketaddr](https://github.com/hashicorp/go-sockaddr) library. In the following example, the client and bind addresses are declaratively specified for the container network interface 'eth0'
8. $ docker run \
9. consul agent -server \
10. -client**=**'{{ GetInterfaceIP "eth0" }}' \
11. -bind**=**'{{ GetInterfaceIP "eth0" }}' \
12. -bootstrap-expect**=**3

## [»](https://www.consul.io/docs/guides/consul-containers.html" \l "stopping-and-restarting-containers)Stopping and Restarting Containers

The official Consul container supports stopping, starting, and restarting. To stop a container, run docker stop:

$ docker stop <container\_id>

To start a container, run docker start:

$ docker start <container\_id>

To do an in-memory reload, send a SIGHUP to the container:

$ docker kill --signal**=**HUP <container\_id>

As long as there are enough servers in the cluster to maintain [quorum](https://www.consul.io/docs/internals/consensus.html#deployment-table), Consul's [Autopilot](https://www.consul.io/docs/guides/autopilot.html) feature will handle removing servers whose containers were stopped. Autopilot's default settings are already configured correctly. If you override them, make sure that the following [settings](https://www.consul.io/docs/agent/options.html#autopilot) are appropriate.

* [cleanup\_dead\_servers](https://www.consul.io/docs/guides/consul-containers.html#cleanup_dead_servers) must be set to true to make sure that a stopped container is removed from the cluster.
* [last\_contact\_threshold](https://www.consul.io/docs/guides/consul-containers.html#last_contact_threshold) should be reasonably small, so that dead servers are removed quickly.
* [server\_stabilization\_time](https://www.consul.io/docs/guides/consul-containers.html#server_stabilization_time) should be sufficiently large (on the order of several seconds) so that unstable servers are not added to the cluster until they stabilize.

If the container running the currently-elected Consul server leader is stopped, a leader election will trigger. This event will cause a new Consul server in the cluster to assume leadership.

When a previously stopped server container is restarted using docker start <container\_id>, and it is configured to obtain a new IP, Autopilot will add it back to the set of Raft peers with the same node-id and the new IP address, after which it can participate as a server again.

## [»](https://www.consul.io/docs/guides/consul-containers.html" \l "known-issues)Known Issues

**All nodes changing IP addresses** Prior to Consul 0.9.3, Consul did not gracefully handle the situation where all nodes in the cluster running inside a container are restarted at the same time, and they all obtain new IP addresses. This has been [fixed](https://github.com/hashicorp/consul/issues/1580) since Consul 0.9.3, and requires "raft\_protocol" to be set to "3" in the configs in Consul 0.9.3. Consul 1.0 makes raft protocol 3 the default.

**Snapshot close error** Due to a [known issue](https://github.com/docker/libnetwork/issues/1204) with half close support in Docker, you will see an error message [ERR] consul: Failed to close snapshot: write tcp <source>-><destination>: write: broken pipe when saving snapshots. This does not affect saving and restoring snapshots when running in Docker.

# Consul Template

The Consul template tool provides a programmatic method for rendering configuration files from a variety of locations, including Consul KV. It is an ideal option for replacing complicated API queries that often require custom formatting. The template tool is based on Go templates and shares many of the same attributes.

Consul template is a useful tool with several uses, we will focus on two of it's use cases.

1. Update configuration files. The Consul template tool can be used to update service configuration files. A common use case is managing load balancer configuration files that need to be updated regularly in a dynamic infrastructure on machines many not be able to directly connect to the Consul cluster.
2. Discover data about the Consul cluster and service. It is possible to collect information about the services in your Consul cluster. For example, you could collect a list of all services running on the cluster or you could discover all service addresses for the Redis service. Note, this use case has limited scope for production.

In this guide we will briefly discuss how consul-template works, how to install it, and two use cases.

Before completing this guide, we assume some familiarity with [Consul KV](https://learn.hashicorp.com/consul/getting-started/kv) and [Go templates](https://golang.org/pkg/text/template/).

## [»](https://www.consul.io/docs/guides/consul-template.html" \l "introduction-to-consul-template)Introduction to Consul Template

Consul template is a simple, yet powerful tool. When initiated, it reads one or more template files and queries Consul for all data needed to render them. Typically, you run consul-template as a daemon which will fetch the initial values and then continue to watch for updates, re-rendering the template whenever there are relevant changes in the cluster. You can alternatively use the -once flag to fetch and render the template once which is useful for testing and setup scripts that are triggered by some other automation for example a provisioning tool. Finally, the template can also run arbitrary commands after the update process completes. For example, it can send the HUP signal to the load balancer service after a configuration change has been made.

The Consul template tool is flexible, it can fit into many different environments and workflows. Depending on the use-case, you may have a single consul-template instance on a handful of hosts or may need to run several instances on every host. Each consul-template process can manage multiple unrelated files though and will de-duplicate the fetches as needed if those files share data dependencies so it can reduce the load on Consul servers to share where possible.

## [»](https://www.consul.io/docs/guides/consul-template.html" \l "install-consul-template)Install Consul Template

For this guide, we are using a local Consul agent in development mode which can be started with consul agent -dev. To quickly set up a local Consul agent, refer to the getting started [guide](https://learn.hashicorp.com/consul/getting-started/install). The Consul agent must be running to complete all of the following steps.

The Consul template tool is not included with the Consul binary and will need to be installed separately. It can be installed from a precompiled binary or compiled from source. We will be installing the precompiled binary.

First, download the binary from the [Consul Template releases page](https://releases.hashicorp.com/consul-template/).

curl -O https://releases.hashicorp.com/consul-template/0.19.5/consul-template<\_version\_OS>.tgz

Next, extract the binary and move it into your $PATH.

tar -zxf consul-template<\_version\_OS>.tgz

To compile from source, please see the instructions in the [contributing section in GitHub](https://github.com/hashicorp/consul-template#contributing).

## [»](https://www.consul.io/docs/guides/consul-template.html" \l "use-case-consul-kv)Use Case: Consul KV

In this first use case example, we will render a template that pulls the HashiCorp address from Consul KV. To do this we will create a simple template that contains the HashiCorp address, run consul-template, add a value to Consul KV for HashiCorp's address, and finally view the rendered file.

First, we will need to create a template file find\_address.tpl to query Consul's KV store:

{{ key "/hashicorp/street\_address" }}

Next, we will run consul-template specifying both the template to use and the file to update.

$ consul-template -template "find\_address.tpl:hashicorp\_address.txt"

The consul-template process will continue to run until you kill it with CRTL+c. For now, we will leave it running.

Finally, open a new terminal so we can write data to the key in Consul using the command line interface.

$ consul kv put hashicorp/street\_address "101 2nd St"

Success! Data written to: hashicorp/street\_address

We can ensure the data was written by viewing the hashicorp\_address.txt file which will be located in the same directory where consul-template was run.

$ cat hashicorp\_address.txt

101 2nd St

If you update the key hashicorp/street\_address, you can see the changes to the file immediately. Go ahead and try consul kv put hashicorp/street\_address "22b Baker ST".

You can see that this simple process can have powerful implications. For example, it is possible to use this same process for updating your [HAProxy load balancer configuration](https://github.com/hashicorp/consul-template/blob/master/examples/haproxy.md).

You can now kill the consul-template process with CTRL+c.

## [»](https://www.consul.io/docs/guides/consul-template.html" \l "use-case-discover-all-services)Use Case: Discover All Services

In this use case example, we will discover all the services running in the Consul cluster. To follow along, you use the local development agent from the previous example.

First, we will need to create a new template all-services.tpl to query all services.

{{range services}}# {{.Name}}{{range service .Name}}

{{.Address}}{{end}}

{{end}}

Next, run Consul template specifying the template we just created and the -once flag. The -once flag will tell the process to run once and then quit.

$ consul-template -template**=**"all-services.tpl:all-services.txt" -once

If you complete this on your local development agent, you should still see the consul service when viewing all-services.txt.

# consul

127.0.0.7

On a development or production cluster, you would see a list of all the services. For example:

# consul

104.131.121.232

# redis

104.131.86.92

104.131.109.224

104.131.59.59

# web

104.131.86.92

104.131.109.224

104.131.59.59

## [»](https://www.consul.io/docs/guides/consul-template.html" \l "conclusion)Conclusion

In this guide we learned how to set up and use the Consul template tool. To see additional examples, refer to the examples folder in [GitHub](https://github.com/hashicorp/consul-template/tree/master/examples).

# Consul-AWS

[Consul-AWS](https://github.com/hashicorp/consul-aws/) syncs the services in an AWS Cloud Map namespace to a Consul datacenter. Consul services will be created in AWS Cloud Map and the other way around. This enables native service discovery across Consul and AWS Cloud Map. This guide will describe how to configure and how to start the sync.

## [»](https://www.consul.io/docs/guides/consul-aws.html#authentication)Authentication

consul-aws needs access to Consul and AWS for uni- and bidirectional sync.

For Consul, the process accepts both the standard CLI flags, -token and the environment variables CONSUL\_HTTP\_TOKEN. This should be set to a Consul ACL token if ACLs are enabled.

For AWS, consul-aws uses the default credential provider chain to find AWS credentials. The default provider chain looks for credentials in the following order: 1. Environment variables. 2. Shared credentials file. 3. If your application is running on an Amazon EC2 instance, IAM role for Amazon EC2.

## [»](https://www.consul.io/docs/guides/consul-aws.html#configuration)Configuration

There are two subcommands available on consul-aws:

* version: display version number
* sync-catalog: start syncing the catalogs

The version subcommand doesn’t do anything besides showing the version, so lets focus on sync-catalog. The following flags are available:

* A set of parameters to connect to your Consul Cluster like -http-addr, -token, -ca-file, -client-cert, and everything else you might need in order to do that

* [-aws-namespace-id](https://www.consul.io/docs/guides/consul-aws.html" \l "aws-namespace-id): The AWS namespace to sync with Consul services.

* [-aws-service-prefix](https://www.consul.io/docs/guides/consul-aws.html" \l "aws-service-prefix): A prefix to prepend to all services written to AWS from Consul. If this is not set then services will have no prefix.

* [-consul-service-prefix](https://www.consul.io/docs/guides/consul-aws.html" \l "consul-service-prefix): A prefix to prepend to all services written to Consul from AWS. If this is not set then services will have no prefix.

* [-to-aws](https://www.consul.io/docs/guides/consul-aws.html" \l "to-aws): If true, Consul services will be synced to AWS (defaults to false).

* [-to-consul](https://www.consul.io/docs/guides/consul-aws.html" \l "to-consul): If true, AWS services will be synced to Consul (defaults to false).

* [-aws-pull-interval](https://www.consul.io/docs/guides/consul-aws.html" \l "aws-pull-interval): The interval between fetching from AWS Cloud Map. Accepts a sequence of decimal numbers, each with optional fraction and a unit suffix, such as "300ms", "10s", "1.5m" (defaults to 30s).

* [-aws-dns-ttl](https://www.consul.io/docs/guides/consul-aws.html" \l "aws-dns-ttl): DNS TTL for services created in AWS Cloud Map in seconds (defaults to 60).

Independent of how you want to use consul-aws it needs to be able to connect to Consul and AWS. Apart from making sure you setup up authenticated access, -aws-namespace-id is mandatory.

## [»](https://www.consul.io/docs/guides/consul-aws.html" \l "syncing-consul-services-to-aws-cloud-map)Syncing Consul services to AWS Cloud Map

Assuming authenticated access is set up, there is little left to do before starting the sync. Using -to-aws command line flag will start the sync to AWS Cloud Map. If -aws-service-prefix is provided, every imported service from Consul will be prefixed. For example:

$ consul-aws -aws-namespace-id ns-hjrgt3bapp7phzff -to-aws -consul-service-prefix consul\_

At this point consul-aws will start importing services into AWS Cloud Map. A service in Consul named web will end up becoming consul\_web in AWS. The individual service instances from Consul will be created in AWS as well.

Services in AWS Cloud Map that were imported from Consul have the following properties:

* Description: “Imported from Consul”
* Record types: A and SRV
* DNS routing policy: Multivalue answer routing

## [»](https://www.consul.io/docs/guides/consul-aws.html" \l "syncing-aws-cloud-map-services-to-consul)Syncing AWS Cloud Map services to Consul

Similar to the previous chapter, there are two relevant flags: -to-consul to turn on the sync and optionally -consul-service-prefix to prefix every service imported into Consul. For example:

$ consul-aws -aws-namespace-id ns-hjrgt3bapp7phzff -to-consul -aws-service-prefix aws\_

At this point consul-aws will start importing services into Consul. A service in AWS named redis will end up becoming aws\_redis in Consul. The individual service instances from AWS will be created in Consul as well.

* Services in Consul that were imported from AWS Cloud Map have the following properties:
* Tag: aws
* Meta-Data: has aws as the source set, as well as the aws-id, the aws-namespace and every custom attribute the instance had in AWS Cloud Map
* Node: the node name is consul-aws

## [»](https://www.consul.io/docs/guides/consul-aws.html" \l "syncing-both-directions)Syncing both directions

To enable bidirectional sync only put together the previous two sections and provide -to-consul and -to-aws as well as optionally -aws-service-prefix and -consul-service-prefix:

$ consul-aws -aws-namespace-id ns-hjrgt3bapp7phzff -to-consul -aws-service-prefix aws\_ -to-aws -consul-service-prefix consul\_

At this point consul-aws will start importing services into Consul from AWS Cloud Map and from AWS Cloud Map to Consul.

## [»](https://www.consul.io/docs/guides/consul-aws.html#summary)Summary

At this point, either uni- or bidirectional sync is set up and service discovery is available across Consul and AWS seamlessly. If you haven’t enabled [ACL](https://www.consul.io/docs/guides/acl.html), now is a good time to read about it.

# Forwarding DNS

By default, DNS is served from port 53. On most operating systems, this requires elevated privileges. Instead of running Consul with an administrative or root account, it is possible to instead forward appropriate queries to Consul, running on an unprivileged port, from another DNS server or port redirect.

In this guide, we will demonstrate forwarding from [BIND](https://www.isc.org/downloads/bind/) as well as [dnsmasq](http://www.thekelleys.org.uk/dnsmasq/doc.html), [Unbound](https://www.unbound.net/), [systemd-resolved](https://www.freedesktop.org/wiki/Software/systemd/resolved/), and [iptables](http://www.netfilter.org/). For the sake of simplicity, BIND and Consul are running on the same machine in this example. For iptables the rules must be set on the same host as the Consul instance and relay hosts should not be on the same host or the redirects will intercept the traffic.

It is worth mentioning that, by default, Consul does not resolve DNS records outside the .consul. zone unless the[recursors](https://www.consul.io/docs/agent/options.html#recursors) configuration option has been set. As an example of how this changes Consul's behavior, suppose a Consul DNS reply includes a CNAME record pointing outside the .consul TLD. The DNS reply will only include CNAME records by default. By contrast, when recursors is set and the upstream resolver is functioning correctly, Consul will try to resolve CNAMEs and include any records (e.g. A, AAAA, PTR) for them in its DNS reply.

You can either do one of the following:

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "bind-setup)BIND Setup

First, you have to disable DNSSEC so that Consul and BIND can communicate. Here is an example of such a configuration:

options {

listen-on port 53 { 127.0.0.1; };

listen-on-v6 port 53 { ::1; };

directory "/var/named";

dump-file "/var/named/data/cache\_dump.db";

statistics-file "/var/named/data/named\_stats.txt";

memstatistics-file "/var/named/data/named\_mem\_stats.txt";

allow-query { localhost; };

recursion yes;

dnssec-enable no;

dnssec-validation no;

/\* Path to ISC DLV key \*/

bindkeys-file "/etc/named.iscdlv.key";

managed-keys-directory "/var/named/dynamic";

};

include "/etc/named/consul.conf";

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "zone-file)Zone File

Then we set up a zone for our Consul managed records in consul.conf:

zone "consul" IN {

type forward;

forward only;

forwarders { 127.0.0.1 port 8600; };

};

Here we assume Consul is running with default settings and is serving DNS on port 8600.

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "dnsmasq-setup)Dnsmasq Setup

Dnsmasq is typically configured via a dnsmasq.conf or a series of files in the /etc/dnsmasq.d directory. In Dnsmasq's configuration file (e.g. /etc/dnsmasq.d/10-consul), add the following:

# Enable forward lookup of the 'consul' domain:

server=/consul/127.0.0.1#8600

# Uncomment and modify as appropriate to enable reverse DNS lookups for

# common netblocks found in RFC 1918, 5735, and 6598:

#rev-server=0.0.0.0/8,127.0.0.1#8600

#rev-server=10.0.0.0/8,127.0.0.1#8600

#rev-server=100.64.0.0/10,127.0.0.1#8600

#rev-server=127.0.0.1/8,127.0.0.1#8600

#rev-server=169.254.0.0/16,127.0.0.1#8600

#rev-server=172.16.0.0/12,127.0.0.1#8600

#rev-server=192.168.0.0/16,127.0.0.1#8600

#rev-server=224.0.0.0/4,127.0.0.1#8600

#rev-server=240.0.0.0/4,127.0.0.1#8600

Once that configuration is created, restart the dnsmasq service.

Additional useful settings in dnsmasq to consider include (see [dnsmasq(8)](http://www.thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html) for additional details):

# Accept DNS queries only from hosts whose address is on a local subnet.

#local-service

# Don't poll /etc/resolv.conf for changes.

#no-poll

# Don't read /etc/resolv.conf. Get upstream servers only from the command

# line or the dnsmasq configuration file (see the "server" directive below).

#no-resolv

# Specify IP address(es) of other DNS servers for queries not handled

# directly by consul. There is normally one 'server' entry set for every

# 'nameserver' parameter found in '/etc/resolv.conf'. See dnsmasq(8)'s

# 'server' configuration option for details.

#server=1.2.3.4

#server=208.67.222.222

#server=8.8.8.8

# Set the size of dnsmasq's cache. The default is 150 names. Setting the

# cache size to zero disables caching.

#cache-size=65536

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "unbound-setup)Unbound Setup

Unbound is typically configured via a unbound.conf or a series of files in the /etc/unbound/unbound.conf.d directory. In an Unbound configuration file (e.g. /etc/unbound/unbound.conf.d/consul.conf), add the following:

#Allow insecure queries to local resolvers

server:

do-not-query-localhost: no

domain-insecure: "consul"

#Add consul as a stub-zone

stub-zone:

name: "consul"

stub-addr: 127.0.0.1@8600

You may have to add the following line to the bottom of your /etc/unbound/unbound.conf file for the new configuration to be included:

include: "/etc/unbound/unbound.conf.d/\*.conf"

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "systemd-resolved-setup)systemd-resolved Setup

systemd-resolved is typically configured with /etc/systemd/resolved.conf. To configure systemd-resolved to send queries for the consul domain to Consul, configure resolved.conf to contain the following:

DNS=127.0.0.1

Domains=~consul

The main limitation with this configuration is that the DNS field cannot contain ports. So for this to work either Consul must be [configured to listen on port 53](https://www.consul.io/docs/agent/options.html#dns_port) instead of 8600 or you can use iptables to map port 53 to 8600. The following iptables commands are sufficient to do the port mapping.

[root@localhost ~]# iptables -t nat -A OUTPUT -d localhost -p udp -m udp --dport 53 -j REDIRECT --to-ports 8600

[root@localhost ~]# iptables -t nat -A OUTPUT -d localhost -p tcp -m tcp --dport 53 -j REDIRECT --to-ports 8600

Binding to port 53 will usually require running either as a privileged user (or on Linux running with the CAP\_NET\_BIND\_SERVICE capability). If using the Consul docker image you will need to add the following to the environment to allow Consul to use the port: CONSUL\_ALLOW\_PRIVILEGED\_PORTS=yes

Note: With this setup, PTR record queries will still be sent out to the other configured resolvers in addition to Consul.

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "iptables-setup)iptables Setup

On Linux systems that support it, incoming requests and requests to the local host can use iptables to forward ports on the same machine without a secondary service. Since Consul, by default, only resolves the .consul TLD, it is especially important to use the recursors option if you wish the iptables setup to resolve for other domains. The recursors should not include the local host as the redirects would just intercept the requests.

The iptables method is suited for situations where an external DNS service is already running in your infrastructure and is used as the recursor or if you want to use an existing DNS server as your query endpoint and forward requests for the consul domain to the Consul server. In both of those cases you may want to query the Consul server but not need the overhead of a separate service on the Consul host.

[root@localhost ~]# iptables -t nat -A PREROUTING -p udp -m udp --dport 53 -j REDIRECT --to-ports 8600

[root@localhost ~]# iptables -t nat -A PREROUTING -p tcp -m tcp --dport 53 -j REDIRECT --to-ports 8600

[root@localhost ~]# iptables -t nat -A OUTPUT -d localhost -p udp -m udp --dport 53 -j REDIRECT --to-ports 8600

[root@localhost ~]# iptables -t nat -A OUTPUT -d localhost -p tcp -m tcp --dport 53 -j REDIRECT --to-ports 8600

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "macos-setup)macOS Setup

On macOS systems, you can use the macOS system resolver to point all .consul requests to consul. Just add a resolver entry in /etc/resolver/ to point at consul. documentation for this feature is available via: man5 resolver. To setup create a new file /etc/resolver/consul (you will need sudo/root access) and put in the file:

nameserver 127.0.0.1

port 8600

This is telling the macOS resolver daemon for all .consul TLD requests, ask 127.0.0.1 on port 8600.

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "testing)Testing

First, perform a DNS query against Consul directly to be sure that the record exists:

[root@localhost ~]# dig @localhost -p 8600 primary.redis.service.dc-1.consul. A

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.23.rc1.32.amzn1 <<>> @localhost primary.redis.service.dc-1.consul. A

; (1 server found)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11536

;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:

;primary.redis.service.dc-1.consul. IN A

;; ANSWER SECTION:

primary.redis.service.dc-1.consul. 0 IN A 172.31.3.234

;; Query time: 4 msec

;; SERVER: 127.0.0.1#53(127.0.0.1)

;; WHEN: Wed Apr 9 17:36:12 2014

;; MSG SIZE rcvd: 76

Then run the same query against your BIND instance and make sure you get a valid result:

[root@localhost ~]# dig @localhost -p 53 primary.redis.service.dc-1.consul. A

; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.23.rc1.32.amzn1 <<>> @localhost primary.redis.service.dc-1.consul. A

; (1 server found)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11536

;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:

;primary.redis.service.dc-1.consul. IN A

;; ANSWER SECTION:

primary.redis.service.dc-1.consul. 0 IN A 172.31.3.234

;; Query time: 4 msec

;; SERVER: 127.0.0.1#53(127.0.0.1)

;; WHEN: Wed Apr 9 17:36:12 2014

;; MSG SIZE rcvd: 76

If desired, verify reverse DNS using the same methodology:

[root@localhost ~]# dig @127.0.0.1 -p 8600 133.139.16.172.in-addr.arpa. PTR

; <<>> DiG 9.10.3-P3 <<>> @127.0.0.1 -p 8600 133.139.16.172.in-addr.arpa. PTR

; (1 server found)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3713

;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; WARNING: recursion requested but not available

;; QUESTION SECTION:

;133.139.16.172.in-addr.arpa. IN PTR

;; ANSWER SECTION:

133.139.16.172.in-addr.arpa. 0 IN PTR consul1.node.dc1.consul.

;; Query time: 3 msec

;; SERVER: 127.0.0.1#8600(127.0.0.1)

;; WHEN: Sun Jan 31 04:25:39 UTC 2016

;; MSG SIZE rcvd: 109

[root@localhost ~]# dig @127.0.0.1 +short -x 172.16.139.133

consul1.node.dc1.consul.

### [»](https://www.consul.io/docs/guides/forwarding.html" \l "troubleshooting)Troubleshooting

If you don't get an answer from your DNS server (e.g. BIND, Dnsmasq) but you do get an answer from Consul, your best bet is to turn on your DNS server's query log to see what's happening.

For BIND:

[root@localhost ~]# rndc querylog

[root@localhost ~]# tail -f /var/log/messages

The log may show errors like this:

error (no valid RRSIG) resolving

error (no valid DS) resolving

This indicates that DNSSEC is not disabled properly.

If you see errors about network connections, verify that there are no firewall or routing problems between the servers running BIND and Consul.

For Dnsmasq, see the log-queries configuration option and the USR1 signal.

# Registering an External Service

Very few infrastructures are entirely self-contained. Most rely on a multitude of external service providers. Consul supports this by allowing for the definition of external services, services that are not provided by a local node. There's also a companion project called [Consul ESM](https://github.com/hashicorp/consul-esm) which is a daemon that functions as an external service monitor that can help run health checks for external services.

Most services are registered in Consul through the use of a [service definition](https://www.consul.io/docs/agent/services.html). However, this approach registers the local node as the service provider. In the case of external services, we must instead register the service with the catalog rather than as part of a standard node service definition.

Once registered, the DNS interface will be able to return the appropriate A records or CNAME records for the service. The service will also appear in standard queries against the API. Consul must be configured with a list of [recursors](https://www.consul.io/docs/agent/options.html#recursors) for it to be able to resolve external service addresses.

Let us suppose we want to register a "search" service that is provided by "[www.google.com](http://www.google.com/)". We might accomplish that like so:

$ curl -X PUT -d '{"Datacenter": "dc1", "Node": "google",

"Address": "www.google.com",

"Service": {"Service": "search", "Port": 80}}'

http://127.0.0.1:8500/v1/catalog/register

Add an upstream DNS server to the list of recursors to Consul's configuration. Example with Google's public DNS server:text "recursors":["8.8.8.8"]

If we do a DNS lookup now, we can see the new search service:

; <<>> DiG 9.8.3-P1 <<>> @127.0.0.1 -p 8600 search.service.consul.

; (1 server found)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 13313

;; flags: qr aa rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:

;search.service.consul. IN A

;; ANSWER SECTION:

search.service.consul. 0 IN CNAME www.google.com.

www.google.com. 264 IN A 74.125.239.114

www.google.com. 264 IN A 74.125.239.115

www.google.com. 264 IN A 74.125.239.116

;; Query time: 41 msec

;; SERVER: 127.0.0.1#8600(127.0.0.1)

;; WHEN: Tue Feb 25 17:45:12 2014

;; MSG SIZE rcvd: 178

If at any time we want to deregister the service, we simply do:

$ curl -X PUT -d '{"Datacenter": "dc1", "Node": "google"}' http://127.0.0.1:8500/v1/catalog/deregister

This will deregister the google node along with all services it provides.

For more information, please see the [HTTP Catalog API](https://www.consul.io/api/catalog.html).

# Multiple Datacenters

## [»](https://www.consul.io/docs/guides/areas.html" \l "advanced-federation-with-network-areas)Advanced Federation with Network Areas

**ENTERPRISE**

This feature requires [Consul Enterprise](https://www.hashicorp.com/products/consul/)

One of the key features of Consul is its support for multiple datacenters. The [architecture](https://www.consul.io/docs/internals/architecture.html) of Consul is designed to promote a low coupling of datacenters so that connectivity issues or failure of any datacenter does not impact the availability of Consul in other datacenters. This means each datacenter runs independently, each having a dedicated group of servers and a private LAN [gossip pool](https://www.consul.io/docs/internals/gossip.html).

In general, data is not replicated between different Consul datacenters. When a request is made for a resource in another datacenter, the local Consul servers forward an RPC request to the remote Consul servers for that resource and return the results. If the remote datacenter is not available, then those resources will also not be available, but that won't otherwise affect the local datacenter. There are some special situations where a limited subset of data can be replicated, such as with Consul's built-in [ACL replication](https://www.consul.io/docs/guides/acl.html#outages-and-acl-replication) capability, or external tools like [consul-replicate](https://github.com/hashicorp/consul-replicate).

This guide covers the advanced form of federating Consul clusters using the new network areas capability added in [Consul Enterprise](https://www.hashicorp.com/products/consul/) version 0.8.0. For the basic form of federation available in the open source version of Consul, please see the [Basic Federation Guide](https://www.consul.io/docs/guides/datacenters.html) for more details.

## [»](https://www.consul.io/docs/guides/areas.html" \l "network-areas)Network Areas

Consul's [Basic Federation](https://www.consul.io/docs/guides/datacenters.html) support relies on all Consul servers in all datacenters having full mesh connectivity via server RPC (8300/tcp) and Serf WAN (8302/tcp and 8302/udp). Securing this setup requires TLS in combination with managing a gossip keyring. With massive Consul deployments, it becomes tricky to support a full mesh with all Consul servers, and to manage the keyring.

Consul Enterprise version 0.8.0 added support for a new federation model based on operator-created network areas. Network areas specify a relationship between a pair of Consul datacenters. Operators create reciprocal areas on each side of the relationship and then join them together, so a given Consul datacenter can participate in many areas, even when some of the peer areas cannot contact each other. This allows for more flexible relationships between Consul datacenters, such as hub/spoke or more general tree structures. Traffic between areas is all performed via server RPC (8300/tcp) so it can be secured with just TLS.

Currently, Consul will only route RPC requests to datacenters it is immediately adjacent to via an area (or via the WAN), but future versions of Consul may add routing support.

The following can be used to manage network areas:

* [Network Areas HTTP Endpoint](https://www.consul.io/api/operator/area.html)
* [Network Areas CLI](https://www.consul.io/docs/commands/operator/area.html)

## [»](https://www.consul.io/docs/guides/areas.html" \l "network-areas-and-the-wan-gossip-pool)Network Areas and the WAN Gossip Pool

Networks areas can be used alongside the Consul's [Basic Federation](https://www.consul.io/docs/guides/datacenters.html) model and the WAN gossip pool. This helps ease migration, and clusters like the [primary datacenter](https://www.consul.io/docs/agent/options.html#primary_datacenter) are more easily managed via the WAN because they need to be available to all Consul datacenters.

A peer datacenter can connected via the WAN gossip pool and a network area at the same time, and RPCs will be forwarded as long as servers are available in either.

## [»](https://www.consul.io/docs/guides/areas.html" \l "getting-started)Getting Started

To get started, follow the [bootstrapping guide](https://www.consul.io/docs/guides/bootstrapping.html) to start each datacenter. After bootstrapping, we should have two datacenters now which we can refer to as dc1 and dc2. Note that datacenter names are opaque to Consul; they are simply labels that help human operators reason about the Consul clusters.

A compatible pair of areas must be created in each datacenter:

(dc1) $ consul operator area create -peer-datacenter=dc2

Created area "cbd364ae-3710-1770-911b-7214e98016c0" with peer datacenter "dc2"!

(dc2) $ consul operator area create -peer-datacenter=dc1

Created area "2aea3145-f1e3-cb1d-a775-67d15ddd89bf" with peer datacenter "dc1"!

Now you can query for the members of the area:

(dc1) $ consul operator area members

Area Node Address Status Build Protocol DC RTT

cbd364ae-3710-1770-911b-7214e98016c0 node-1.dc1 127.0.0.1:8300 alive 0.8.0\_entrc1 2 dc1 0s

Consul will automatically make sure that all servers within the datacenter where the area was created are joined to the area using the LAN information. We need to join with at least one Consul server in the other datacenter to complete the area:

(dc1) $ consul operator area join -peer-datacenter=dc2 127.0.0.2

Address Joined Error

127.0.0.2 true (none)

With a successful join, we should now see the remote Consul servers as part of the area's members:

(dc1) $ consul operator area members

Area Node Address Status Build Protocol DC RTT

cbd364ae-3710-1770-911b-7214e98016c0 node-1.dc1 127.0.0.1:8300 alive 0.8.0\_entrc1 2 dc1 0s

cbd364ae-3710-1770-911b-7214e98016c0 node-2.dc2 127.0.0.2:8300 alive 0.8.0\_entrc1 2 dc2 581.649µs

Now we can route RPC commands in both directions. Here's a sample command to set a KV entry in dc2 from dc1:

(dc1) $ consul kv put -datacenter=dc2 hello world

Success! Data written to: hello

The DNS interface supports federation as well:

(dc1) $ dig @127.0.0.1 -p 8600 consul.service.dc2.consul

; <<>> DiG 9.8.3-P1 <<>> @127.0.0.1 -p 8600 consul.service.dc2.consul

; (1 server found)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49069

;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; WARNING: recursion requested but not available

;; QUESTION SECTION:

;consul.service.dc2.consul. IN A

;; ANSWER SECTION:

consul.service.dc2.consul. 0 IN A 127.0.0.2

;; Query time: 3 msec

;; SERVER: 127.0.0.1#8600(127.0.0.1)

;; WHEN: Wed Mar 29 11:27:35 2017

;; MSG SIZE rcvd: 59

There are a few networking requirements that must be satisfied for this to work. Of course, all server nodes must be able to talk to each other via their server RPC ports (8300/tcp). If service discovery is to be used across datacenters, the network must be able to route traffic between IP addresses across regions as well. Usually, this means that all datacenters must be connected using a VPN or other tunneling mechanism. Consul does not handle VPN or NAT traversal for you.

The [translate\_wan\_addrs](https://www.consul.io/docs/agent/options.html#translate_wan_addrs) configuration provides a basic address rewriting capability.

# Multiple Datacenters

## [»](https://www.consul.io/docs/guides/datacenters.html" \l "basic-federation-with-the-wan-gossip-pool)Basic Federation with the WAN Gossip Pool

One of the key features of Consul is its support for multiple datacenters. The [architecture](https://www.consul.io/docs/internals/architecture.html) of Consul is designed to promote a low coupling of datacenters so that connectivity issues or failure of any datacenter does not impact the availability of Consul in other datacenters. This means each datacenter runs independently, each having a dedicated group of servers and a private LAN [gossip pool](https://www.consul.io/docs/internals/gossip.html).

In general, data is not replicated between different Consul datacenters. When a request is made for a resource in another datacenter, the local Consul servers forward an RPC request to the remote Consul servers for that resource and return the results. If the remote datacenter is not available, then those resources will also not be available, but that won't otherwise affect the local datacenter. There are some special situations where a limited subset of data can be replicated, such as with Consul's built-in [ACL replication](https://www.consul.io/docs/guides/acl.html#outages-and-acl-replication) capability, or external tools like [consul-replicate](https://github.com/hashicorp/consul-replicate).

This guide covers the basic form of federating Consul clusters using a single WAN gossip pool, interconnecting all Consul servers. [Consul Enterprise](https://www.hashicorp.com/products/consul/) version 0.8.0 added support for an advanced multiple datacenter capability. Please see the[Advanced Federation Guide](https://www.consul.io/docs/guides/areas.html) for more details.

## [»](https://www.consul.io/docs/guides/datacenters.html#getting-started)Getting Started

To get started, follow the [bootstrapping guide](https://www.consul.io/docs/guides/bootstrapping.html) to start each datacenter. After bootstrapping, we should have two datacenters now which we can refer to as dc1 and dc2. Note that datacenter names are opaque to Consul; they are simply labels that help human operators reason about the Consul clusters.

To query the known WAN nodes, we use the [members](https://www.consul.io/docs/commands/members.html) command with the -wan parameter:

$ consul members -wan

...

This will provide a list of all known members in the WAN gossip pool. This should only contain server nodes. Client nodes send requests to a datacenter-local server, so they do not participate in WAN gossip. Client requests are forwarded by local servers to a server in the target datacenter as necessary.

The next step is to ensure that all the server nodes join the WAN gossip pool (include all the servers in all the datacenters):

$ consul join -wan <server 1> <server 2> ...

...

The [join](https://www.consul.io/docs/commands/join.html) command is used with the -wan flag to indicate we are attempting to join a server in the WAN gossip pool. As with LAN gossip, you only need to join a single existing member, and the gossip protocol will be used to exchange information about all known members. For the initial setup, however, each server will only know about itself and must be added to the cluster. Consul 0.8.0 added WAN join flooding, so if one Consul server in a datacenter joins the WAN, it will automatically join the other servers in its local datacenter that it knows about via the LAN.

Once the join is complete, the [members](https://www.consul.io/docs/commands/members.html) command can be used to verify that all server nodes gossiping over WAN.

We can also verify that both datacenters are known using the [HTTP Catalog API](https://www.consul.io/api/catalog.html#catalog_datacenters):

$ curl http://localhost:8500/v1/catalog/datacenters

["dc1", "dc2"]

As a simple test, you can try to query the nodes in each datacenter:

$ curl http://localhost:8500/v1/catalog/nodes?dc=dc1

...

$ curl http://localhost:8500/v1/catalog/nodes?dc=dc2

...

In order to persist the join information, the following can be added to the consul configuration in each of the servernodes in the cluster. For example, in dc1 server nodes: ... "retry\_join\_wan":[ "dc2-server-1", ... "dc2-server-N" ], ...

There are a few networking requirements that must be satisfied for this to work. Of course, all server nodes must be able to talk to each other. Otherwise, the gossip protocol as well as RPC forwarding will not work. If service discovery is to be used across datacenters, the network must be able to route traffic between IP addresses across regions as well. Usually, this means that all datacenters must be connected using a VPN or other tunneling mechanism. Consul does not handle VPN or NAT traversal for you.

Note that for RPC forwarding to work the bind address must be accessible from remote nodes. Configuring serf\_wan, advertise\_wan\_addr and translate\_wan\_addrs can lead to a situation where consul members -wan lists remote nodes but RPC operations fail with one of the following errors:

* [No path to datacenter](https://www.consul.io/docs/guides/datacenters.html" \l "no-path-to-datacenter)

* [rpc error getting client: failed to get conn: dial tcp <LOCAL\_ADDR>:0-><REMOTE\_ADDR>:<REMOTE\_RPC\_PORT>: i/o timeout](https://www.consul.io/docs/guides/datacenters.html" \l "rpc-error-getting-client-failed-to-get-conn-dial-tcp-lt-local_addr-gt-0-gt-lt-remote_addr-gt-lt-remote_rpc_port-gt-i-o-timeout)

The most likely cause of these errors is that bind\_addr is set to a private address preventing the RPC server from accepting connections across the WAN. Setting bind\_addr to a public address (or one that can be routed across the WAN) will resolve this issue. Be aware that exposing the RPC server on a public port should only be done **after** firewall rules have been established.

The [translate\_wan\_addrs](https://www.consul.io/docs/agent/options.html#translate_wan_addrs) configuration provides a basic address rewriting capability.

# Geo Failover

Within a datacenter, Consul provides automatic failover for services by omitting failed service instances from DNS lookups, and by providing service health information in APIs. When there are no more instances of a service available in the local datacenter, it can be challenging to implement failover policies to other datacenters because typically that logic would need to be written into each application.

Fortunately, Consul has a [prepared query](https://www.consul.io/api/query.html) capability that lets users define failover policies in a centralized way. It's easy to expose these to applications using Consul's DNS interface and it's also available to applications that consume Consul's APIs. These policies range from fully static lists of alternate datacenters to fully dynamic policies that make use of Consul's [network coordinate](https://www.consul.io/docs/internals/coordinates.html) subsystem to automatically determine the next best datacenter to fail over to based on network round trip time. Prepared queries can be made with policies specific to certain services and prepared query templates allow one policy to apply to many, or even all services, with just a small number of templates.

This guide shows how to build geo failover policies using prepared queries through a set of examples.

## [»](https://www.consul.io/docs/guides/geo-failover.html#prepared-queries)Prepared Queries

Prepared queries are objects that are defined at the datacenter level, similar to the values in Consul's KV store. They are created once and then invoked by applications to perform the query and get the latest results.

Here's an example request to create a prepared query:

$ curl \

--request POST \

--data \

'{

"Name": "api",

"Service": {

"Service": "api",

"Tags": ["v1.2.3"]

}

}' http://127.0.0.1:8500/v1/query

{"ID":"fe3b8d40-0ee0-8783-6cc2-ab1aa9bb16c1"}

This creates a prepared query called "api" that does a lookup for all instances of the "api" service with the tag "v1.2.3". This policy could be used to control which version of a "api" applications should be using in a centralized way. By [updating this prepared query](https://www.consul.io/api/query.html#update-prepared-query) to look for the tag "v1.2.4" applications could start to find the newer version of the service without having to reconfigure anything.

Applications can make use of this query in two ways. Since we gave the prepared query a name, they can simply do a DNS lookup for "api.query.consul" instead of "api.service.consul". Now with the prepared query, there's the additional filter policy working behind the scenes that the application doesn't have to know about. Queries can also be executed using the [prepared query execute API](https://www.consul.io/api/query.html#execute-prepared-query) for applications that integrate with Consul's APIs directly.

## [»](https://www.consul.io/docs/guides/geo-failover.html" \l "failover-policies)Failover Policies

Using the techniques in this section we will develop prepared queries with failover policies where simply changing application configurations to look up "api.query.consul" instead of "api.service.consul" via DNS will result in automatic geo failover to the next closest federated Consul datacenters, in order of increasing network round trip time.

Failover is just another policy choice for a prepared query, it works in the same manner as the previous example and is similarly transparent to applications. The failover policy is configured using the Failover structure, which contains two fields, both of which are optional, and determine what happens if no healthy nodes are available in the local datacenter when the query is executed.

* [NearestN](https://www.consul.io/docs/guides/geo-failover.html" \l "nearestn) (int: 0) - Specifies that the query will be forwarded to up to NearestN other datacenters based on their estimated network round trip time using [network coordinates](https://www.consul.io/docs/internals/coordinates.html).

* [Datacenters](https://www.consul.io/docs/guides/geo-failover.html" \l "datacenters) (array<string>: nil) - Specifies a fixed list of remote datacenters to forward the query to if there are no healthy nodes in the local datacenter. Datacenters are queried in the order given in the list.

The following sections show examples using these fields to implement different geo failover policies.

### [»](https://www.consul.io/docs/guides/geo-failover.html" \l "static-policy)Static Policy

A static failover policy includes a fixed list of datacenters to contact once there are no healthy instances in the local datacenter.

Here's the example from the introduction, expanded with a static failover policy:

$ curl \

--request POST \

--data \

'{

"Name": "api",

"Service": {

"Service": "api",

"Tags": ["v1.2.3"],

"Failover": {

"Datacenters": ["dc1", "dc2"]

}

}

}' http://127.0.0.1:8500/v1/query

{"ID":"fe3b8d40-0ee0-8783-6cc2-ab1aa9bb16c1"}

When this query is executed, such as with a DNS lookup to "api.query.consul", the following actions will occur:

1. Consul servers in the local datacenter will attempt to find healthy instances of the "api" service with the required tag.
2. If none are available locally, the Consul servers will make an RPC request to the Consul servers in "dc1" to perform the query there.
3. If none are available in "dc1", then an RPC will be made to the Consul servers in "dc2" to perform the query there.
4. Finally an error will be returned if none of these datacenters had any instances available.

### [»](https://www.consul.io/docs/guides/geo-failover.html" \l "dynamic-policy)Dynamic Policy

In a complex federated environment with many Consul datacenters, it can be cumbersome to set static failover policies, so Consul offers a dynamic option based on Consul's [network coordinate](https://www.consul.io/docs/internals/coordinates.html) subsystem. Consul continuously maintains an estimate of the network round trip time from the local datacenter to the servers in other datacenters it is federated with. Each server uses the median round trip time from itself to the servers in the remote datacenter. This means that failover can simply try other remote datacenters in order of increasing network round trip time, and if datacenters come and go, or experience network issues, this order will adjust automatically.

Here's the example from the introduction, expanded with a dynamic failover policy:

$ curl \

--request POST \

--data \

'{

"Name": "api",

"Service": {

"Service": "api",

"Tags": ["v1.2.3"],

"Failover": {

"NearestN": 2

}

}

}' http://127.0.0.1:8500/v1/query

{"ID":"fe3b8d40-0ee0-8783-6cc2-ab1aa9bb16c1"}

This query is resolved in a similar fashion to the previous example, except the choice of "dc1" or "dc2", or possibly some other datacenter, is made automatically.

### [»](https://www.consul.io/docs/guides/geo-failover.html" \l "hybrid-policy)Hybrid Policy

It is possible to combine Datacenters and NearestN in the same policy. The NearestN queries will be performed first, followed by the list given by Datacenters. A given datacenter will only be queried one time during a failover, even if it is selected by both NearestN and is listed in Datacenters. This is useful for allowing a limited number of round trip-based attempts, followed by a static configuration for some known datacenter to failover to.

## [»](https://www.consul.io/docs/guides/geo-failover.html" \l "templates)Templates

For datacenters with many services, it can be cumbersome to define a prepared query to apply a geo failover policy for each service. Consul provides a [prepared query template](https://www.consul.io/api/query.html#prepared-query-templates) capability to allow one prepared query to apply to many, and even all, services.

Here's an example request to create a prepared query template that applies a dynamic geo failover policy to all services:

$ curl \

--request POST \

--data \

'{

"Name": "",

"Template": {

"Type": "name\_prefix\_match"

},

"Service": {

"Service": "${name.full}",

"Failover": {

"NearestN": 2

}

}

}' http://127.0.0.1:8500/v1/query

{"ID":"fe3b8d40-0ee0-8783-6cc2-ab1aa9bb16c1"}

Templates can match on prefixes or use full regular expressions to determine which services they match. In this case, we've chosen the name\_prefix\_match type and given it an empty name, which means that it will match any service. If multiple queries are registered, the most specific one will be selected, so it's possible to have a template like this as a catch-all, and then apply more specific policies to certain services.

With this one prepared query template in place, simply changing application configurations to look up "api.query.consul" instead of "api.service.consul" via DNS will result in automatic geo failover to the next closest federated Consul datacenters, in order of increasing network round trip time.

# Leader Election

This guide describes how to build client-side leader election using Consul. If you are interested in the leader election used internally by Consul, please refer to the [consensus protocol](https://www.consul.io/docs/internals/consensus.html) documentation instead.

There are a number of ways that leader election can be built, so our goal is not to cover all the possible methods. Instead, we will focus on using Consul's support for [sessions](https://www.consul.io/docs/internals/sessions.html). Sessions allow us to build a system that can gracefully handle failures.

**Note:** JSON output in this guide has been pretty-printed for easier reading. Actual values returned from the API will not be formatted.

## [»](https://www.consul.io/docs/guides/leader-election.html" \l "contending-nodes)Contending Nodes

Let's imagine we have a set of nodes who are attempting to acquire leadership for a given service. All nodes that are participating should agree on a given key to coordinate. A good pattern is simply:

service/<service name>/leader

We'll abbreviate this pattern as simply <key> for the rest of this guide.

The first step is to create a session using the [Session HTTP API](https://www.consul.io/api/session.html#session_create):

curl -X PUT -d '{"Name": "dbservice"}' \

http://localhost:8500/v1/session/create

This will return a JSON object containing the session ID:

{

"ID": "4ca8e74b-6350-7587-addf-a18084928f3c"

}

The next step is to acquire a session for a given key from this node using the PUT method on a [KV entry](https://www.consul.io/api/kv.html) with the ?acquire=<session> query parameter. The <body> of the PUT should be a JSON object representing the local node. This value is opaque to Consul, but it should contain whatever information clients require to communicate with your application (e.g., it could be a JSON object that contains the node's name and the application's port).

Attempt to acquire the <key>. This will look something like (note that <session> is the ID returned by the call to[/v1/session/create](https://www.consul.io/api/session.html#session_create)):

curl -X PUT -d <body> http://localhost:8500/v1/kv/<key>?acquire=<session>

This will either return true or false. If true, the lock has been acquired and the local node is now the leader. If false is returned, some other node has acquired the lock.

All nodes now remain in an idle waiting state. In this state, we watch for changes on <key>. This is because the lock may be released, the node may fail, etc. The leader must also watch for changes since its lock may be released by an operator or automatically released due to a false positive in the failure detector.

By default, the session makes use of only the gossip failure detector. That is, the session is considered held by a node as long as the default Serf health check has not declared the node unhealthy. Additional checks can be specified if desired.

Watching for changes is done via a blocking query against <key>. If we ever notice that the Session of the <key> is blank, there is no leader, and we should retry lock acquisition. Each attempt to acquire the key should be separated by a timed wait. This is because Consul may be enforcing a [lock-delay](https://www.consul.io/docs/internals/sessions.html).

If the leader ever wishes to step down voluntarily, this should be done by simply releasing the lock:

curl -X PUT http://localhost:8500/v1/kv/<key>?release=<session>

## [»](https://www.consul.io/docs/guides/leader-election.html" \l "discovering-a-leader)Discovering a Leader

Another common practice regarding leader election is for nodes to wish to identify the leader for a given service.

As with leader election, all nodes that are participating should agree on the key being used to coordinate. This key will be referred to as just key.

Clients have a very simple role, they simply read <key> to discover who the current leader is:

curl http://localhost:8500/v1/kv/<key>

[

{

"Session": "4ca8e74b-6350-7587-addf-a18084928f3c",

"Value": "Ym9keQ==",

"Flags": 0,

"Key": "<key>",

"LockIndex": 1,

"ModifyIndex": 29,

"CreateIndex": 29

}

]

If the key has no associated Session, then there is no leader. Otherwise, the value of the key will provide all the application-dependent information required as a Base64 encoded blob in the Value field.

You can query the [/v1/session/info](https://www.consul.io/api/session.html#session_info) endpoint to get details about the session:

curl http://localhost:8500/v1/session/info/4ca8e74b-6350-7587-addf-a18084928f3c

[

{

"LockDelay": 1.5e+10,

"Checks": [

"serfHealth"

],

"Node": "consul-primary-bjsiobmvdij6-node-lhe5ihreel7y",

"Name": "dbservice",

"ID": "4ca8e74b-6350-7587-addf-a18084928f3c",

"CreateIndex": 28

}

]

Clients should also watch the key using a blocking query for any changes. If the leader steps down or fails, the Sessionassociated with the key will be cleared. When a new leader is elected, the key value will also be updated.

Using the acquire param is optional. This means that if you use leader election to update a key, you must not update the key without the acquire parameter.

# Monitoring Consul with Telegraf

Consul makes available a range of metrics in various formats in order to measure the health and stability of a cluster, and diagnose or predict potential issues.

There are number of monitoring tools and options, but for the purposes of this guide we are going to use the [telegraf\_plugin](https://github.com/influxdata/telegraf/tree/master/plugins/inputs/consul) in conjunction with the Statsd protocol supported by Consul.

You can read the full breakdown of metrics with Consul in the [telemetry documentation](https://www.consul.io/docs/agent/telemetry.html)

## [»](https://www.consul.io/docs/guides/monitoring-telegraf.html" \l "configuring-telegraf)Configuring Telegraf

# Installing Telegraf

Installing Telegraf is straightforward on most Linux distributions. We recommend following the [official Telegraf installation documentation](https://docs.influxdata.com/telegraf/v1.6/introduction/installation/).

# Configuring Telegraf

Besides acting as a statsd agent, Telegraf can collect additional metrics about the host that the Consul agent is running on. Telegraf itself ships with a wide range of [input plugins](https://docs.influxdata.com/telegraf/v1.6/plugins/inputs/) to collect data from lots of sources for this purpose.

We're going to enable some of the most common ones to monitor CPU, memory, disk I/O, networking, and process status, as these are useful for debugging Consul cluster issues.

The telegraf.conf file starts with global options:

[agent]

interval = "10s"

flush\_interval = "10s"

omit\_hostname = false

We set the default collection interval to 10 seconds and ask Telegraf to include a host tag in each metric.

As mentioned above, Telegraf also allows you to set additional tags on the metrics that pass through it. In this case, we are adding tags for the server role and datacenter. We can then use these tags in Grafana to filter queries (for example, to create a dashboard showing only servers with the consul-server role, or only servers in the us-east-1 datacenter).

[global\_tags]

role = "consul-server"

datacenter = "us-east-1"

Next, we set up a statsd listener on UDP port 8125, with instructions to calculate percentile metrics and to parse DogStatsD-compatible tags, when they're sent:

[[inputs.statsd]]

protocol = "udp"

service\_address = ":8125"

delete\_gauges = true

delete\_counters = true

delete\_sets = true

delete\_timings = true

percentiles = [90]

metric\_separator = "\_"

parse\_data\_dog\_tags = true

allowed\_pending\_messages = 10000

percentile\_limit = 1000

The full reference to all the available statsd-related options in Telegraf is [here](https://github.com/influxdata/telegraf/tree/release-1.6/plugins/inputs/statsd).

Now we can configure inputs for things like CPU, memory, network I/O, and disk I/O. Most of them don't require any configuration, but make sure the interfaces list in inputs.net matches the interface names you see in ifconfig.

[[inputs.cpu]]

percpu = true

totalcpu = true

collect\_cpu\_time = false

[[inputs.disk]]

*# mount\_points = ["/"]*

*# ignore\_fs = ["tmpfs", "devtmpfs"]*

[[inputs.diskio]]

*# devices = ["sda", "sdb"]*

*# skip\_serial\_number = false*

[[inputs.kernel]]

*# no configuration*

[[inputs.linux\_sysctl\_fs]]

*# no configuration*

[[inputs.mem]]

*# no configuration*

[[inputs.net]]

interfaces = ["enp0s\*"]

[[inputs.netstat]]

*# no configuration*

[[inputs.processes]]

*# no configuration*

[[inputs.swap]]

*# no configuration*

[[inputs.system]]

*# no configuration*

Another useful plugin is the [procstat](https://github.com/influxdata/telegraf/tree/release-1.6/plugins/inputs/procstat) plugin, which reports metrics for processes you select:

[[inputs.procstat]]

pattern = "(consul)"

Telegraf even includes a [plugin](https://github.com/influxdata/telegraf/tree/release-1.6/plugins/inputs/consul) that monitors the health checks associated with the Consul agent, using Consul API to query the data.

It's important to note: the plugin itself will not report the telemetry, Consul will report those stats already using StatsD protocol.

[[inputs.consul]]

address = "localhost:8500"

scheme = "http"

## [»](https://www.consul.io/docs/guides/monitoring-telegraf.html" \l "telegraf-configuration-for-consul)Telegraf Configuration for Consul

Asking Consul to send telemetry to Telegraf is as simple as adding a telemetry section to your agent configuration:

{

"telemetry": {

"dogstatsd\_addr": "localhost:8125",

"disable\_hostname": **true**

}

}

As you can see, we only need to specify two options. The dogstatsd\_addr specifies the hostname and port of the statsd daemon.

Note that we specify DogStatsD format instead of plain statsd, which tells Consul to send [tags](https://docs.datadoghq.com/getting_started/tagging/) with each metric. Tags can be used by Grafana to filter data on your dashboards (for example, displaying only the data for which role=consul-server. Telegraf is compatible with the DogStatsD format and allows us to add our own tags too.

The second option tells Consul not to insert the hostname in the names of the metrics it sends to statsd, since the hostnames will be sent as tags. Without this option, the single metric consul.raft.apply would become multiple metrics:

consul.server1.raft.apply

consul.server2.raft.apply

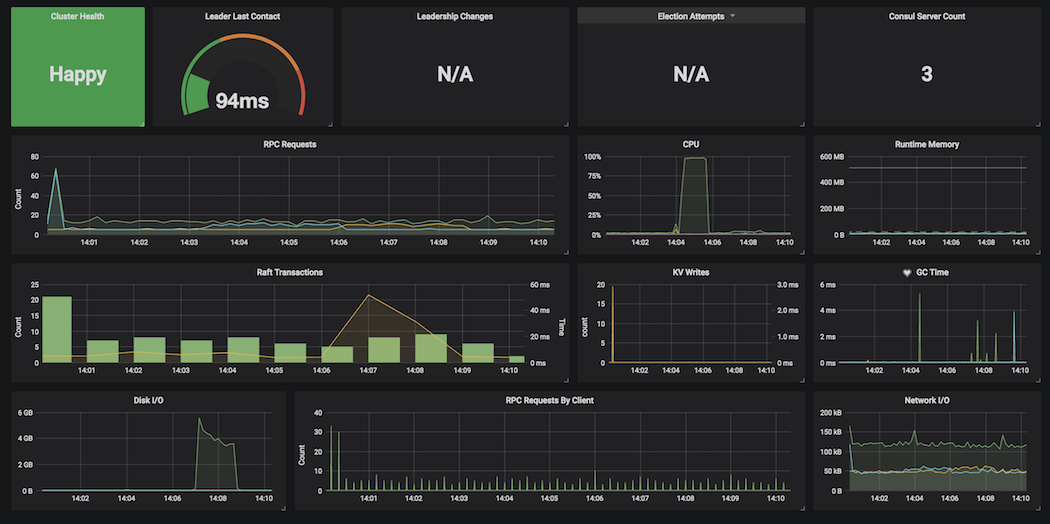
consul.server3.raft.apply

If you are using a different agent (e.g. Circonus, Statsite, or plain statsd), you may want to change this configuration, and you can find the configuration reference [here](https://www.consul.io/docs/agent/options.html#telemetry).

## [»](https://www.consul.io/docs/guides/monitoring-telegraf.html" \l "visualising-telegraf-consul-metrics)Visualising Telegraf Consul Metrics

There a number of ways of consuming the information from Telegraf. Generally they are visualised using a tool like [Grafana](https://www.influxdata.com/partners/grafana/) or [Chronograf](https://www.influxdata.com/time-series-platform/chronograf/).

Here is an example Grafana dashboard:

[](https://www.consul.io/assets/images/grafana-screenshot-600ab2df.png)

## [»](https://www.consul.io/docs/guides/monitoring-telegraf.html" \l "metric-aggregates-and-alerting-from-telegraf)Metric Aggregates and Alerting from Telegraf

### [»](https://www.consul.io/docs/guides/monitoring-telegraf.html#memory-usage)Memory usage

| **Metric Name** | **Description** |
| --- | --- |
| mem.total | Total amount of physical memory (RAM) available on the server. |
| mem.used\_percent | Percentage of physical memory in use. |
| swap.used\_percent | Percentage of swap space in use. |

**Why they're important:** Consul keeps all of its data in memory. If Consul consumes all available memory, it will crash. You should also monitor total available RAM to make sure some RAM is available for other processes, and swap usage should remain at 0% for best performance.

**What to look for:** If mem.used\_percent is over 90%, or if swap.used\_percent is greater than 0.

### [»](https://www.consul.io/docs/guides/monitoring-telegraf.html" \l "file-descriptors)File descriptors

| **Metric Name** | **Description** |
| --- | --- |
| linux\_sysctl\_fs.file-nr | Number of file handles being used across all processes on the host. |
| linux\_sysctl\_fs.file-max | Total number of available file handles. |

**Why it's important:** Practically anything Consul does -- receiving a connection from another host, sending data between servers, writing snapshots to disk -- requires a file descriptor handle. If Consul runs out of handles, it will stop accepting connections. See [the Consul FAQ](https://www.consul.io/docs/faq.html#q-does-consul-require-certain-user-process-resource-limits-) for more details.

By default, process and kernel limits are fairly conservative. You will want to increase these beyond the defaults.

**What to look for:** If file-nr exceeds 80% of file-max.

### [»](https://www.consul.io/docs/guides/monitoring-telegraf.html" \l "cpu-usage)CPU usage

| **Metric Name** | **Description** |
| --- | --- |
| cpu.user\_cpu | Percentage of CPU being used by user processes (such as Consul). |
| cpu.iowait\_cpu | Percentage of CPU time spent waiting for I/O tasks to complete. |

**Why they're important:** Consul is not particularly demanding of CPU time, but a spike in CPU usage might indicate too many operations taking place at once, and iowait\_cpu is critical -- it means Consul is waiting for data to be written to disk, a sign that Raft might be writing snapshots to disk too often.

**What to look for:** if cpu.iowait\_cpu greater than 10%.

### [»](https://www.consul.io/docs/guides/monitoring-telegraf.html" \l "network-activity-bytes-recived)Network activity - Bytes Recived

| **Metric Name** | **Description** |
| --- | --- |
| net.bytes\_recv | Bytes received on each network interface. |
| net.bytes\_sent | Bytes transmitted on each network interface. |

**Why they're important:** A sudden spike in network traffic to Consul might be the result of a misconfigured application client causing too many requests to Consul. This is the raw data from the system, rather than a specific Consul metric.

**What to look for:** Sudden large changes to the net metrics (greater than 50% deviation from baseline).

**NOTE:** The net metrics are counters, so in order to calculate rates (such as bytes/second), you will need to apply a function such as [non\_negative\_difference](https://docs.influxdata.com/influxdb/v1.5/query_language/functions/#non-negative-difference).

### [»](https://www.consul.io/docs/guides/monitoring-telegraf.html" \l "disk-activity)Disk activity

| **Metric Name** | **Description** |
| --- | --- |
| diskio.read\_bytes | Bytes read from each block device. |
| diskio.write\_bytes | Bytes written to each block device. |

**Why they're important:** If the Consul host is writing a lot of data to disk, such as under high volume workloads, there may be frequent major I/O spikes during leader elections. This is because under heavy load, Consul is checkpointing Raft snapshots to disk frequently.

It may also be caused by Consul having debug/trace logging enabled in production, which can impact performance.

Too much disk I/O can cause the rest of the system to slow down or become unavailable, as the kernel spends all its time waiting for I/O to complete.

**What to look for:** Sudden large changes to the diskio metrics (greater than 50% deviation from baseline, or more than 3 standard deviations from baseline).

**NOTE:** The diskio metrics are counters, so in order to calculate rates (such as bytes/second), you will need to apply a function such as [non\_negative\_difference](https://docs.influxdata.com/influxdb/v1.5/query_language/functions/#non-negative-difference).

# Network Segments

## [»](https://www.consul.io/docs/guides/segments.html" \l "partial-lan-connectivity-with-network-segments)Partial LAN Connectivity with Network Segments

**ENTERPRISE**

This feature requires [Consul Enterprise](https://www.hashicorp.com/products/consul/)

Many advanced Consul users have the need to run clusters with segmented networks, meaning that not all agents can be in a full mesh. This is usually the result of business policies enforced via network rules or firewalls. Prior to Consul 0.9.3 this was only possible through federation, which for some users is too heavyweight or expensive as it requires running multiple servers per segment.

By default, all Consul agents in one datacenter are part of a shared gossip pool over the LAN; this means that the partial connectivity caused by segmented networks would cause health flapping as nodes failed to communicate. In this guide we will cover the Network Segments feature, added in [Consul Enterprise](https://www.hashicorp.com/products/consul/) version 0.9.3, which allows users to configure Consul to support this kind of segmented network topology.

This guide will cover the basic configuration for setting up multiple segments, as well as how to configure a prepared query to limit service discovery to the services in the local agent's network segment.

## [»](https://www.consul.io/docs/guides/segments.html" \l "configuring-network-segments)Configuring Network Segments

All Consul agents are part of the default network segment, "", unless a segment is specified in their configuration. In a standard cluster setup all agents will normally be part of this default segment and as a result, part of one shared LAN gossip pool. Network segments can be used to break up the LAN gossip pool into multiple isolated smaller pools by specifying the configuration for segments on the servers. Each desired segment must be given a name and port, as well as optionally a custom bind and advertise address for that segment's gossip listener to bind to on the server.

A few things to note:

1. Servers will be a part of all segments they have been configured with. They are the common point linking the different segments together. The configured list of segments is specified by the [segments](https://www.consul.io/docs/agent/options.html#segments) option.
2. Client agents can only be part of one segment at a given time, specified by the [-segment](https://www.consul.io/docs/agent/options.html#_segment) option.
3. Clients can only join agents in the same segment as them. If they attempt to join a client in another segment, or the listening port of another segment on a server, they will get a segment mismatch error.

Once the servers have been configured with the correct segment info, the clients only need to specify their own segment in the [Agent Config](https://www.consul.io/docs/agent/options.html#_segment) and join by connecting to another agent within the same segment. If joining to a Consul server, client will need to provide the server's port for their segment along with the address of the server when performing the join (for example, consul agent -retry-join "consul.domain.internal:1234").

## [»](https://www.consul.io/docs/guides/segments.html#getting-started)Getting Started

To get started, follow the [bootstrapping guide](https://www.consul.io/docs/guides/bootstrapping.html) to start a server or group of servers, with the following section added to the configuration (you may need to adjust the bind/advertise addresses for your setup):

{

...

"segments": [

{"name": "alpha", "bind": "{{GetPrivateIP}}", "advertise": "{{GetPrivateIP}}", "port": 8303},

{"name": "beta", "bind": "{{GetPrivateIP}}", "advertise": "{{GetPrivateIP}}", "port": 8304}

]

}

You should see a log message on the servers for each segment's listener as the agent starts up:

2017/08/30 19:05:13 [INFO] serf: EventMemberJoin: server1.dc1 192.168.0.4

2017/08/30 19:05:13 [INFO] serf: EventMemberJoin: server1 192.168.0.4

2017/08/30 19:05:13 [INFO] consul: Started listener for LAN segment "alpha" on 192.168.0.4:8303

2017/08/30 19:05:13 [INFO] serf: EventMemberJoin: server1 192.168.0.4

2017/08/30 19:05:13 [INFO] consul: Started listener for LAN segment "beta" on 192.168.0.4:8304

2017/08/30 19:05:13 [INFO] serf: EventMemberJoin: server1 192.168.0.4

Running consul members should show the server as being part of all segments:

(server1) $ consul members

Node Address Status Type Build Protocol DC Segment

server1 192.168.0.4:8301 alive server 0.9.3+ent 2 dc1 <all>

Next, start a client agent in the 'alpha' segment, with -join set to the server's segment address/port for that segment:

(client1) $ consul agent ... -join 192.168.0.4:8303 -node client1 -segment alpha

After the join is successful, we should see the client show up by running the consul members command on the server again:

(server1) $ consul members

Node Address Status Type Build Protocol DC Segment

server1 192.168.0.4:8301 alive server 0.9.3+ent 2 dc1 <all>

client1 192.168.0.5:8301 alive client 0.9.3+ent 2 dc1 alpha

Now join another client in segment 'beta' and run the consul members command another time:

(client2) $ consul agent ... -join 192.168.0.4:8304 -node client2 -segment beta

(server1) $ consul members

Node Address Status Type Build Protocol DC Segment

server1 192.168.0.4:8301 alive server 0.9.3+ent 2 dc1 <all>

client1 192.168.0.5:8301 alive client 0.9.3+ent 2 dc1 alpha

client2 192.168.0.6:8301 alive client 0.9.3+ent 2 dc1 beta

If we pass the -segment flag when running consul members, we can limit the view to agents in a specific segment:

(server1) $ consul members -segment alpha

Node Address Status Type Build Protocol DC Segment

client1 192.168.0.5:8301 alive client 0.9.3+ent 2 dc1 alpha

server1 192.168.0.4:8303 alive server 0.9.3+ent 2 dc1 alpha

Using the consul catalog nodes command, we can filter on an internal metadata key, consul-network-segment, which stores the network segment of the node:

(server1) $ consul catalog nodes -node-meta consul-network-segment=alpha

Node ID Address DC

client1 4c29819c 192.168.0.5 dc1

With this metadata key, we can construct a [Prepared Query](https://www.consul.io/api/query.html) that can be used for DNS to return only services within the same network segment as the local agent.

First, register a service on each of the client nodes:

(client1) $ curl \

--request PUT \

--data '{"Name": "redis", "Port": 8000}' \

localhost:8500/v1/agent/service/register

(client2) $ curl \

--request PUT \

--data '{"Name": "redis", "Port": 9000}' \

localhost:8500/v1/agent/service/register

Next, write the following to query.json and create the query using the HTTP endpoint:

(server1) $ curl \

--request POST \

--data \

'{

"Name": "",

"Template": {

"Type": "name\_prefix\_match"

},

"Service": {

"Service": "${name.full}",

"NodeMeta": {"consul-network-segment": "${agent.segment}"}

}

}' localhost:8500/v1/query

{"ID":"6f49dd24-de9b-0b6c-fd29-525eca069419"}

Now, we can replace any dns lookups of the form <service>.service.consul with <service>.query.consul to look up only services within the same network segment:

**Client 1:**

(client1) $ dig @127.0.0.1 -p 8600 redis.query.consul SRV

; <<>> DiG 9.8.3-P1 <<>> @127.0.0.1 -p 8600 redis.query.consul SRV

; (1 server found)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3149

;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; WARNING: recursion requested but not available

;; QUESTION SECTION:

;redis.query.consul. IN SRV

;; ANSWER SECTION:

redis.query.consul. 0 IN SRV 1 1 8000 client1.node.dc1.consul.

;; ADDITIONAL SECTION:

client1.node.dc1.consul. 0 IN A 192.168.0.5

**Client 2:**

(client2) $ dig @127.0.0.1 -p 8600 redis.query.consul SRV

; <<>> DiG 9.8.3-P1 <<>> @127.0.0.1 -p 8600 redis.query.consul SRV

; (1 server found)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 3149

;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; WARNING: recursion requested but not available

;; QUESTION SECTION:

;redis.query.consul. IN SRV

;; ANSWER SECTION:

redis.query.consul. 0 IN SRV 1 1 9000 client2.node.dc1.consul.

;; ADDITIONAL SECTION:

client2.node.dc1.consul. 0 IN A 192.168.0.6

# Semaphore

This guide demonstrates how to implement a distributed semaphore using the Consul KV store. This is useful when you want to coordinate many services while restricting access to certain resources.

If you only need mutual exclusion or leader election, [this guide](https://www.consul.io/docs/guides/leader-election.html) provides a simpler algorithm that can be used instead.

There are a number of ways that a semaphore can be built, so our goal is not to cover all the possible methods. Instead, we will focus on using Consul's support for [sessions](https://www.consul.io/docs/internals/sessions.html). Sessions allow us to build a system that can gracefully handle failures.

**Note:** JSON output in this guide has been pretty-printed for easier reading. Actual values returned from the API will not be formatted.

## [»](https://www.consul.io/docs/guides/semaphore.html#contending-nodes)Contending Nodes

Let's imagine we have a set of nodes who are attempting to acquire a slot in the semaphore. All nodes that are participating should agree on three decisions: the prefix in the KV store used to coordinate, a single key to use as a lock, and a limit on the number of slot holders.

For the prefix we will be using for coordination, a good pattern is simply:

service/<service name>

We'll abbreviate this pattern as simply <prefix> for the rest of this guide.

The first step is for each contender to create a session. This is done using the [Session HTTP API](https://www.consul.io/api/session.html#session_create):

curl -X PUT -d '{"Name": "db-semaphore"}' \

http://localhost:8500/v1/session/create

This will return a JSON object contain the session ID:

{

"ID": "4ca8e74b-6350-7587-addf-a18084928f3c"

}

**Note:** Sessions by default only make use of the gossip failure detector. That is, the session is considered held by a node as long as the default Serf health check has not declared the node unhealthy. Additional checks can be specified at session creation if desired.

Next, we create a lock contender entry. Each contender creates a kv entry that is tied to a session. This is done so that if a contender is holding a slot and fails, its session is detached from the key, which can then be detected by the other contenders.

Create the contender key by doing an acquire on <prefix>/<session> via PUT. This is something like:

curl -X PUT -d <body> http://localhost:8500/v1/kv/<prefix>/<session>?acquire=<session>

body can be used to associate a meaningful value with the contender, such as its node’s name. This body is opaque to Consul but can be useful for human operators.

The <session> value is the ID returned by the call to [/v1/session/create](https://www.consul.io/api/session.html#session_create).

The call will either return true or false. If true, the contender entry has been created. If false, the contender node was not created; it's likely that this indicates a session invalidation.

The next step is to create a single key to coordinate which holders are currently reserving a slot. A good choice for this lock key is simply <prefix>/.lock. We will refer to this special coordinating key as <lock>.

This is done with:

curl -X PUT -d <body> http://localhost:8500/v1/kv/<lock>?cas=0

Since the lock is being created, a cas index of 0 is used so that the key is only put if it does not exist.

body should contain both the intended slot limit for the semaphore and the session ids of the current holders (initially only of the creator). A simple JSON body like the following works:

{

"Limit": 2,

"Holders": [

"<session>"

]

}

The current state of the semaphore is read by doing a GET on the entire <prefix>:

curl http://localhost:8500/v1/kv/<prefix>?recurse

Within the list of the entries, we should find two keys: the <lock> and the contender key ‘/’.

[

{

"LockIndex": 0,

"Key": "<lock>",

"Flags": 0,

"Value": "eyJMaW1pdCI6IDIsIkhvbGRlcnMiOlsiPHNlc3Npb24+Il19",

"Session": "",

"CreateIndex": 898,

"ModifyIndex": 901

},

{

"LockIndex": 1,

"Key": "<prefix>/<session>",

"Flags": 0,

"Value": null,

"Session": "<session>",

"CreateIndex": 897,

"ModifyIndex": 897

}

]

Note that the Value we embedded into <lock> is Base64 encoded when returned by the API.

When the <lock> is read and its Value is decoded, we can verify the Limit agrees with the Holders count. This is used to detect a potential conflict. The next step is to determine which of the current slot holders are still alive. As part of the results of the GET, we also have all the contender entries. By scanning those entries, we create a set of all the Sessionvalues. Any of the Holders that are not in that set are pruned. In effect, we are creating a set of live contenders based on the list results and doing a set difference with the Holders to detect and prune any potentially failed holders. In this example <session> is present in Holders and is attached to the key <prefix>/<session>, so no pruning is required.

If the number of holders after pruning is less than the limit, a contender attempts acquisition by adding its own session to the Holders list and doing a Check-And-Set update of the <lock>. This performs an optimistic update.

This is done with:

curl -X PUT -d <Updated Lock Body> http://localhost:8500/v1/kv/<lock>?cas=<lock-modify-index>

lock-modify-index is the latest ModifyIndex value known for <lock>, 901 in this example.

If this request succeeds with true, the contender now holds a slot in the semaphore. If this fails with false, then likely there was a race with another contender to acquire the slot.

To re-attempt the acquisition, we watch for changes on <prefix>. This is because a slot may be released, a node may fail, etc. Watching for changes is done via a blocking query against /kv/<prefix>?recurse.

Slot holders **must** continously watch for changes to <prefix> since their slot can be released by an operator or automatically released due to a false positive in the failure detector. On changes to <prefix> the lock’s Holders list must be re-checked to ensure the slot is still held. Additionally, if the watch fails to connect the slot should be considered lost.

This semaphore system is purely advisory. Therefore it is up to the client to verify that a slot is held before (and during) execution of some critical operation.

Lastly, if a slot holder ever wishes to release its slot voluntarily, it should be done by doing a Check-And-Set operation against <lock> to remove its session from the Holders object. Once that is done, both its contender key <prefix>/<session> and session should be deleted.

# Server Performance

Since Consul servers run a [consensus protocol](https://www.consul.io/docs/internals/consensus.html) to process all write operations and are contacted on nearly all read operations, server performance is critical for overall throughput and health of a Consul cluster. Servers are generally I/O bound for writes because the underlying Raft log store performs a sync to disk every time an entry is appended. Servers are generally CPU bound for reads since reads work from a fully in-memory data store that is optimized for concurrent access.

## [»](https://www.consul.io/docs/guides/performance.html" \l "minimum-server-requirements)Minimum Server Requirements

In Consul 0.7, the default server [performance parameters](https://www.consul.io/docs/agent/options.html#performance) were tuned to allow Consul to run reliably (but relatively slowly) on a server cluster of three [AWS t2.micro](https://aws.amazon.com/ec2/instance-types/) instances. These thresholds were determined empirically using a leader instance that was under sufficient read, write, and network load to cause it to permanently be at zero CPU credits, forcing it to the baseline performance mode for that instance type. Real-world workloads typically have more bursts of activity, so this is a conservative and pessimistic tuning strategy.

This default was chosen based on feedback from users, many of whom wanted a low cost way to run small production or development clusters with low cost compute resources, at the expense of some performance in leader failure detection and leader election times.

The default performance configuration is equivalent to this:

{

"performance": {

"raft\_multiplier": 5

}

}

## [»](https://www.consul.io/docs/guides/performance.html" \l "production-server-requirements)Production Server Requirements

When running Consul 0.7 and later in production, it is recommended to configure the server [performance parameters](https://www.consul.io/docs/agent/options.html#performance)back to Consul's original high-performance settings. This will let Consul servers detect a failed leader and complete leader elections much more quickly than the default configuration which extends key Raft timeouts by a factor of 5, so it can be quite slow during these events.

The high performance configuration is simple and looks like this:

{

"performance": {

"raft\_multiplier": 1

}

}

This value must take into account the network latency between the servers and the read/write load on the servers.

The value of raft\_multiplier is a scaling factor and directly affects the following parameters:

| **Param** | **Value** |  |
| --- | --- | --- |
| HeartbeatTimeout | 1000ms | default |
| ElectionTimeout | 1000ms | default |
| LeaderLeaseTimeout | 500ms | default |

So a scaling factor of 5 (i.e. raft\_multiplier: 5) updates the following values:

| **Param** | **Value** | **Calculation** |
| --- | --- | --- |
| HeartbeatTimeout | 5000ms | 5 x 1000ms |
| ElectionTimeout | 5000ms | 5 x 1000ms |
| LeaderLeaseTimeout | 2500ms | 5 x 500ms |

**NOTE** Wide networks with more latency will perform better with larger values of raft\_multiplier.

The trade off is between leader stability and time to recover from an actual leader failure. A short multiplier minimizes failure detection and election time but may be triggered frequently in high latency situations. This can cause constant leadership churn and associated unavailability. A high multiplier reduces the chances that spurious failures will cause leadership churn but it does this at the expense of taking longer to detect real failures and thus takes longer to restore cluster availability.

Leadership instability can also be caused by under-provisioned CPU resources and is more likely in environments where CPU cycles are shared with other workloads. In order for a server to remain the leader, it must send frequent heartbeat messages to all other servers every few hundred milliseconds. If some number of these are missing or late due to the leader not having sufficient CPU to send them on time, the other servers will detect it as failed and hold a new election.

It's best to benchmark with a realistic workload when choosing a production server for Consul. Here are some general recommendations:

* Consul will make use of multiple cores, and at least 2 cores are recommended.
* Spurious leader elections can be caused by networking issues between the servers or insufficient CPU resources. Users in cloud environments often bump their servers up to the next instance class with improved networking and CPU until leader elections stabilize, and in Consul 0.7 or later the [performance parameters](https://www.consul.io/docs/agent/options.html#performance) configuration now gives you tools to trade off performance instead of upsizing servers. You can use the [consul.raft.leader.lastContacttelemetry](https://www.consul.io/docs/agent/telemetry.html#last-contact) to observe how the Raft timing is performing and guide the decision to de-tune Raft performance or add more powerful servers.
* For DNS-heavy workloads, configuring all Consul agents in a cluster with the [allow\_stale](https://www.consul.io/docs/agent/options.html#allow_stale) configuration option will allow reads to scale across all Consul servers, not just the leader. Consul 0.7 and later enables stale reads for DNS by default. See [Stale Reads](https://www.consul.io/docs/guides/dns-cache.html#stale) in the [DNS Caching](https://www.consul.io/docs/guides/dns-cache.html) guide for more details. It's also good to set reasonable, non-zero [DNS TTL values](https://www.consul.io/docs/guides/dns-cache.html#ttl) if your clients will respect them.
* In other applications that perform high volumes of reads against Consul, consider using the [stale consistency mode](https://www.consul.io/api/index.html#consistency)available to allow reads to scale across all the servers and not just be forwarded to the leader.
* In Consul 0.9.3 and later, a new [limits](https://www.consul.io/docs/agent/options.html#limits) configuration is available on Consul clients to limit the RPC request rate they are allowed to make against the Consul servers. After hitting the limit, requests will start to return rate limit errors until time has passed and more requests are allowed. Configuring this across the cluster can help with enforcing a max desired application load level on the servers, and can help mitigate abusive applications.

## [»](https://www.consul.io/docs/guides/performance.html" \l "memory-requirements)Memory Requirements

Consul server agents operate on a working set of data comprised of key/value entries, the service catalog, prepared queries, access control lists, and sessions in memory. These data are persisted through Raft to disk in the form of a snapshot and log of changes since the previous snapshot for durability.

When planning for memory requirements, you should typically allocate enough RAM for your server agents to contain between 2 to 4 times the working set size. You can determine the working set size by noting the value ofconsul.runtime.alloc\_bytes in the [Telemetry data](https://www.consul.io/docs/agent/telemetry.html).

NOTE: Consul is not designed to serve as a general purpose database, and you should keep this in mind when choosing what data are populated to the key/value store.

## [»](https://www.consul.io/docs/guides/performance.html" \l "read-write-tuning)Read/Write Tuning

Consul is write limited by disk I/O and read limited by CPU. Memory requirements will be dependent on the total size of KV pairs stored and should be sized according to that data (as should the hard drive storage). The limit on a key’s value size is 512KB.

Consul is write limited by disk I/O and read limited by CPU.

For **write-heavy** workloads, the total RAM available for overhead must approximately be equal to

RAM NEEDED = number of keys \* average key size \* 2-3x

Since writes must be synced to disk (persistent storage) on a quorum of servers before they are committed, deploying a disk with high write throughput (or an SSD) will enhance performance on the write side. ([Documentation](https://www.consul.io/docs/agent/options.html#_data_dir))

For a **read-heavy** workload, configure all Consul server agents with the allow\_stale DNS option, or query the API with the stale [consistency mode](https://www.consul.io/api/index.html#consistency-modes). By default, all queries made to the server are RPC forwarded to and serviced by the leader. By enabling stale reads, any server will respond to any query, thereby reducing overhead on the leader. Typically, the stale response is 100ms or less from consistent mode but it drastically improves performance and reduces latency under high load.

If the leader server is out of memory or the disk is full, the server eventually stops responding, loses its election and cannot move past its last commit time. However, by configuring max\_stale and setting it to a large value, Consul will continue to respond to queries during such outage scenarios. ([max\_stale documentation](https://www.consul.io/docs/agent/options.html#max_stale)).

It should be noted that stale is not appropriate for coordination where strong consistency is important (i.e. locking or application leader election). For critical cases, the optional consistent API query mode is required for true linearizability; the trade off is that this turns a read into a full quorum write so requires more resources and takes longer.

**Read-heavy** clusters may take advantage of the [enhanced reading](https://www.consul.io/docs/enterprise/read-scale/index.html) feature (Enterprise) for better scalability. This feature allows additional servers to be introduced as non-voters. Being a non-voter, the server will still participate in data replication, but it will not block the leader from committing log entries.

Consul’s agents use network sockets for communicating with the other nodes (gossip) and with the server agent. In addition, file descriptors are also opened for watch handlers, health checks, and log files. For a **write heavy** cluster, the ulimit size must be increased from the default value (1024) to prevent the leader from running out of file descriptors.

To prevent any CPU spikes from a misconfigured client, RPC requests to the server should be [rate limited](https://www.consul.io/docs/agent/options.html#limits)

**NOTE** Rate limiting is configured on the client agent only.

In addition, two [performance indicators](https://www.consul.io/docs/agent/telemetry.html) — consul.runtime.alloc\_bytes and consul.runtime.heap\_objects — can help diagnose if the current sizing is not adequately meeting the load.

## [»](https://www.consul.io/docs/guides/performance.html" \l "connect-certificate-signing-cpu-limits)Connect Certificate Signing CPU Limits

If you enable [Connect](https://www.consul.io/docs/connect/index.html), the leader server will need to perform public key signing operations for every service instance in the cluster. Typically these operations are fast on modern hardware, however when the CA is changed or it's key rotated, the leader will face an influx of requests for new certificates for every service instance running.

While the client agents distribute these randomly over 30 seconds to avoid an immediate thundering herd, they don't have enough information to tune that period based on the number of certificates in use in the cluster so picking longer smearing results in artificially slow rotations for small clusters.

Smearing requests over 30s is sufficient to bring RPC load to a reasonable level in all but the very largest clusters, but the extra CPU load from cryptographic operations could impact the server's normal work. To limit that, Consul since 1.4.1 exposes two ways to limit the impact Certificate signing has on the leader [csr\_max\_per\_second](https://www.consul.io/docs/agent/options.html#ca_csr_max_per_second) and [csr\_max\_concurrent](https://www.consul.io/docs/agent/options.html#ca_csr_max_concurrent).

By default we set a limit of 50 per second which is reasonable on modest hardware but may be too low and impact rotation times if more than 1500 service instances are using Connect in the cluster. csr\_max\_per\_second is likely best if you have fewer than four cores available since a whole core being used by signing is likely to impact the server stability if it's all or a large portion of the cores available. The downside is that you need to capacity plan: how many service instances will need Connect certificates? What CSR rate can your server tolerate without impacting stability? How fast do you want CA rotations to process?

For larger production deployments, we generally recommend multiple CPU cores for servers to handle the normal workload. With four or more cores available, it's simpler to limit signing CPU impact with csr\_max\_concurrent rather than tune the rate limit. This effectively sets how many CPU cores can be monopolized by certificate signing work (although it doesn't pin that work to specific cores). In this case csr\_max\_per\_second should be disabled (set to 0).

For example if you have an 8 core server, setting csr\_max\_concurrent to 1 would allow you to process CSRs as fast as a single core can (which is likely sufficient for the very large clusters), without consuming all available CPU cores and impacting normal server work or stability.

# Overview

By using the sc command either on Powershell or the Windows command line, you can make Consul run as a service. For more details about the sc command the Windows page for [sc](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682107(v=vs.85).aspx) should help you get started.

Please remember to create a permanent directory for storing the configuration files. It is always advisable to start Consul with the -config-dir option.

The steps presented here , we assume that the user has launched Powershell with Adminstrator capabilities.

## [»](https://www.consul.io/docs/guides/windows-guide.html" \l "running-consul-run-as-a-service-on-windows)Running Consul run as a service on Windows

### [»](https://www.consul.io/docs/guides/windows-guide.html" \l "installing-consul-as-a-service)Installing Consul as a Service

Download the Consul binary for your architecture.

Use the sc command to create a Service named **Consul**, which starts in the dev mode.

sc.exe create "Consul" binPath="Path to the Consul.exe arg1 arg2 ...argN"

[SC] CreateService SUCCESS

If you get an output that is similar to the one above, then your service is registered with the Service manager.

If you get an error, please check that you have specified the proper path to the binary and check if you've entered the arguments correctly for the Consul service.

### [»](https://www.consul.io/docs/guides/windows-guide.html" \l "running-consul-as-a-service)Running Consul as a service

You have two ways to start the service.

* Go to the Windows Service Manager, and look for **Consul** under the service name. Click the start button to start the service.
* Using the sc command:
* sc.exe start "Consul"
* SERVICE\_NAME: Consul
* TYPE : 10 WIN32\_OWN\_PROCESS
* STATE : 4 RUNNING
* (STOPPABLE, NOT\_PAUSABLE, ACCEPTS\_SHUTDOWN)
* WIN32\_EXIT\_CODE : 0 (0x0)
* SERVICE\_EXIT\_CODE : 0 (0x0)
* CHECKPOINT : 0x0
* WAIT\_HINT : 0x0
* PID : 8008
* FLAGS :

The service automatically starts up during/after boot, so you don't need to launch Consul from the command-line again.

# Frequently Asked Questions

## [»](https://www.consul.io/docs/faq.html" \l "q-what-is-checkpoint-does-consul-call-home-)Q: What is Checkpoint? / Does Consul call home?

Consul makes use of a HashiCorp service called [Checkpoint](http://checkpoint.hashicorp.com/) which is used to check for updates and critical security bulletins. Only anonymous information, which cannot be used to identify the user or host, is sent to Checkpoint . An anonymous ID is sent which helps de-duplicate warning messages. This anonymous ID can be disabled. In fact, using the Checkpoint service is optional and can be disabled.

See [disable\_anonymous\_signature](https://www.consul.io/docs/agent/options.html#disable_anonymous_signature) and [disable\_update\_check](https://www.consul.io/docs/agent/options.html#disable_update_check).

## [»](https://www.consul.io/docs/faq.html" \l "q-does-consul-rely-on-udp-broadcast-or-multicast-)Q: Does Consul rely on UDP Broadcast or Multicast?

Consul uses the [Serf](https://www.serf.io/) gossip protocol which relies on TCP and UDP unicast. Broadcast and Multicast are rarely available in a multi-tenant or cloud network environment. For that reason, Consul and Serf were both designed to avoid any dependence on those capabilities.

## [»](https://www.consul.io/docs/faq.html" \l "q-is-consul-eventually-or-strongly-consistent-)Q: Is Consul eventually or strongly consistent?

Consul has two important subsystems, the service catalog and the gossip protocol. The service catalog stores all the nodes, service instances, health check data, ACLs, and KV information. It is strongly consistent, and replicated using the [consensus protocol](https://www.consul.io/docs/internals/consensus.html).

The [gossip protocol](https://www.consul.io/docs/internals/gossip.html) is used to track which nodes are part of the cluster and to detect a node or agent failure. This information is eventually consistent by nature. When the servers detects a change in membership, or receive a health update, they update the service catalog appropriately.

Because of this split, the answer to the question is subtle. Almost all client APIs interact with the service catalog and are strongly consistent. Updates to the catalog may come via the gossip protocol which is eventually consistent, meaning the current state of the catalog can lag behind until the state is reconciled.

## [»](https://www.consul.io/docs/faq.html" \l "q-are-failed-or-left-nodes-ever-removed-)Q: Are failed or left nodes ever removed?

To prevent an accumulation of dead nodes (nodes in either failed or left states), Consul will automatically remove dead nodes out of the catalog. This process is called reaping. This is currently done on a configurable interval of 72 hours. Reaping is similar to leaving, causing all associated services to be deregistered. Changing the reap interval for aesthetic reasons to trim the number of failed or left nodes is not advised (nodes in the failed or left state do not cause any additional burden on Consul).

## [»](https://www.consul.io/docs/faq.html" \l "q-does-consul-support-delta-updates-for-watchers-or-blocking-queries-)Q: Does Consul support delta updates for watchers or blocking queries?

Consul does not currently support sending a delta or a change only response to a watcher or a blocking query. The API simply allows for an edge-trigger return with the full result. A client should keep the results of their last read and compute the delta client side.

By design, Consul offloads this to clients instead of attempting to support the delta calculation. This avoids expensive state maintenance on the servers as well as race conditions between data updates and watch registrations.

## [»](https://www.consul.io/docs/faq.html" \l "q-what-network-ports-does-consul-use-)Q: What network ports does Consul use?

The [Ports Used](https://www.consul.io/docs/agent/options.html#ports) section of the Configuration documentation lists all ports that Consul uses.

## [»](https://www.consul.io/docs/faq.html" \l "q-does-consul-require-certain-user-process-resource-limits-)Q: Does Consul require certain user process resource limits?

There should be only a small number of open file descriptors required for a Consul client agent. The gossip layers perform transient connections with other nodes, each connection to the client agent (such as for a blocking query) will open a connection, and there will typically be connections to one of the Consul servers. A small number of file descriptors are also required for watch handlers, health checks, log files, and so on.

For a Consul server agent, you should plan on the above requirements and an additional incoming connection from each of the nodes in the cluster. This should not be the common case, but in the worst case if there is a problem with the other servers you would expect the other client agents to all connect to a single server and so preparation for this possibility is helpful.

The default ulimits are usually sufficient for Consul, but you should closely scrutinize your own environment's specific needs and identify the root cause of any excessive resource utilization before arbitrarily increasing the limits.

## [»](https://www.consul.io/docs/faq.html" \l "q-what-is-the-per-key-value-size-limitation-for-consul-39-s-key-value-store-)Q: What is the per-key value size limitation for Consul's key/value store?

The limit on a key's value size is 512KB. This is strictly enforced and an HTTP 413 status will be returned to any client that attempts to store more than that limit in a value. It should be noted that the Consul key/value store is not designed to be used as a general purpose database. See [Server Performance](https://www.consul.io/docs/guides/performance.html) for more details.

## [»](https://www.consul.io/docs/faq.html" \l "q-what-data-is-replicated-between-consul-datacenters-)Q: What data is replicated between Consul datacenters?

In general, data is not replicated between different Consul datacenters. When a request is made for a resource in another datacenter, the local Consul servers forward an RPC request to the remote Consul servers for that resource and return the results. If the remote datacenter is not available, then those resources will also not be available, but that won't otherwise affect the local datacenter. There are some special situations where a limited subset of data can be replicated, such as with Consul's built-in [ACL replication](https://www.consul.io/docs/guides/acl.html#outages-and-acl-replication) capability, or external tools like [consul-replicate](https://github.com/hashicorp/consul-replicate).

# Consul Enterprise

Consul Enterprise simplifies operations by automating workflows. It adds support for microservices deployments across complex network topologies. It also increases both scalability and resilience. Features include:

* [Automated Backups](https://www.consul.io/docs/enterprise/backups/index.html)
* [Automated Upgrades](https://www.consul.io/docs/enterprise/upgrades/index.html)
* [Enhanced Read Scalability](https://www.consul.io/docs/enterprise/read-scale/index.html)
* [Redundancy Zones](https://www.consul.io/docs/enterprise/redundancy/index.html)
* [Advanced Federation for Complex Network Topologies](https://www.consul.io/docs/enterprise/federation/index.html)
* [Network Segments](https://www.consul.io/docs/enterprise/network-segments/index.html)
* [Sentinel](https://www.consul.io/docs/enterprise/sentinel/index.html)

These features are part of [Consul Enterprise](https://www.hashicorp.com/consul.html).

## [»](https://www.consul.io/docs/enterprise/index.html" \l "licensing)Licensing

Licensing capabilities were added to Consul Enterprise v1.1.0. The license is set once for a datacenter and will automatically propagate to all nodes within the datacenter over a period of time scaled between 1 and 20 minutes depending on the number of nodes in the datacenter. The license can be set via the [API](https://www.consul.io/api/operator/license.html) or the [CLI](https://www.consul.io/docs/commands/license.html). When Consul is first started, a 30 minute temporary license is available to allow for time to license the datacenter. The license should be set within ten minutes of starting the first Consul process to allow time for the license to propagate.