# Consul首页、文档和下载 - 服务的注册和发现 - 开源中国社区

https://www.oschina.net/p/consul

Consul 简化了分布式环境中的服务的注册和发现流程，通过 HTTP 或者 DNS 接口发现。支持外部 SaaS 提供者等。

在线演示：http://demo.consul.io/ui/

使用示例：

admin@hashicorp: 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

admin@hashicorp:

# 使用consul实现分布式服务注册和发现[转] - 强子哥哥

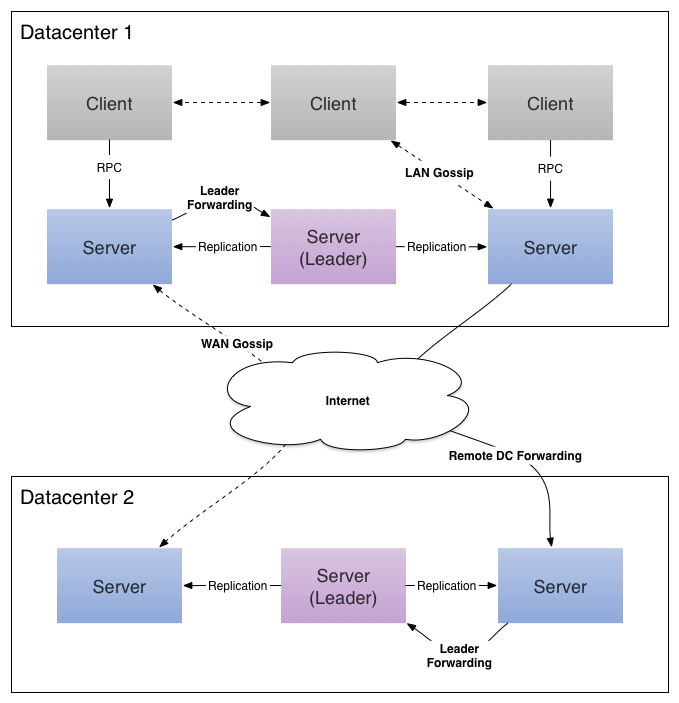
https://my.oschina.net/qiangzigege/blog/691443

[Consul](https://github.com/hashicorp/consul)是[HashiCorp](https://www.hashicorp.com/)公司推出的开源工具，用于实现分布式系统的服务发现与配置。与其他分布式服务注册与发现的方案，比如 [Airbnb](https://www.airbnb.com/)的SmartStack等相比，Consul的方案更“一站式”，内置了服务注册与发现框 架、分布一致性协议实现、健康检查、Key/Value存储、多数据中心方案，不再需要依赖其他工具（比如[ZooKeeper](http://tonybai.com/tag/zookeeper)等）。使用起来也较 为简单。Consul用[Golang](http://tonybai.com/tag/go)实现，因此具有天然可移植性(支持Linux、windows和Mac OS X)；安装包仅包含一个可执行文件，方便部署，与[Docker](http://tonybai.com/tag/docker)等轻量级容器可**无缝配合**。

本文是Consul的入门介绍，并用一些例子说明如何使用Consul实现服务的注册和发现。

**一、建立Consul Cluster**

要想利用Consul提供的服务实现服务的注册与发现，我们需要建立Consul Cluster。在Consul方案中，每个提供服务的节点上都要部署和运行Consul的agent，所有运行Consul agent节点的集合构成Consul Cluster。Consul agent有两种运行模式：Server和Client。这里的Server和Client只是Consul集群层面的区分，与搭建在Cluster之上 的应用服务无关。以Server模式运行的Consul agent节点用于维护Consul集群的状态，官方建议每个Consul Cluster至少有3个或以上的运行在Server mode的Agent，Client节点不限。



每个数据中心的Consul Cluster都会在运行于server模式下的agent节点中选出一个Leader节点，这个选举过程通过Consul实现的raft协议保证，多个 server节点上的Consul数据信息是强一致的。处于client mode的Consul agent节点比较简单，无状态，仅仅负责将请求转发给Server agent节点。

下面我们就来搭建一个实验Consul Cluster。

实验环境和节点角色如下：

n1(Ubuntu 14.04 x86\_64): 10.10.105.71  server mode  
n2(Ubuntu 12.04 x86\_64): 10.10.126.101 server mode    with Consul Web UI  
n3(Ubuntu 9.04 i386): 10.10.126.187    client mode

在三台主机上分别下载和安装Consul包，安装包很简单，只是包含一个可执行文件consul。在n2主机上还要下载一份Consul Web UI包，支持图形化展示Consul cluster中的节点状态和服务状态。

Consul Cluster的启动过程如下：

**n1主机：**

$ consul agent -server -bootstrap-expect 2 -data-dir /tmp/consul -node=n1 -bind=10.10.105.71 -dc=dc1  
==> WARNING: Expect Mode enabled, expecting 2 servers  
==> WARNING: It is highly recommended to set GOMAXPROCS higher than 1  
==> Starting Consul agent…  
==> Starting Consul agent RPC…  
==> Consul agent running!  
         Node name: 'n1'  
        Datacenter: 'dc1'  
            Server: true (bootstrap: false)  
       Client Addr: 127.0.0.1 (HTTP: 8500, HTTPS: -1, DNS: 8600, RPC: 8400)  
      Cluster Addr: 10.10.105.71 (LAN: 8301, WAN: 8302)  
    Gossip encrypt: false, RPC-TLS: false, TLS-Incoming: false  
             Atlas:

==> Log data will now stream in as it occurs:

    2015/07/03 09:18:25 [INFO] serf: EventMemberJoin: n1 10.10.105.71  
    2015/07/03 09:18:25 [INFO] serf: EventMemberJoin: n1.dc1 10.10.105.71  
    2015/07/03 09:18:25 [INFO] raft: Node at 10.10.105.71:8300 [Follower] entering Follower state  
    2015/07/03 09:18:25 [INFO] consul: adding server n1 (Addr: 10.10.105.71:8300) (DC: dc1)  
    2015/07/03 09:18:25 [INFO] consul: adding server n1.dc1 (Addr: 10.10.105.71:8300) (DC: dc1)  
    2015/07/03 09:18:25 [ERR] agent: failed to sync remote state: No cluster leader  
    2015/07/03 09:18:26 [WARN] raft: EnableSingleNode disabled, and no known peers. Aborting election.1

**n2主机：**

$ consul agent -server -bootstrap-expect 2 -data-dir /tmp/consul -node=n2 -bind=10.10.126.101 **-ui-dir ./dist**  -dc=dc1  
==> WARNING: Expect Mode enabled, expecting 2 servers  
==> WARNING: It is highly recommended to set GOMAXPROCS higher than 1  
==> Starting Consul agent…  
==> Starting Consul agent RPC…  
==> Consul agent running!  
         Node name: 'n2'  
        Datacenter: 'dc1'  
            Server: true (bootstrap: false)  
       Client Addr: 127.0.0.1 (HTTP: 8500, HTTPS: -1, DNS: 8600, RPC: 8400)  
      Cluster Addr: 10.10.126.101 (LAN: 8301, WAN: 8302)  
    Gossip encrypt: false, RPC-TLS: false, TLS-Incoming: false  
             Atlas:

==> Log data will now stream in as it occurs:

    2015/07/03 11:30:32 [INFO] serf: EventMemberJoin: n2 10.10.126.101  
    2015/07/03 11:30:32 [INFO] serf: EventMemberJoin: n2.dc1 10.10.126.101  
    2015/07/03 11:30:32 [INFO] raft: Node at 10.10.126.101:8300 [Follower] entering Follower state  
    2015/07/03 11:30:32 [INFO] consul: adding server n2 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 11:30:32 [INFO] consul: adding server n2.dc1 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 11:30:32 [ERR] agent: failed to sync remote state: No cluster leader  
    2015/07/03 11:30:33 [WARN] raft: EnableSingleNode disabled, and no known peers. Aborting election.

从两个server agent的启动日志可以看出，n1、n2启动后并不知道集群其他节点的存在。以n1为例，通过consul members和consul info查看当前agent状态：

$ consul members  
Node  Address            Status  Type    Build  Protocol  DC  
n1    10.10.105.71:8301  alive   server  0.5.2  2         dc1

$ consul info  
… …  
consul:  
    bootstrap = false  
    known\_datacenters = 1  
    leader = false  
    server = true  
raft:  
    applied\_index = 0  
    commit\_index = 0  
    fsm\_pending = 0  
    last\_contact = never  
    last\_log\_index = 0  
    last\_log\_term = 0  
    last\_snapshot\_index = 0  
    last\_snapshot\_term = 0  
    num\_peers = 0  
    state = **Follower**  
    term = 0  
… …

可以看出，n1上的agent当前状态是Follower，bootstrap = false；n2同样也是这个情况。整个Cluster并未完成Bootstrap过程。

我们用consul join命令触发Cluster bootstrap过程，我们在n1上执行如下命令：

$ consul join 10.10.126.101  
Successfully joined cluster by contacting 1 nodes.

我们通过consul join子命令将当前节点加入包含成员10.10.126.101（也就是n2)的集群中去。命令执行结果通过n1和n2的日志可以观察到：

**n1主机:**

2015/07/03 09:29:48 [INFO] agent: (LAN) joining: [10.10.126.101]  
    2015/07/03 09:29:48 [INFO] serf: EventMemberJoin: n2 10.10.126.101  
    2015/07/03 09:29:48 [INFO] agent: (LAN) joined: 1 Err:   
    2015/07/03 09:29:48 [INFO] consul: adding server n2 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 09:29:48 [INFO] consul: Attempting bootstrap with nodes: [10.10.126.101:8300 10.10.105.71:8300]  
    2015/07/03 09:29:49 [INFO] consul: New leader elected: n2  
    2015/07/03 09:29:50 [INFO] agent: Synced service 'consul'

**n2主机:**

2015/07/03 11:40:53 [INFO] serf: EventMemberJoin: n1 10.10.105.71  
    2015/07/03 11:40:53 [INFO] consul: adding server n1 (Addr: 10.10.105.71:8300) (DC: dc1)  
    2015/07/03 11:40:53 [INFO] consul: Attempting bootstrap with nodes: [10.10.126.101:8300 10.10.105.71:8300]  
    2015/07/03 11:40:54 [WARN] raft: Heartbeat timeout reached, starting election  
    2015/07/03 11:40:54 [INFO] raft: Node at 10.10.126.101:8300 [Candidate] entering Candidate state  
    2015/07/03 11:40:54 [INFO] raft: Election won. Tally: 2  
    2015/07/03 11:40:54 [INFO] raft: Node at 10.10.126.101:8300 [Leader] entering Leader state  
    2015/07/03 11:40:54 [INFO] consul: cluster leadership acquired  
    2015/07/03 11:40:54 [INFO] consul: New leader elected: n2  
    2015/07/03 11:40:54 [INFO] raft: pipelining replication to peer 10.10.105.71:8300  
    2015/07/03 11:40:54 [INFO] consul: member 'n2' joined, marking health alive  
    2015/07/03 11:40:54 [INFO] consul: member 'n1' joined, marking health alive  
    2015/07/03 11:40:55 [INFO] agent: Synced service 'consul'

join后，两台主机互相知道了对方，并进行了leader election过程，n2被选举为Leader。

在n2主机上通过consul info确认一下n2 agent的状态：

$consul info  
… …  
consul:  
    bootstrap = false  
    known\_datacenters = 1  
    leader = true  
    server = true  
raft:  
    applied\_index = 10  
    commit\_index = 10  
    fsm\_pending = 0  
    last\_contact = never  
    last\_log\_index = 10  
    last\_log\_term = 1  
    last\_snapshot\_index = 0  
    last\_snapshot\_term = 0  
    num\_peers = 1  
    state = **Leader**  
    term = 1  
… …

$ consul members  
Node  Address             Status  Type    Build  Protocol  DC  
n2    10.10.126.101:8301  alive   server  0.5.2  2         dc1  
n1    10.10.105.71:8301   alive   server  0.5.2  2         dc1

可以看到n2的state已经为Leader了，n1的state依旧是Follower。

到这里，n1和n2就成为了dc1这个数据中心Consul Cluster的两个节点，而且是用来维护集群状态的Server node。n2被选举为Leader，n1是Folllower。

如果作为Leader的n2退出集群，我们来看看集群状态会发生怎样变化。在n2上，我们通过consul leave命令告诉n2上的agent离开集群并退出：

$ consul leave  
Graceful leave complete

n2上Agent的日志：

2015/07/03 14:04:40 [INFO] agent.rpc: Accepted client: 127.0.0.1:35853  
    2015/07/03 14:04:40 [INFO] agent.rpc: Graceful leave triggered  
    2015/07/03 14:04:40 [INFO] consul: server starting leave  
    2015/07/03 14:04:40 [INFO] raft: Removed peer 10.10.105.71:8300, stopping replication (Index: 7)  
    2015/07/03 14:04:40 [INFO] raft: Removed ourself, transitioning to follower  
    2015/07/03 14:04:40 [INFO] raft: Node at 10.10.126.101:8300 [Follower] entering Follower state  
    2015/07/03 14:04:40 [INFO] serf: EventMemberLeave: n2.dc1 10.10.126.101  
    2015/07/03 14:04:40 [INFO] consul: cluster leadership lost  
    2015/07/03 14:04:40 [INFO] raft: aborting pipeline replication to peer 10.10.105.71:8300  
    2015/07/03 14:04:40 [INFO] consul: removing server n2.dc1 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 14:04:41 [INFO] serf: EventMemberLeave: n2 10.10.126.101  
    2015/07/03 14:04:41 [INFO] consul: removing server n2 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 14:04:41 [INFO] agent: requesting shutdown  
    2015/07/03 14:04:41 [INFO] consul: shutting down server  
    2015/07/03 14:04:42 [INFO] agent: shutdown complete

n1上的日志：

2015/07/03 11:53:36 [INFO] serf: EventMemberLeave: n2 10.10.126.101  
2015/07/03 11:53:36 [INFO] consul: removing server n2 (Addr: 10.10.126.101:8300) (DC: dc1)  
2015/07/03 11:55:15 [ERR] agent: failed to sync remote state: No cluster leader

这个时候我们在n1上通过consul info查看，n1的状态依旧是Follower，也就是说在双server节点的集群下，一个server退出，将产生无Leader状态。在三 server节点集群里，Leader退出，其余两个会再协商选出一个新Leader，但一旦再退出一个节点，同样集群就不会再有Leader了。 当然，如果是单节点bootstrap的集群( -bootstrap-expect 1 )，集群只有一个server节点，那这个server节点自然当选Leader。

现在我们在n1上通过consul members查看集群状态：

$ consul members  
Node  Address             Status  Type    Build  Protocol  DC  
n1    10.10.105.71:8301   alive   server  0.5.2  2         dc1  
n2    10.10.126.101:8301  **left**   server  0.5.2  2         dc1

执行结果显示：n2是Left状态。我们重新启动n2，再来看看集群的状态变化。

$ consul agent -server -bootstrap-expect 2 -data-dir /tmp/consul -node=n2 -bind=10.10.126.101 -ui-dir ./dist  -dc=dc1  
… …  
==> Log data will now stream in as it occurs:

    2015/07/03 14:13:46 [INFO] serf: EventMemberJoin: n2 10.10.126.101  
    2015/07/03 14:13:46 [INFO] raft: Node at 10.10.126.101:8300 [Follower] entering Follower state  
    2015/07/03 14:13:46 [INFO] consul: adding server n2 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 14:13:46 [INFO] serf: EventMemberJoin: n2.dc1 10.10.126.101  
    2015/07/03 14:13:46 [INFO] consul: adding server n2.dc1 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 14:13:46 [ERR] agent: failed to sync remote state: No cluster leader  
    2015/07/03 14:13:48 [WARN] raft: EnableSingleNode disabled, and no known peers. Aborting election.  
… …

n2启动后，并未自动加入之前的cluster，而是依旧如第一次启动那样，看不到peers，孤立运行。

我们再来在n1上join一下：consul join 10.10.126.101

n1的日志变为：

2015/07/03 12:04:55 [INFO] consul: adding server n2 (Addr: 10.10.126.101:8300) (DC: dc1)  
2015/07/03 12:04:56 [ERR] agent: failed to sync remote state: No cluster leader

n2的日志变为：

    2015/07/03 14:16:00 [INFO] serf: EventMemberJoin: n1 10.10.105.71  
    2015/07/03 14:16:00 [INFO] consul: adding server n1 (Addr: 10.10.105.71:8300) (DC: dc1)  
    2015/07/03 14:16:00 [INFO] consul: New leader elected: n2  
    2015/07/03 14:16:01 [ERR] agent: failed to sync remote state: No cluster leader

n1和n2无法再选出Leader，通过info命令看，两个节点都变成了Follower，集群仍然处于无Leader状态。

这个问题在consul的github repositroy issues中被多人多次提及，但作者似乎不将此作为bug。产生这个问题的原因是当n2退出时，consul会将/tmp/consul/raft /peers.json的内容由：

["10.10.105.71:8300", "10.10.126.101:8300"]

改为

null

n2重启后，该文件并未改变，依旧为null，n2启动就不会重新自动join到n1的cluster中。

关于这个问题的cluster恢复方法，官方在[Outage Recovery](https://www.consul.io/docs/guides/outage.html)一文中有明确说明。我们来测试一下：

我们打开n1和n2的/tmp/consul/raft/peers.json，将其内容统一修改为：

["10.10.126.101:8300","10.10.105.71:8300"]

然后重启n2，但加上-rejoin命令：

$ consul agent -server -bootstrap-expect 2 -data-dir /tmp/consul -node=n2 -bind=10.10.126.101 -ui-dir ./dist  -dc=dc1 **-rejoin**

…. …

    2015/07/03 14:56:02 [WARN] raft: Election timeout reached, restarting election  
    2015/07/03 14:56:02 [INFO] raft: Node at 10.10.126.101:8300 [Candidate] entering Candidate state  
    2015/07/03 14:56:02 [INFO] raft: Election won. Tally: 2  
    2015/07/03 14:56:02 [INFO] raft: Node at 10.10.126.101:8300 [Leader] entering Leader state  
    2015/07/03 14:56:02 [INFO] consul: cluster leadership acquired  
    2015/07/03 14:56:02 [INFO] consul: New leader elected: n2  
…….

n1上的日志：

2015/07/03 12:44:52 [INFO] serf: EventMemberJoin: n2 10.10.126.101  
    2015/07/03 12:44:52 [INFO] consul: adding server n2 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 12:44:54 [INFO] consul: New leader elected: n2  
    2015/07/03 12:44:55 [WARN] raft: Rejecting vote from 10.10.126.101:8300 since we have a leader: 10.10.126.101:8300  
    2015/07/03 12:44:56 [WARN] raft: Heartbeat timeout reached, starting election  
    2015/07/03 12:44:56 [INFO] raft: Node at 10.10.105.71:8300 [Candidate] entering Candidate state  
    2015/07/03 12:44:56 [ERR] raft: Failed to make RequestVote RPC to 10.10.126.101:8300: EOF  
    2015/07/03 12:44:57 [INFO] raft: Node at 10.10.105.71:8300 [Follower] entering Follower state  
    2015/07/03 12:44:57 [INFO] consul: New leader elected: n2

这回集群的Leader重新选举成功，集群状态恢复。

接下来我们启动n3上的client mode agent：

$ consul agent  -data-dir /tmp/consul -node=n3 -bind=10.10.126.187  -dc=dc1  
==> WARNING: It is highly recommended to set GOMAXPROCS higher than 1  
==> Starting Consul agent…  
==> Starting Consul agent RPC…  
==> Consul agent running!  
         Node name: 'n3'  
        Datacenter: 'dc1'  
            Server: false (bootstrap: false)  
       Client Addr: 127.0.0.1 (HTTP: 8500, HTTPS: -1, DNS: 8600, RPC: 8400)  
      Cluster Addr: 10.10.126.187 (LAN: 8301, WAN: 8302)  
    Gossip encrypt: false, RPC-TLS: false, TLS-Incoming: false  
             Atlas:

==> Log data will now stream in as it occurs:

    2015/07/03 14:55:17 [INFO] serf: EventMemberJoin: n3 10.10.126.187  
    2015/07/03 14:55:17 [ERR] agent: failed to sync remote state: No known Consul servers

在n3上join n1后，n3的日志输出如下：

   2015/07/03 14:59:31 [INFO] agent: (LAN) joining: [10.10.105.71]  
    2015/07/03 14:59:31 [INFO] serf: EventMemberJoin: n2 10.10.126.101  
    2015/07/03 14:59:31 [INFO] serf: EventMemberJoin: n1 10.10.105.71  
    2015/07/03 14:59:31 [INFO] agent: (LAN) joined: 1 Err:   
    2015/07/03 14:59:31 [INFO] consul: adding server n2 (Addr: 10.10.126.101:8300) (DC: dc1)  
    2015/07/03 14:59:31 [INFO] consul: adding server n1 (Addr: 10.10.105.71:8300) (DC: dc1)

n3上consul members可以查看到如下内容：

$ consul members  
Node  Address             Status  Type    Build  Protocol  DC  
n1    10.10.105.71:8301   alive   server  0.5.2  2         dc1  
n3    10.10.126.187:8301  alive   client  0.5.2  2         dc1  
n2    10.10.126.101:8301  alive   server  0.5.2  2         dc1

处于client mode的agent可以自由退出和启动，不会出现server mode下agent的问题。

**二、服务注册与发现**

我们建立Consul Cluster是为了实现服务的注册和发现。Consul支持两种服务注册的方式，一种是通过Consul的服务注册HTTP API，由服务自身在启动后调用API注册自己，另外一种则是通过在配置文件中定义服务的方式进行注册。Consul文档中建议使用后面一种方式来做服务 配置和服务注册。

我们还是用例子来说明一下如何做服务配置。前面我们已经建立了Consul Cluster，Cluster里包含了三个Node：两个Server mode node，一个Client mode Node。我们计划在n2、n3上部署一类服务web3，于是我们需要分别在n2、n3上增加Consul agent的配置文件。

Consul agent在启动时可以通过-config-dir来指定配置文件所在目录，比如以n3为例，我们可以如此启动n3：

consul agent -data-dir /tmp/consul -node=n3 -bind=10.10.126.187 -dc=dc1 -config-dir=./conf

这样在./conf下的所有文件扩展为.json的文件都会被Consul agent作为配置文件读取。

我们以n3为例，我们在n3的consul agent的配置文件目录下创建web3.json文件：

//web3.json  
{  
  "service": {  
    "name": "web3",  
    "tags": ["master"],  
    "address": "127.0.0.1",  
    "port": 10000,  
    "checks": [  
      {  
        "http": "http://localhost:10000/health",  
        "interval": "10s"  
      }  
    ]  
  }  
}

这个配置就是我们在n3节点上为web3这个服务做的服务定义，定义中包含服务的name、address、port等，还包含一个服务检测的配 置，这里 我们每隔10s对服务进行一次健康检查，这要求服务增加对/health的处理逻辑。同理，我们在n2上也建立同样配置文件（n2需重启，并带上 -config-dir命令行选项），**服务注册**就这么简单。

在重启后的n2、n3日志中，我们能发现如下的错误内容：

2015/07/06 13:48:11 [WARN] agent: http request failed 'http://localhost:10000/health' : Get http://localhost:10000/health: dial tcp 127.0.0.1:10000: connect failed"

这就是agent对定义的服务的check日志。为了避免这个错误日志刷屏，我们在n2、n3上各部署一个web3服务实例。以n3上的web3为例，其源码如下：

//web3.go  
package main

import (  
    "fmt"  
    "net/http"  
)

func handler(w http.ResponseWriter, r \*http.Request) {  
    fmt.Println("hello Web3! This is n3")  
    fmt.Fprintf(w, "Hello Web3! This is n3")  
}

func healthHandler(w http.ResponseWriter, r \*http.Request) {  
    fmt.Println("health check!")  
}

func main() {  
    http.HandleFunc("/", handler)  
    http.HandleFunc("/health", healthHandler)  
    http.ListenAndServe(":10000", nil)  
}

一旦n2、n3上的web3服务实例启动，我们就可以尝试发现这些服务了。

Consul提供了两种发现服务的方式，一种是通过HTTP API查看存在哪些服务；另外一种是通过consul agent内置的DNS服务来做。两者的差别在于后者可以根据服务check的实时状态动态调整available服务节点列表。我们这里也着重说明适用 DNS方式进行服务发现的具体步骤。

在配置和部署完web3服务后，我们就可以通过DNS命令来查询服务的具体信息了。consul为服务编排的内置域名为 “NAME.service.consul"，这样我们的web3的域名为:web3.service.consul。我们在n1通过dig工具来查看一 下，注意是在n1上，n1上并未定义和部署web3服务，但集群中服务的信息已经被同步到n1上了，信息是一致的：

$ dig [@127.0.0.1](http://my.oschina.net/u/567043) -p 8600 web3.service.consul SRV

; <<>> DiG 9.9.5-3-Ubuntu <<>> [@127.0.0.1](http://my.oschina.net/u/567043) -p 8600 web3.service.consul SRV  
; (1 server found)  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6713  
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 2  
;; WARNING: recursion requested but not available

;; QUESTION SECTION:  
;web3.service.consul.        IN    SRV

;; ANSWER SECTION:  
web3.service.consul.    0    IN    SRV    1 1 10000 n2.node.dc1.consul.  
web3.service.consul.    0    IN    SRV    1 1 10000 n3.node.dc1.consul.

;; ADDITIONAL SECTION:  
n2.node.dc1.consul.    0    IN    A    127.0.0.1  
n3.node.dc1.consul.    0    IN    A    127.0.0.1

;; Query time: 2 msec  
;; SERVER: 127.0.0.1#8600(127.0.0.1)  
;; WHEN: Mon Jul 06 12:12:53 CST 2015  
;; MSG SIZE  rcvd: 219

可以看到在ANSWER SECTION中，我们得到了两个结果：n2和n3上各有一个web3的服务。在dig命令中我们用了SRV标志，那是因为我们需要的服务信息不仅有ip地址，还需要有端口号。

现在我们停掉n2上的web3服务，10s后，我们再来查一下：

$ dig [@127.0.0.1](http://my.oschina.net/u/567043) -p 8600 web3.service.consul SRV

; <<>> DiG 9.9.5-3-Ubuntu <<>> [@127.0.0.1](http://my.oschina.net/u/567043) -p 8600 web3.service.consul SRV  
; (1 server found)  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25136  
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1  
;; WARNING: recursion requested but not available

;; QUESTION SECTION:  
;web3.service.consul.        IN    SRV

;; ANSWER SECTION:  
web3.service.consul.    0    IN    SRV    1 1 10000 n3.node.dc1.consul.

;; ADDITIONAL SECTION:  
n3.node.dc1.consul.    0    IN    A    127.0.0.1

;; Query time: 3 msec  
;; SERVER: 127.0.0.1#8600(127.0.0.1)  
;; WHEN: Mon Jul 06 12:16:39 CST 2015  
;; MSG SIZE  rcvd: 128

结果显示，只有n3上这一个web3服务可用了。通过下面Consul Agent日志：

dns: node 'n2' failing health check 'service web3' check', dropping from service 'web3'

我们可以看到consul agent将health check失败的web3从结果列表中剔除了，这样web3服务的客户端在服务发现过程中就只能获取到当前可用的web3服务节点了，这个好处是在实际应 用中大大降低了客户端实现”服务发现“时的难度。另外consul agent DNS在返回查询结果时也支持DNS Server常见的策略，至少是支持轮询。你可以多次执行dig命令，可以看到n2和n3的排列顺序是不同的。还有一点值得注意的是：由于考虑DNS cache对consul agent查询结果的影响，默认情况下所有由consul agent返回的结果TTL值均设为0，也就是说不支持dns结果缓存。

接下来，我们使用golang实现一个demo级别的服务发现的客户端，这里会用到第三方dns client库"github.com/miekg/dns"。

// servicediscovery.go  
package main

import (  
    "fmt"  
    "log"

    "github.com/miekg/dns"  
)

const (  
        srvName = "web3.service.consul"  
        agentAddr = "127.0.0.1:8600"  
)

func main() {  
    c := new(dns.Client)

    m := new(dns.Msg)  
    m.SetQuestion(dns.Fqdn(srvName), dns.TypeSRV)  
    m.RecursionDesired = true

    r, \_, err := c.Exchange(m, agentAddr)  
    if r == nil {  
        log.Fatalf("dns query error: %s\n", err.Error())  
    }

    if r.Rcode != dns.RcodeSuccess {  
        log.Fatalf("dns query error: %v\n", r.Rcode)  
    }  
     
    for \_, a := range r.Answer {  
        b, ok := a.(\*dns.SRV)  
        if ok {  
            m.SetQuestion(dns.Fqdn(b.Target), dns.TypeA)  
            r1, \_, err := c.Exchange(m, agentAddr)  
            if r1 == nil {  
                log.Fatalf("dns query error: %v, %v\n", r1.Rcode, err)  
            }  
            for \_, a1 := range r1.Answer {  
                c, ok := a1.(\*dns.A)  
                if ok {  
                   fmt.Printf("%s – %s:%d\n", b.Target, c.A, b.Port)  
                }  
            }  
        }  
    }  
}

我们执行该程序：  
$ go run servicediscovery.go  
n2.node.dc1.consul. – 10.10.126.101:10000  
n3.node.dc1.consul. – 10.10.126.187:10000

注意各个node上的服务check是由其node上的agent上进行的，一旦那个node上的agent出现问题，则位于那个node上的所有 service也将会被置为unavailable状态。比如我们停掉n3上的agent，那么我们在进行web3服务节点查询时，就只能获取到n2这一 个节点上有可用的web3服务了。

在真实的程序中，我们可以像上面demo中那样，每Request都做一次DNS查询，不过这样的代价也很高。稍复杂些，我们可以结合dns结果本地缓存+定期查询+每遇到Failed查询的方式来综合考量服务的发现方法或利用Consul提供的watch命令等。

以上仅仅是Consul的一个入门。真实场景中，理想的方案需要考虑的事情还有很多。Consul自身目前演进到0.5.2版本，还有不完善之处， 但它已 经被很多公司用于production环境。Consul不是孤立的，要充分发挥出Consul的优势，在真实方案中，我们还要考虑与 [Docker](https://www.docker.com/)，HAProxy，Mesos等工具的结合。

© 2015, [bigwhite](http://tonybai.com/). 版权所有.

本文来自：[Tony Bai](http://studygolang.com/wr?u=http://tonybai.com)

感谢作者：bigwhite

查看原文：[使用consul实现分布式服务注册和发现](http://studygolang.com/wr?u=http%3a%2f%2ftonybai.com%2f2015%2f07%2f06%2fimplement-distributed-services-registery-and-discovery-by-consul%2f)

# hashicorp/consul: Service Discovery and Configuration Made Easy

https://github.com/hashicorp/consul

* Website: [https://www.consul.io](https://www.consul.io/)
* Chat: [Gitter](https://gitter.im/hashicorp-consul/Lobby)
* Mailing list: [Google Groups](https://groups.google.com/group/consul-tool/)

Consul is a tool for service discovery and configuration. Consul is distributed, highly available, and extremely scalable.

Consul provides several key features:

* **Service Discovery** - Consul makes it simple for services to register themselves and to discover other services via a DNS or HTTP interface. External services such as SaaS providers can be registered as well.
* **Health Checking** - Health Checking enables Consul to quickly alert operators about any issues in a cluster. The integration with service discovery prevents routing traffic to unhealthy hosts and enables service level circuit breakers.
* **Key/Value Storage** - A flexible key/value store enables storing dynamic configuration, feature flagging, coordination, leader election and more. The simple HTTP API makes it easy to use anywhere.
* **Multi-Datacenter** - Consul is built to be datacenter aware, and can support any number of regions without complex configuration.

Consul runs on Linux, Mac OS X, FreeBSD, Solaris, and Windows.

Quick Start

An extensive quick start is viewable on the Consul website:

<https://www.consul.io/intro/getting-started/install.html>

Documentation

Full, comprehensive documentation is viewable on the Consul website:

<https://www.consul.io/docs>

Developing Consul

If you wish to work on Consul itself, you'll first need [Go](https://golang.org/) installed (version 1.8+ is required). Make sure you have Go properly installed, including setting up your [GOPATH](https://golang.org/doc/code.html#GOPATH).

Next, clone this repository into $GOPATH/src/github.com/hashicorp/consul and then just type make. In a few moments, you'll have a working consul executable:

$ make

...

$ bin/consul

...

Note:*make*will also place a copy of the binary in the first part of your*$GOPATH*.

You can run tests by typing make test.

If you make any changes to the code, run make format in order to automatically format the code according to Go standards.

Building Consul on Windows

Make sure Go 1.8+ is installed on your system and that the Go command is in your %PATH%.

For building Consul on Windows, you also need to have MinGW installed. [TDM-GCC](http://tdm-gcc.tdragon.net/) is a simple bundle installer which has all the required tools for building Consul with MinGW.

Install TDM-GCC and make sure it has been added to your %PATH%.

If all goes well, you should be able to build Consul by running make.bat from a command prompt.

See also [golang/winstrap](https://github.com/golang/winstrap) and [golang/wiki/WindowsBuild](https://github.com/golang/go/wiki/WindowsBuild) for more information of how to set up a general Go build environment on Windows with MinGW.

Vendoring

Consul currently uses [govendor](https://github.com/kardianos/govendor) for vendoring.