# consul注册发现

## 1 consul原理与实战

### 1.1 consul简介?

> consul是google开源的一个使用go语言开发的服务发现、配置管理中心服务。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等）。服务部署简单，只有一个可运行的二进制的包。每个节点都需要运行agent，他有两种运行模式server和client。每个数据中心官方建议需要3或5个server节点以保证数据安全，同时保证server-leader的选举能够正确的进行。

类似的工具还有：ZooKeeper，etcd等等。

### 1.2 为什么使用服务发现

防止硬编码、容灾、水平扩缩容、提高运维效率等等，只要你想使用服务发现总能找到合适的理由。

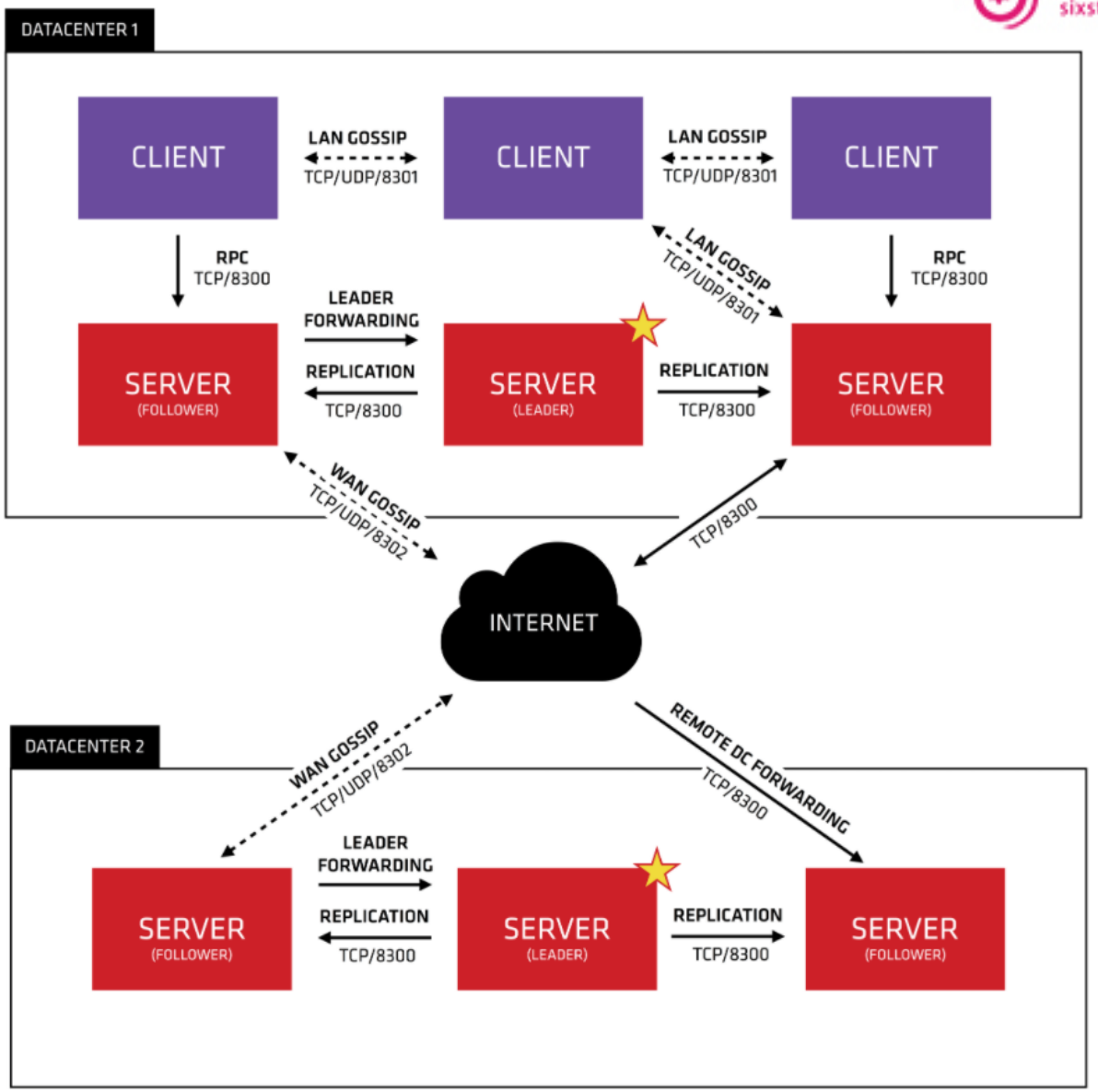
一般的说法是因为使用微服务架构。传统的单体架构不够灵活不能很好的适应变化，从而向微服务架构进行转换。

而伴随着大量服务的出现，管理运维十分不便，于是开始搞一些自动化的策略，服务发现应运而生。所以如果需要使用服务发现，你应该有一些对服务治理的痛点。

但是引入服务发现就可能引入一些技术栈，增加系统总体的复杂度，如果你只有很少的几个服务，比如 10 个以下，并且业务不怎么变化，吞吐量预计也很稳定，可能就没有必要使用服务发现。

### 1.3 consul工作原理

掌握原理前，需了解下consul的组成结构，即多数据中心




集群中的特性功能：

- 节点角色：leader(领导者)、Follower(跟随者)
- consul节点身份：server、client
- Agent：即节点内包含的一个后台处理程序
- 健康检查：由client的Agent来进行健康检查的功能

首先 Consul 支持多数据中心，在上图中有两个 DataCenter，他们通过 Internet 互联，同时请注意为了提高通信效率，只有 Server 节点才加入跨数据中心的通信。

在单个数据中心中，Consul 分为 Client 和 Server 两种节点（所有的节点也被称为 Agent），Server 节点保存数据，Client 负责健康检查及转发数据请求到 Server。

Server 节点有一个 Leader 和多个 Follower，Leader 节点会将数据同步到 Follower，Server 的数量推荐是 3 个或者 5 个，在 Leader 挂掉的时候会启动选举机制产生一个新的 Leader。

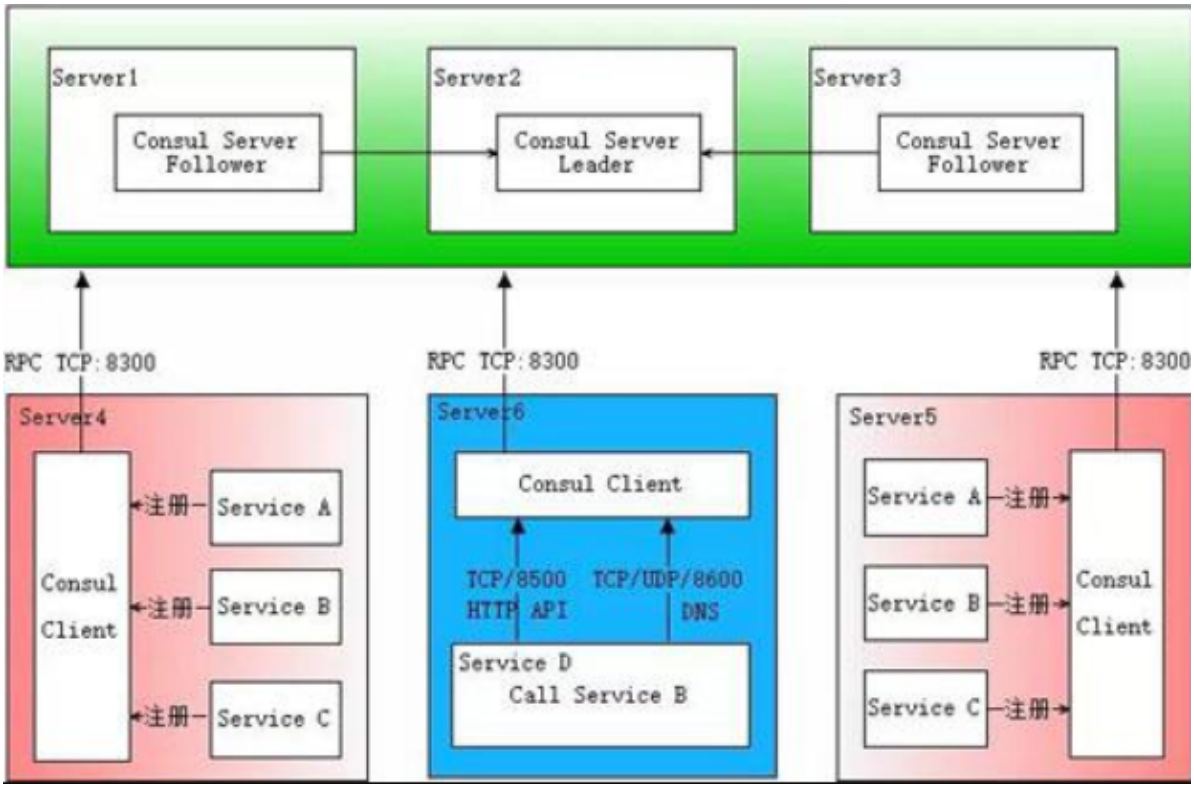
集群内的 Consul 节点通过 gossip 协议（流言协议）维护成员关系，也就是说某个节点了解集群内现在还有哪些节点，这些节点是 Client 还是 Server。

单个数据中心的流言协议同时使用 TCP 和 UDP 通信，并且都使用 8301 端口。跨数据中心的流言协议也同时使用 TCP 和 UDP 通信，端口使用 8302。

集群内数据的读写请求既可以直接发到 Server，也可以通过 Client 使用 RPC 转发到 Server，请求最终会到达 Leader 节点。

在允许数据轻微陈旧的情况下，读请求也可以在普通的 Server 节点完成，集群内数据的读写和复制都是通过 TCP 的 8300 端口完成。

## 2 consul服务发现原理




首先需要有一个正常的 Consul 集群，有 Server，有 Leader。这里在服务器 Server1、Server2、Server3 上分别部署了 Consul Server。

假设他们选举了 Server2 上的 Consul Server 节点为 Leader。这些服务器上最好只部署 Consul 程序，以尽量维护 Consul Server 的稳定。

然后在服务器 Server4 和 Server5 上通过 Consul Client 分别注册 Service A、B、C，这里每个 Service 分别部署在了两个服务器上，这样可以避免 Service 的单点问题。

服务注册到 Consul 可以通过 HTTP API（8500 端口）的方式，也可以通过 Consul 配置文件的方式。

Consul Client 可以认为是无状态的，它将注册信息通过 RPC 转发到 Consul Server，服务信息保存在 Server 的各个节点中，并且通过 Raft 实现了强一致性。

最后在服务器 Server6 中 Program D 需要访问 Service B，这时候 Program D 首先访问本机 Consul Client 提供的 HTTP API，本机 Client 会将请求转发到 Consul Server。

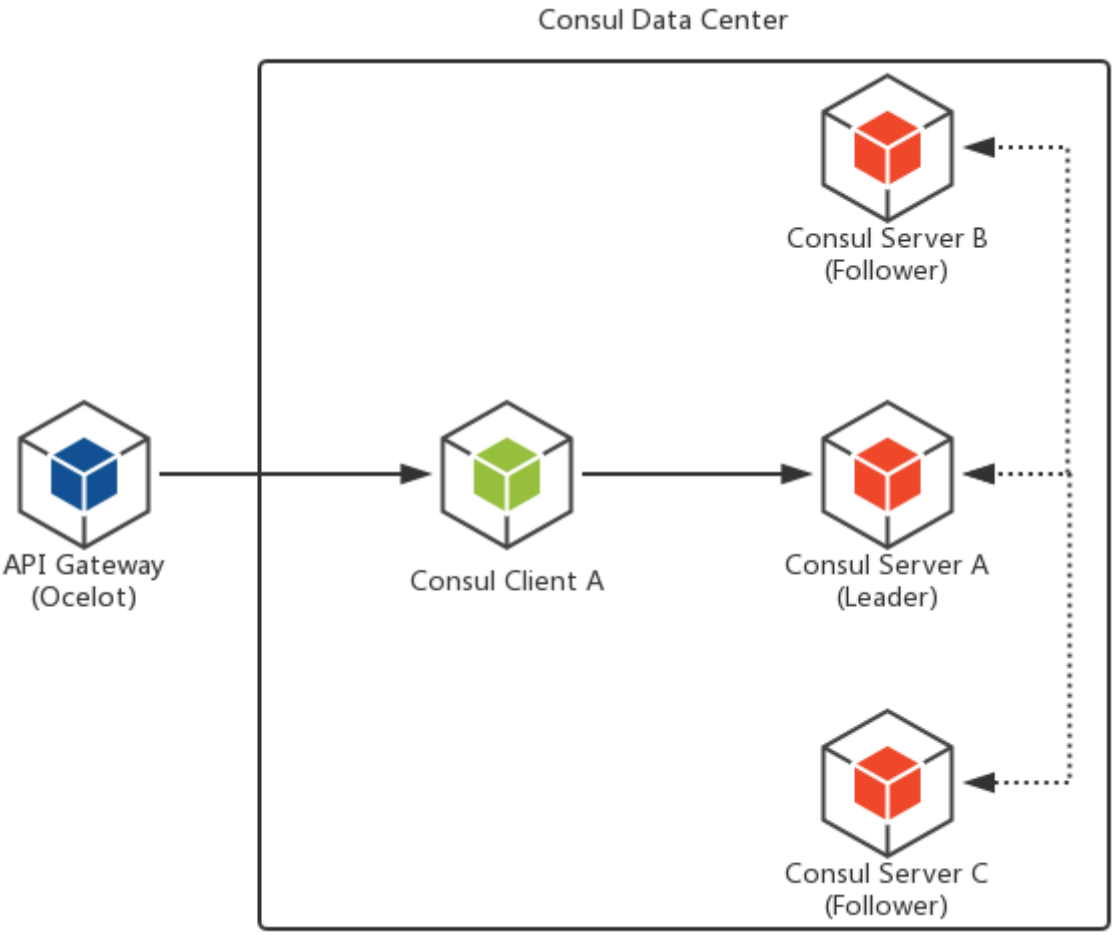
Consul Server 查询到 Service B 当前的信息返回，最终 Program D 拿到了 Service B 的所有部署的 IP 和端口，然后就可以选择 Service B 的其中一个部署并向其发起请求了。

如果服务发现采用的是 DNS 方式，则 Program D 中直接使用 Service B 的服务发现域名，域名解析请求首先到达本机 DNS 代理，然后转发到本机 Consul Client，本机 Client 会将请求转发到 Consul Server。

Consul Server 查询到 Service B 当前的信息返回，最终 Program D 拿到了 Service B 的某个部署的 IP 和端口。

# 3 consul集群






为了更快的熟悉 Consul 的原理及其使用方式，最好还是自己实际测试下。

启动 Consul 集群

执行命令拉取最新版本的 Consul 镜像：

```
docker pull consul:1.4.4
```

然后就可以启动集群了，这里启动 4 个 Consul Agent，3 个 Server（会选举出一个 Leader），1 个 Client。

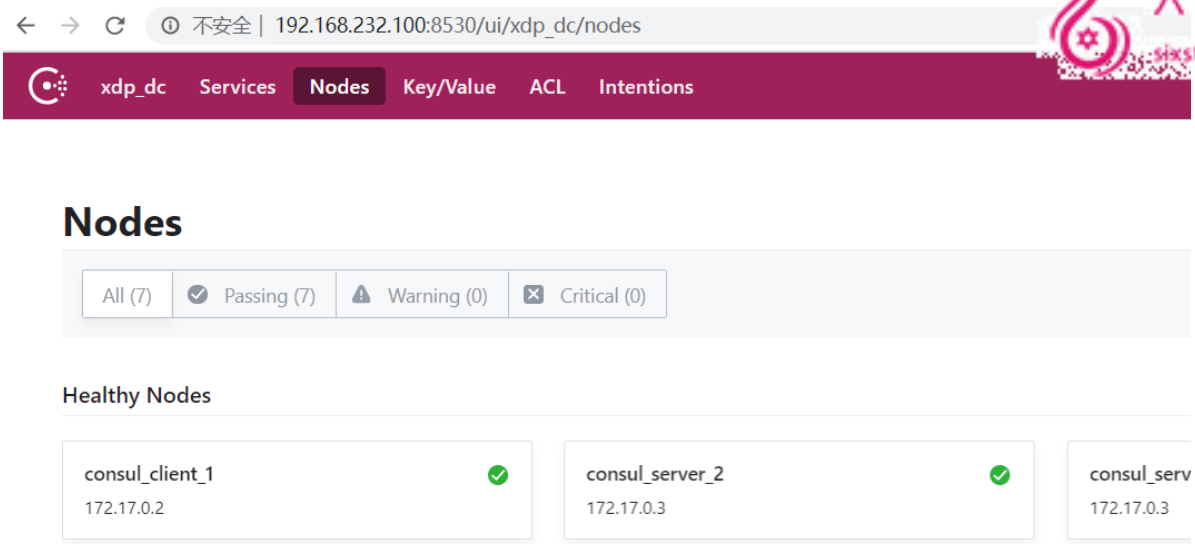
```
#创建自定义网段
[root@localhost ~]# docker network create --subnet=172.200.7.0/10 mynetwork
5685db7f3bdbf8701bed03a42f46928562be06401abcb2b9eb4ce7752666bd35


#启动第1个Server节点，集群要求要有3个Server，将容器8500端口映射到主机8900端口，同时开启管理界面
```

```
[root@localhost ~]# docker run -d -p 8510:8500 -v
/XiLife/consul/data/server1:/consul/data -v
/XiLife/consul/conf/server1:/consul/config -e CONSUL_BIND_INTERFACE='eth0' --
network=my-redis-network --ip 172.200.7.6 --privileged=true --
name=consul1.4.4_server_public_2 consul:1.4.4 agent -server -bootstrap-expect=3 -
ui -node=consul1.4.4_server_public_2 -client='0.0.0.0' -data-dir /consul/data -
config-dir  /consul/config -datacenter=xdp_dc

[root@localhost ~]# docker inspect consul1.4.4_server_public_2 | grep
"IPAddress"
            "SecondaryIPAddresses": null,
            "IPAddress": "172.200.7.6",
                    "IPAddress": "172.200.7.6",


#启动第2个Server节点，并加入集群
[root@localhost ~]# docker run -d -p 8520:8500 -v
/XiLife/consul/data/server2:/consul/data -v
/XiLife/consul/conf/server2:/consul/config -e CONSUL_BIND_INTERFACE='eth0' --
network=my-redis-network --ip 172.200.7.7 --privileged=true --
name=consul1.4.4_server_public_3 consul:1.4.4 agent -server -ui -
node=consul1.4.4_server_public_3 -client='0.0.0.0' -datacenter=xdp_dc -data-dir
/consul/data -config-dir /consul/config -join=172.200.7.6

#启动第3个Server节点，并加入集群
[root@localhost ~]# docker run -d -p 8530:8500 -v
/XiLife/consul/data/server3:/consul/data -v
/XiLife/consul/conf/server3:/consul/config -e CONSUL_BIND_INTERFACE='eth0' --
network=my-redis-network --ip 172.200.7.8 --privileged=true --
name=consul1.4.4_server_public_4 consul:1.4.4 agent -server -ui -
node=consul1.4.4_server_public_4 -client='0.0.0.0' -datacenter=xdp_dc -data-dir
/consul/data -config-dir /consul/config -join=172.200.7.6
```

验证1：

```
[root@localhost ~]# docker exec consul1.4.4_server_public_2  consul operator raft
list-peers
Node             ID                                    Address          State
Voter  RaftProtocol
consul1.4.4_server_public_3  296b195d-67ab-335d-4941-05c9b2bb9d54
172.200.7.7:8300  leader    true    3
consul1.4.4_server_public_2  b16a836e-77db-4213-d3f3-3c095f6b9afb
172.200.7.6:8300  follower  true    3
consul1.4.4_server_public_4  60fc6a62-62a1-328b-d186-51a33e4690cf
172.200.7.8:8300  follower  true    3
```

验证2：http://192.168.30.128:8530/




第 1 个启动容器的 IP 一般是 172.200.7.6，后边启动的几个容器 IP 会排着来：172.200.7.7、172.200.7.8、172.200.7.9。

这些 Consul 节点在 Docker 的容器内是互通的，他们通过桥接的模式通信。但是如果主机要访问容器内的网络，需要做端口映射。

在启动容器时，将 Consul 的 8500 端口映射到了主机的 8300 端口，这样就可以方便的通过主机的浏览器查看集群信息。

Consul Client实例创建

（1）准备services.json配置文件，向Consul注册两个同样的Product API服务

```
{
    "services": [
      {
        "id": "test1",
        "name": "test1name",
        "tags": [ "xdp-/core.product" ],
        "address": "192.168.232.100",
        "port": 18306,
        "checks": [
          {
            "name": "core.product.check",
            "http": "http://192.168.232.100:18306",
            "interval": "10s", //心跳健康检测频率
            "timeout": "5s" //检测超时时间
          }
        ]
      },
      {
        "id": "core.product-/192.168.16.170:8001",
        "name": "core.product",
        "tags": [ "xdp-/core.product" ],
        "address": "192.168.232.100",
        "port": 18306,
        "checks": [
          {
            "name": "core.product.check",
```

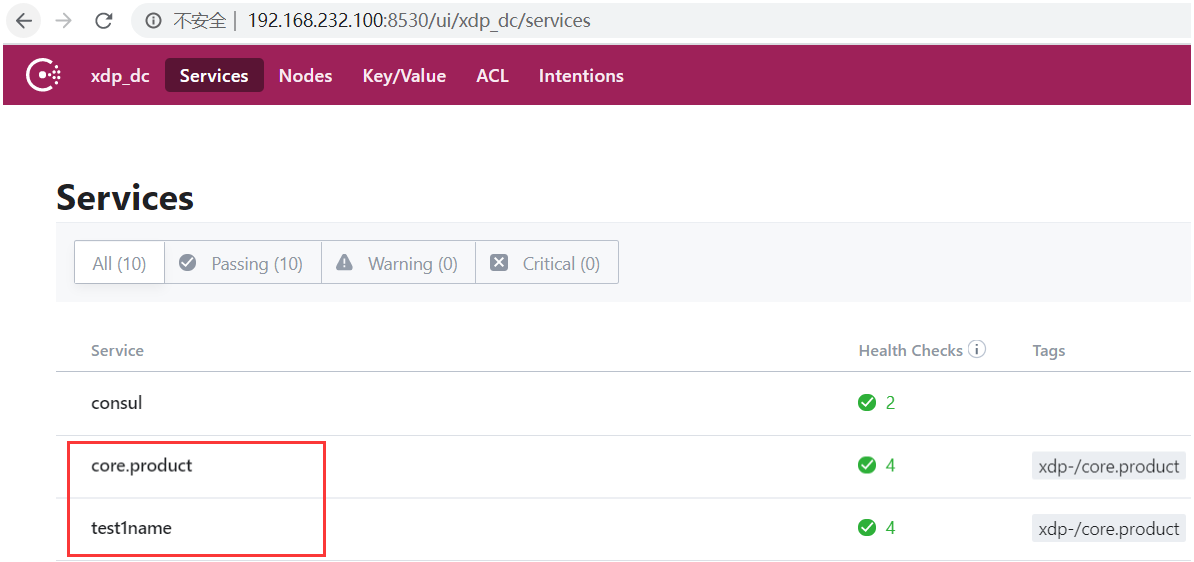
```
            "http": "http://192.168.232.100:18306",
            "interval": "10s",  //心跳健康检测频率
            "timeout": "5s" //检测超时时间
          }
        ]
      }
    ]
  }
```

（2）Consul Client实例

> docker run -d -p 8550:8500 -v /XiLife/consul/conf/client1:/consul/config -e CONSUL_BIND_INTERFACE='eth0'  --network=my-redis-network --ip 172.200.7.9 --name=consul1.4.4_client_public_5  --privileged=true consul:1.4.4 agent -node=consul1.4.4_client_public_5  -join=172.200.7.6 -client='0.0.0.0' -datacenter=xdp_dc -config-dir /consul/config

将services.json文件放到主机的 /XiLife/consul/conf/下的client1 server1 server2 server3目录中

（3）验证




服务注册成功以后，调用方获取相应服务地址的过程就是服务发现。Consul 提供了多种方式。

HTTP API 方式

容器内：

```
curl http://127.0.0.1:8510/v1/health/service/test1name?passing=true
```

容器外：

http://192.168.30.128:8530/v1/health/service/test1name?passing=false


[{"Node":{"ID":"b7b95d8b-50f2-17e3-2635-6151172855ec","Node":"59f09be6c698","Address":"172.17.0.5","Datacenter":"dc1","TaggedAd
{"consul-network-segment":""},"CreateIndex":164,"ModifyIndex":165},"Service":{"ID":"hello1","Service":"hello","Tags":["primary"
{"Address":"172.17.0.5","Port":5000}},"Meta":null,"Port":5000,"Weights":{"Passing":1,"Warning":1},"EnableTagOverride":false,"Pr
[{"Node":"59f09be6c698","CheckID":"serfHealth","Name":"Serf Health Status","Status":"passing","Notes":"","Output":"Agent alive
{},"CreateIndex":164,"ModifyIndex":164},{"Node":"59f09be6c698","CheckID":"service:hello1","Name":"Service 'hello' check","Statu
","ServiceID":"hello1","ServiceName":"hello","ServiceTags":["primary"],"Type":"http","Definition":{},"CreateIndex":184,"ModifyI

返回的信息包括注册的 Consul 节点信息、服务信息及服务的健康检查信息。

passing=true //过滤不健康节点

passing=false //不过滤

如果服务有多个部署，会返回服务的多条信息，调用方需要决定使用哪个部署，常见的可以随机或者轮询。

为了提高服务吞吐量，以及减轻 Consul 的压力，还可以缓存获取到的服务节点信息，不过要做好容错的方案，因为缓存服务部署可能会变得不可用。具体是否缓存需要结合自己的访问量及容错规则来确定。

上边的参数 passing 默认为 false，也就是说不健康的节点也会返回，结合获取节点全部服务的方法，这里可以做到获取全部服务的实时健康状态，并对不健康的服务进行报警处理。

## 3.1节点和服务注销

节点和服务的注销可以使用 HTTP API:

注销任意节点和服务：/catalog/deregister

注销当前节点的服务：/agent/service/deregister/:service_id

```
curl -X PUT
http://192.168.30.128:8530/v1/agent/service/deregister/core.product-/192.168.16.
170:8001
curl -X PUT
http://192.168.30.128:8510/v1/agent/service/deregister/wi_user_service1
```

注意：如果注销的服务还在运行，则会再次同步到 catalog 中，因此应该只在 Agent 不可用时才使用 catalog 的注销 API。

节点在宕机时状态会变为 failed，默认情况下 72 小时后会被从集群移除。

如果某个节点不继续使用了，也可以在本机使用 consul leave 命令，或者在其他节点使用 consul force-leave 节点 id，则节点上的服务和健康检查全部注销。

## 3.2 Consul 集群之负载均衡

本例可以将主机端口8500均匀地分散到8510，8520，8530.

可以通过Nginx建了一个负载均衡入口，即通过8500访问8510、8520即8530。

参考：

```
server {
listen 8500;
location / {
proxy_pass http://xdpconsul;
proxy_redirect default;
}
}
```

```
upstream xdpconsul {
server 127.0.0.1:8510 weight=2;
server 127.0.0.1:8520 weight=1;
server 127.0.0.1:8530 weight=1;
}
```

## 4. consul微服务与微服务

consul微服务也是针对功能而言的，一般分布式场景下，像用户服务、商品服务、订单服务、购物车客服服务等都是集群的方式部署，随便一个集群下面在分各地点、机房、机器、机器。那整个服务治理来说，压力肯定是非常大的。所以咱们可以根据不同的服务来单独的抽取consul进行专治。即用户服务就采取用户consul集群来管理用户服务配置、订单服务就采用订单相关的内容。

既然咱们都把consul进行服务化了，那么业务上的微服务就不言而喻了，就是服务拆分独立为多个子系统来进行构建。