|  |
| --- |
| KCloudy |
| **SpringBoot In Linux** |
| [文档副标题] |

|  |
| --- |
| 田 长军  [日期] |

目录

[**1.** **部署前准备** 0](#_Toc21768647)

[**1.1** **安装Java** 0](#_Toc21768648)

[**1.2** **安装Maven** 0](#_Toc21768649)

[**1.2.1** **下载Maven安装包** 0](#_Toc21768650)

[**1.2.2** **解压Maven安装包** 0](#_Toc21768651)

[**1.2.3** **配置Maven路径** 0](#_Toc21768652)

[**1.2.4** **验证Maven是否配置成功** 0](#_Toc21768653)

[**1.2.5** **使用mvn -v时，出现错误：JAVA\_HOME should point to a JDK not a JRE** 0](#_Toc21768654)

[**1.2.6** **配置国内阿里云Maven镜像** 0](#_Toc21768655)

[**1.3** **安装Git** 0](#_Toc21768656)

[**1.4** **安装Docker** 0](#_Toc21768657)

[**1.4.1** **使用yum源安装** 0](#_Toc21768658)

[**1.4.2** **安装Docker Compose** 0](#_Toc21768659)

[**1.4.3** **安装Docker Machine** 0](#_Toc21768660)

[**1.4.4** **使用国内Docker镜像源** 0](#_Toc21768661)

[**1.4.5** **启动及测试** 0](#_Toc21768662)

[**2.** **Spring Consul** 0](#_Toc21768663)

[**2.1** **安装Consul** 0](#_Toc21768664)

[**2.1.1** **安装三个Consul服务** 0](#_Toc21768665)

[**2.1.2** **使用Docker安装Consul** 0](#_Toc21768666)

[**2.1.3** **其他** 0](#_Toc21768667)

[**2.2** **其他** 0](#_Toc21768668)

[**3.** **其他** 0](#_Toc21768669)

**SpringBoot In Linux**

1. **部署前准备**
   1. **安装Java**

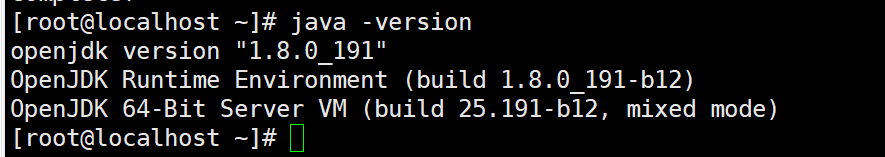
sudo yum -y install java-1.8.0-openjdk #jre

sudo yum -y install java-1.8.0-openjdk-devel #jdk

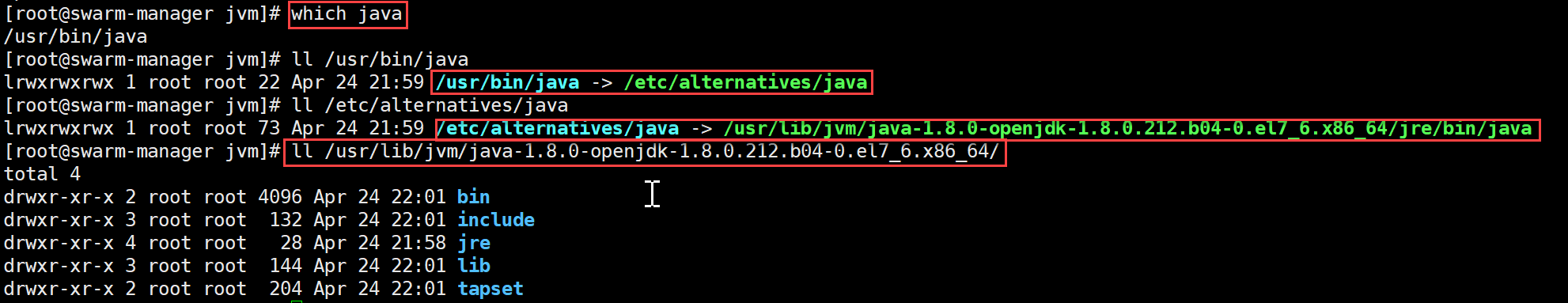
但对于centos有个问题，不可以安装成gcj(例如：gij(GNU libgcj) version 4.4.6 20110731(Red Hat 4.4.6-3)),导致[Jekins不工作](https://issues.jenkins-ci.org/browse/JENKINS-743),如果查看java版本类似上例，需要卸载，安装其他版本

java -version #查看Java版本

yum remove java #卸载异常版本



which java #查看java执行路径，备注：java实际的路径为-/usr/lib/jvm



设置java路径：vi /etc/profile 在文件最后添加下列配置

export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.212.b04-0.el7\_6.x86\_64

export JRE\_HOME=$JAVA\_HOME/jre

export CLASSPATH=$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH

export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH

使配置生效： source /etc/profile

* 1. **安装Maven**
     1. **下载Maven安装包**

wget <https://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.2/binaries/apache-maven-3.6.2-bin.tar.gz>

* + 1. **解压Maven安装包**

tar -zxvf apache-maven-3.6.2-bin.tar.gz

mv apache-maven-3.6.2 /usr/local/maven3

* + 1. **配置Maven路径**

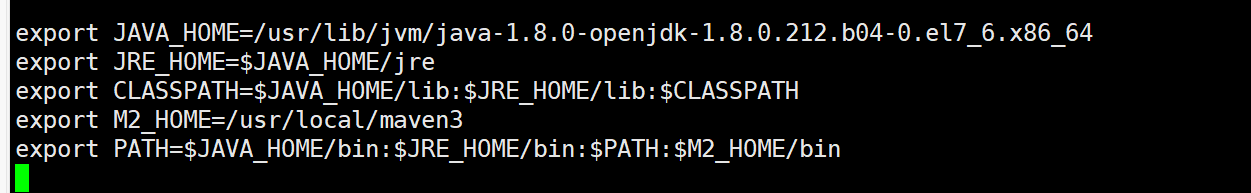
vi /etc/profile #最后添加以下内容

export M2\_HOME=/usr/local/maven3

export PATH=$PATH:$JAVA\_HOME/bin:$M2\_HOME/bin #保存退出后运行下面的命令使配置生效，或者重启服务器生效

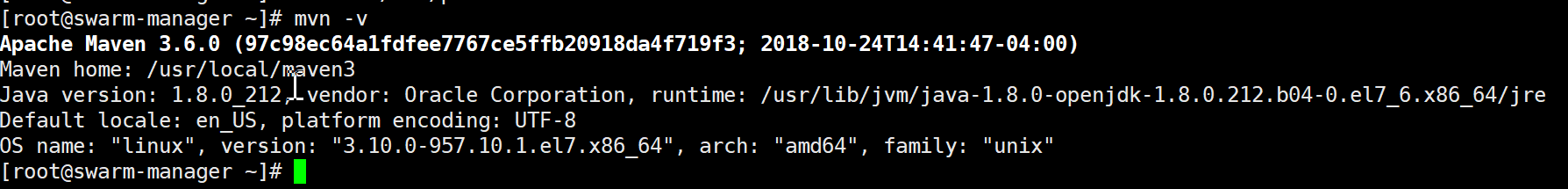
source /etc/profile

配置结果如下图：

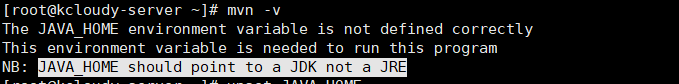


* + 1. **验证Maven是否配置成功**

mvn -v



* + 1. **使用mvn -v时，出现错误：JAVA\_HOME should point to a JDK not a JRE**



unset JAVA\_HOME #重置下JAVA\_HOME

* + 1. **配置国内阿里云Maven镜像**

vi /usr/local/maven3/conf/settings.xml

配置mirrors的子节点，添加如下mirror

<mirror>

<id>nexus-aliyun</id>

<mirrorOf>central</mirrorOf>

<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>



* 1. **安装Git**

sudo yum -y install git

git --version

* 1. **安装Docker**
     1. **使用yum源安装**

sudo wget -P /etc/yum.repos.d/ <http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

sudo yum install -y docker-ce-18.09.3-3.el7 docker-ce-cli-18.09.3 containerd.io-18.09.3

docker --version

* + 1. **安装Docker Compose**

curl -L https://github.com/docker/compose/releases/download/1.23.2/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

docker-compose --version

* + 1. **安装Docker Machine**

sudo wget -P /etc/yum.repos.d/ http://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo

sudo yum install -y VirtualBox-5.1 #安装VirtualBox

curl -L https://github.com/docker/machine/releases/download/v0.16.1/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine &&

chmod +x /tmp/docker-machine &&

sudo cp /tmp/docker-machine /usr/local/bin/docker-machine

docker-machine -version

* + 1. **使用国内Docker镜像源**

vi /etc/docker/daemon.json #修改为以下内容

{

"exec-opts": ["native.cgroupdriver=systemd"],

"registry-mirrors" : [

"http://ovfftd6p.mirror.aliyuncs.com",

"http://registry.docker-cn.com",

"http://docker.mirrors.ustc.edu.cn",

"http://hub-mirror.c.163.com"

],

"insecure-registries" : [

"registry.docker-cn.com",

"docker.mirrors.ustc.edu.cn"

],

"debug" : true,

"experimental" : true

}

* + 1. **启动及测试**
       1. **启动Docker**

service docker start

chkconfig docker on

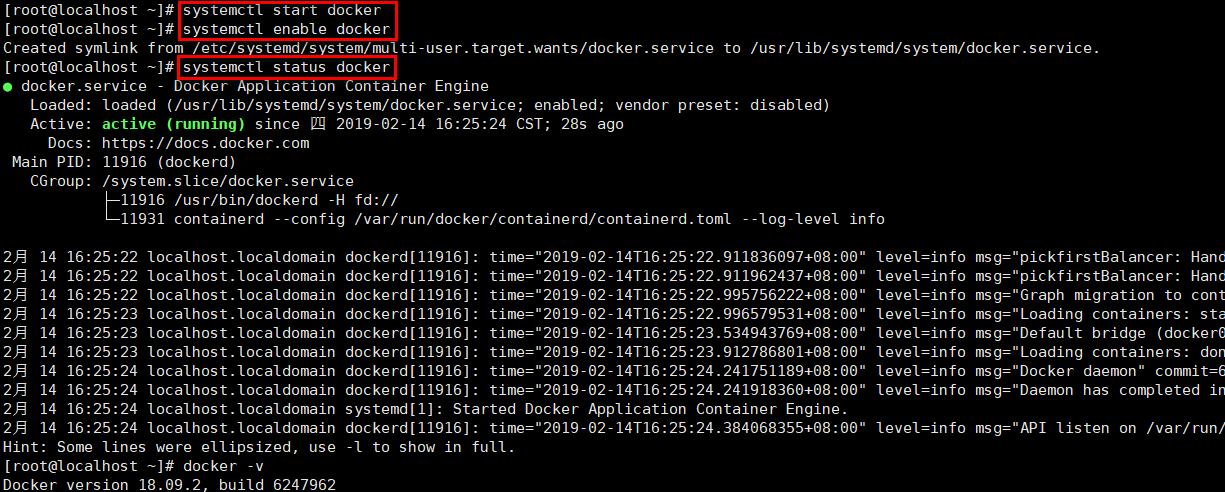
#LCTT 译注：此处采用了旧式的 sysv 语法，如采用CentOS 7中支持的新式 systemd 语法，如下：

systemctl start docker.service #开机自启 Docker CE

systemctl enable docker.service

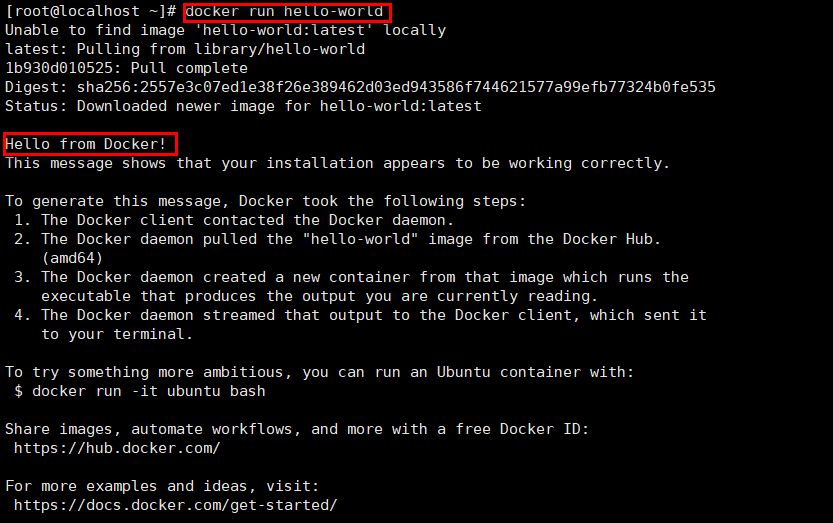
systemctl status docker #查看Docker状态

docker -v #查看docker版本



* + - 1. **测试Docker是否安装正确**

docker run hello-world



1. **Spring Consul**
   1. **安装Consul**
      1. **安装三个Consul服务**

192.168.56.12:8300作为Consul node1(leader)

192.168.56.12:8400作为Consul node2

192.168.56.12:8500作为Consul node3

* + 1. **使用Docker安装Consul**
* 搜索镜像

docker search consul

* 下载镜像

docker pull progrium/consul

* 安装镜像

#consul 服务端 node1

docker run -d -h node1 --name consul\_node1 --restart=always \

-v /data/consul:/consul/data \

-p 8300:8300 -p 8301:8301 -p 8301:8301/udp -p 8302:8302 -p 8302:8302/udp -p 8400:8400 -p 8500:8500 \

-e 'CONSUL\_LOCAL\_CONFIG={"skip\_leave\_on\_interrupt": true}' \

-e CONSUL\_BIND\_INTERFACE=eth0 \

progrium/consul -server -bootstrap

#获取 node1 的IP

JOIN\_IP="$(docker inspect -f '{{.NetworkSettings.IPAddress}}' consul\_node1)"

#consul 服务端 node2

docker run --name consul\_node2 -d --restart=always \

-e 'CONSUL\_LOCAL\_CONFIG={"skip\_leave\_on\_interrupt": true}' \

-e CONSUL\_BIND\_INTERFACE=eth0 \

progrium/consul -h node2 –server -join $JOIN\_IP

#consul 服务端 node3

docker run --name consul\_node3 -d --restart=always \

-e 'CONSUL\_LOCAL\_CONFIG={"skip\_leave\_on\_interrupt": true}' \

-e CONSUL\_BIND\_INTERFACE=eth0 \

progrium/consul –h node3 -server -join $JOIN\_IP

agent : 表示启动 Agent 进程。

-server：表示启动 Consul Server 模式。

-client：表示启动 Consul Cilent 模式。

-bootstrap：表示这个节点是 Server-Leader ，每个数据中心只能运行一台服务器。技术角度上讲 Leader 是通过 Raft 算法选举的，但是集群第一次启动时需要一个引导 Leader，在引导群集后，建议不要使用此标志。

-ui：表示启动 Web UI 管理器，默认开放端口 8500，所以上面使用 Docker 命令把 8500 端口对外开放。

-node：节点的名称，集群中必须是唯一的。

-client：表示 Consul 将绑定客户端接口的地址，0.0.0.0 表示所有地址都可以访问。

-join：表示加入到某一个集群中去。 如：-json=192.168.1.23

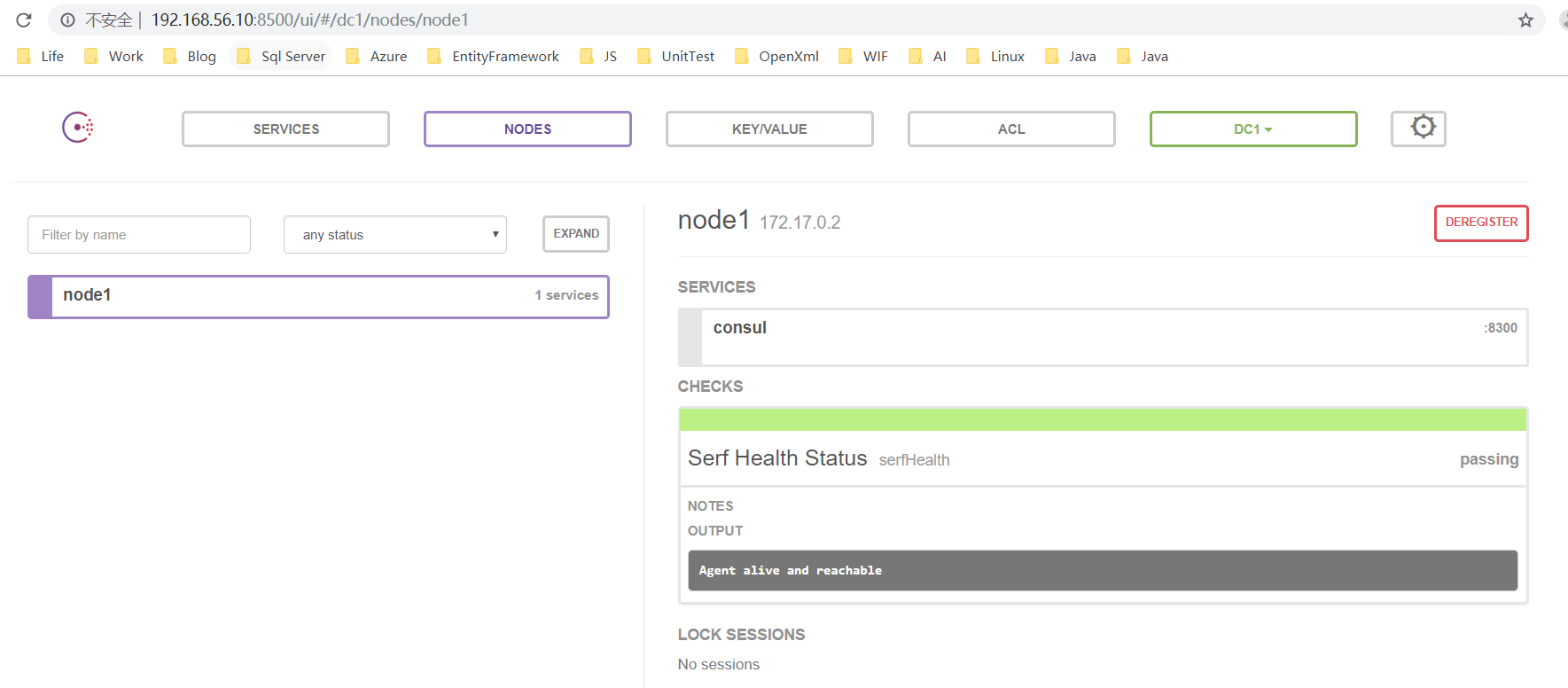
* 看集群成员

docker exec -t consul\_node1 consul members

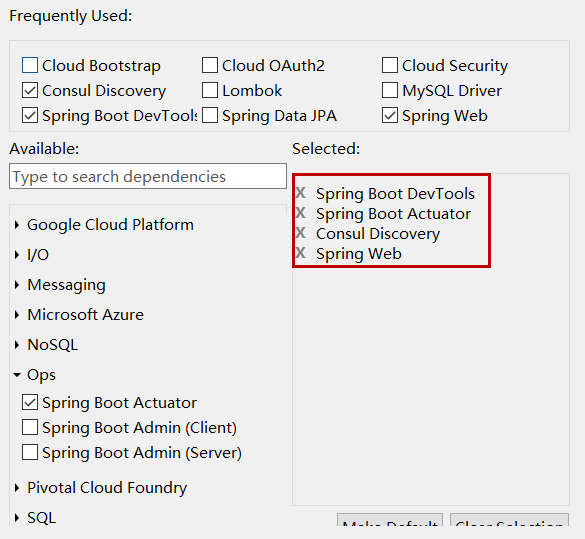
docker exec -t consul\_node1 consul operator raft list-peers

* + 1. **安装完成，使用WebUI查看结果**

访问路径：<http://192.168.56.10:8500/ui/#/dc1/services/consul>



* 1. **创建Consul的服务端**
     1. **创建SpringCloud项目，选择相关组件如下：**



pom.xml如下

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-consul-discovery</artifactId>

</dependency>

* + 1. **配置Consul服务端**

spring:

application:

name: service-provider

cloud:

consul:

discovery:

health-check-path: /actuator/health # 健康健康路径，也可以自己写

health-check-interval: 10s # 检测轮训时间 1m 代码1分钟

#instance-id: consul-client00 实例ID，唯一值

instance-id: ${spring.application.name}:${vcap.application.instance\_id:${spring.application.instance\_id:${random.value}}}

host: 192.168.56.10

port: 8500

* + 1. **设置启动类**

@EnableDiscoveryClient

@SpringBootApplication

public class SpringBootConsulApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootConsulApplication.class, args);

}

}

* + 1. **创建相关服务的RestController**

@RestController

public class HelloController {

@RequestMapping("/hello")

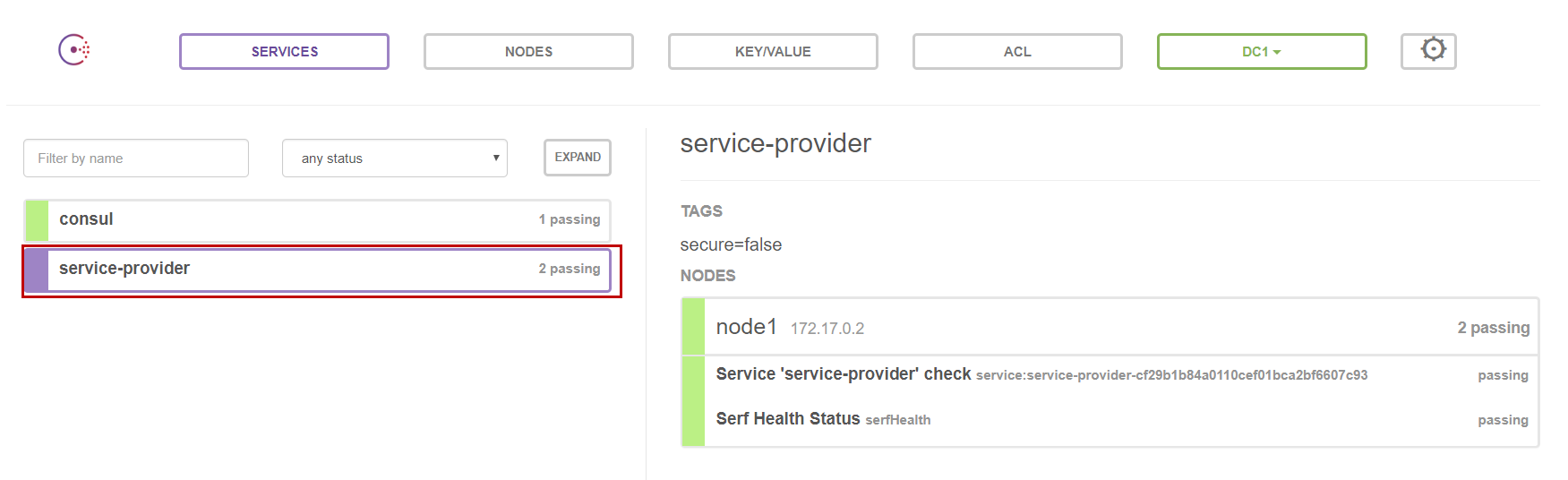
public @ResponseBody User hello() {

return new User(1, "hello consul");

}

}

* + 1. **启动项目，在Consul中查看已注册的服务**



* 1. **创建Consul的消费端**
     1. **创建SpringCloud项目（项目结构同消费端）**
     2. **配置Consul消费端**

server.port=8504

spring.application.name=service-consumer

spring.cloud.consul.host=192.168.56.10

spring.cloud.consul.port=8500

#设置不需要注册到 consul 中

spring.cloud.consul.discovery.register=false

* + 1. **测试调用服务端Rest服务**

@RestController

public class ServiceController {

@Autowired

private LoadBalancerClient loadBalancer;

@Autowired

private DiscoveryClient discoveryClient;

/\*\*

\* 获取所有服务

\*/

@RequestMapping("/services")

public Object services() {

return discoveryClient.getInstances("service-provider");

}

/\*\*

\* 从所有服务中选择一个服务（轮询）

\*/

@RequestMapping("/discover")

public Object discover() {

return loadBalancer.choose("service-provider ").getUri().toString();

}

/\*\*

\* 调用远程接口

\*

\* @return

\*/

@RequestMapping("/call")

public User call() {

ServiceInstance serviceInstance = loadBalancer.choose("service-provider");

System.out.println("服务地址：" + serviceInstance.getUri());

System.out.println("服务名称：" + serviceInstance.getServiceId());

User callServiceResult = new RestTemplate().getForObject(serviceInstance.getUri().toString() + "/hello",

User.class);

System.out.println(callServiceResult);

return callServiceResult;

}

}

Controller 中有俩个方法，一个是获取所有服务名为service-producer的服务信息并返回到页面，一个是随机从服务名为service-producer的服务中获取一个并返回到页面。

* + 1. **启动项目，访问Services服务：**<http://127.0.0.1:8504/services>，返回结果如下：

[{"instanceId":"service-provider-cf29b1b84a0110cef01bca2bf6607c93","serviceId":"service-provider","host":"192.168.2.115","port":8503,"secure":false,"metadata":{"secure":"false"},"uri":"http://192.168.2.115:8503","scheme":null}]

* + 1. **其他**
  1. **其他**

1. **其他**