# **[Registrator中文文档](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html)**

<https://www.cnblogs.com/zhangdk/p/Registrator_reference.html>

<http://www.mamicode.com/info-detail-2383285.html>

目录

* [快速入门](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E5%BF%AB%E9%80%9F%E5%85%A5%E9%97%A8)
  + [概述](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%A6%82%E8%BF%B0)
  + [准备](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E5%87%86%E5%A4%87)
  + [运行Registrator](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E8%BF%90%E8%A1%8Cregistrator)
  + [运行Redis](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E8%BF%90%E8%A1%8Credis)
  + [下一步](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E4%B8%8B%E4%B8%80%E6%AD%A5)
* [运行参考](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E8%BF%90%E8%A1%8C%E5%8F%82%E8%80%83)
  + [运行Registrator](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E8%BF%90%E8%A1%8Cregistrator-1)
  + [Docker选项](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "docker%E9%80%89%E9%A1%B9)
  + [Registrator选项](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "registrator%E9%80%89%E9%A1%B9)
  + [Consul ACL令牌](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-acl%E4%BB%A4%E7%89%8C)
  + [注册URI](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%B3%A8%E5%86%8Curi)
* [服务模型](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%9C%8D%E5%8A%A1%E6%A8%A1%E5%9E%8B)
  + [容器覆盖](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E5%AE%B9%E5%99%A8%E8%A6%86%E7%9B%96)
  + [发现服务](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E5%8F%91%E7%8E%B0%E6%9C%8D%E5%8A%A1)
  + [服务名称](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%9C%8D%E5%8A%A1%E5%90%8D%E7%A7%B0)
  + [IP和端口](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "ip%E5%92%8C%E7%AB%AF%E5%8F%A3)
  + [Tags和Attributes](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "tags%E5%92%8Cattributes)
  + [Unique ID](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "unique-id)
  + [示例](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E7%A4%BA%E4%BE%8B)
    - [缺省的单个服务](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E7%BC%BA%E7%9C%81%E7%9A%84%E5%8D%95%E4%B8%AA%E6%9C%8D%E5%8A%A1)
    - [指定元数据的单个服务](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%8C%87%E5%AE%9A%E5%85%83%E6%95%B0%E6%8D%AE%E7%9A%84%E5%8D%95%E4%B8%AA%E6%9C%8D%E5%8A%A1)
    - [缺省的多个服务](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E7%BC%BA%E7%9C%81%E7%9A%84%E5%A4%9A%E4%B8%AA%E6%9C%8D%E5%8A%A1)
    - [指定元数据的多个服务](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%8C%87%E5%AE%9A%E5%85%83%E6%95%B0%E6%8D%AE%E7%9A%84%E5%A4%9A%E4%B8%AA%E6%9C%8D%E5%8A%A1)
    - [使用labels定义元数据](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E4%BD%BF%E7%94%A8labels%E5%AE%9A%E4%B9%89%E5%85%83%E6%95%B0%E6%8D%AE)
* [注册后端](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%B3%A8%E5%86%8C%E5%90%8E%E7%AB%AF)
  + [Consul](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul)
    - [Consul HTTP Check](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-http-check)
    - [Consul HTTPS Check](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-https-check)
    - [Consul TCP Check](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-tcp-check)
    - [Consul Script Check](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-script-check)
    - [Consul TTL Check](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-ttl-check)
    - [Consul Initial Health Check Status](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-initial-health-check-status)
    - [Consul Critical Service Deregistration](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-critical-service-deregistration)
  + [Consul KV](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "consul-kv)
  + [Etcd](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "etcd)
  + [SkyDNS 2](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "skydns-2)
  + [Zookeeper Store](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "zookeeper-store)

# **快速入门**

这个简短的教程帮你快速开始使用Registrator。完整参看，查看[运行参考](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E8%BF%90%E8%A1%8C%E5%8F%82%E8%80%83)

## **概述**

Registrator监控新建的Docker容器，并且检查判定这些容器提供的服务。从我们的目的出发，任何监听在某个端口的程序都是服务。Registrator发现在容器内发现的任务服务，都将被添加到一个服务注册端，比如Consul或etcd。  
在这个教程中，我们将使用Registrator和Consul,运行一个Redis容器并自动添加到Consul。

## **准备**

我们需要一个运行Docker的主机，可以是一个本地的[boot2doker](http://boot2docker.io/)虚拟机和一个安装了docker client可以连接到虚拟机的shell。  
我们在容器里运行一个Server模式的consul实例

$ docker run -d --name=consul --net=host gliderlabs/consul-server -bootstrap

Consul在产品模式下的不是这样运行的，我们这里使用这种方式完成教程。现在我们可以访问通过Docker主机的IP访问Consul的HTTP API。

$ curl $(boot2doker ip):8500/v1/catalog/services

{"consul":[]}

现在我们可以启动Registrator了。

## **运行Registrator**

Registrator运行在每台主机上，我们这里只有一台主机，就运行一次就行。启动Registrator需要配置如何连接到注册机，即这里的Consul。  
除了标志选项，唯一需要的参数就是注册机URI。注册机URI编码了注册机类型，如何连接等选项。

$ docker run -d \

--name=registrator --net=host \

gliderlabs/registrator:latest \

consul://localhost:8500

先说一下Docker运行参数。首先，我们独立的运行容器和命名。我们也采用主机网络模式。这确保Registrator拥有实际主机的主机名和IP，也使Registrator更容易连接到Consul。我们也必须挂载Docker socket。

最后一行是Registrator运行参数，只有注册机URI。我们使用consul//localhost:8500，因为Registrator和Consul 运行在同一个网口。

$ docker logs registrator

我们应该可以看到Registrator运行起来并在“监听Docker事件"。Registrator正常运行了。

## **运行Redis**

现在当你启动的容器如果提供任何服务，他们将被添加到Consul。我们现在运行标准镜像库的Redis：

$ docker run -d -P --name=redis redis

我们使用-P发布所有端口,除了Registrator我们不经常这样使用。这样不仅发布了容器的所有端口，而且随机分配了一个主机端口。从Registrator和Consul提供服务发现的角度看，端口并不重要。尽管还有一些情况，你仍然想手动指定端口。

我们再来看看Consul的服务端：

$ curl $(boot2docker ip):8500/v1/catalog/services

{"consul":[],"redis":[]}

现在Consul已经有了一个redis服务。我们可以通过Consul的服端API查看更多诸如服务发布端口等信息：

$ curl $(boot2docker ip):8500/v1/catalog/service/redis

[{"Node":"boot2docker","Address":"10.0.2.15","ServiceID":"boot2docker:redis:6379","ServiceName":"redis","ServiceTags":null,"ServiceAddress":"","ServicePort":32768}]

如果我们移除redis容器，我们能够看到服务也从consul移除了。

$ docker rm -f redis

redis$ curl $(boot2docker ip):8500/v1/catalog/service/redis

[]

就到这了。我知道单独来看这并没什么意义，但是当服务注册到Consul后你可以做很多事情。当然，这超出了Registrator的范围，它做的就是把容器中的服务放进Consul。

## **下一步**

这还有很多方法配置Registrator，并且有很多方法运行容器，自定义服务。想了解这些，去看看[运行参考](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E8%BF%90%E8%A1%8C%E5%8F%82%E8%80%83)和[服务模型](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%9C%8D%E5%8A%A1%E6%A8%A1%E5%9E%8B)

# **运行参考**

Registrator设计是在每个主机上运行一次。你可以在一个集群中运行单个Registrator，但是在每个主机上运行可以使你或得更好的扩展属性和更简单的配置。从一定程度的自动化来说，每个主机都运行比在某个地方运行一次更简单。

## **运行Registrator**

docker run [docker options] gliderlabs/registrator[:tag] [options] <registry uri>

Registrator要求和推荐一些Docker选项，也有它自己的选项集，然后需要个注册机URI。下面是一个运行Registrator的经典方式：

$ docker run -d \

--name=registrator \

--net=host \

--volume=/var/run/docker.sock:/tmp/docker.sock \

gliderlabs/registrator:latest \

consul://localhost:8500

## **Docker选项**

| **Option** | **Required** | **Description** |
| --- | --- | --- |
| --volume=/var/run/docker.sock:/tmp/docker.sock | yes | 允许Registrator访问Docker API |
| --net=host | recommended | 帮助Registrator获取主机级的IP和主机名 |

与设置主机网络模式相比，另一个可选的方案是设置容器名字为宿主主机名（-h $HOSTNAME），并且使用下面Registrator的-ip选项。

## **Registrator选项**

| **Option** | **Since** | **Description** |
| --- | --- | --- |
| -cleanup | v7 | 清理悬挂服务 |
| -deregister | v6 | 取消注册退出的服务 "always"或"on-success".缺省值：always |
| -internal |  | 使用暴露端口代替发布端口 |
| -ip | v6 | 强制设置注册服务使用的IP地址 |
| -resync | v6 | 服务再同步频率。缺省值：0，never |
| -retry-attempts | v7 | 与注册机后台建立连接的最大重试次数 |
| -retry-interval | v7 | 重试间隔（以毫秒为单位） |
| -tags | v5 | 强制设置所有注册服务的都好分割的tags |
| -ttl |  | 服务TTL。缺省值0，no expiry(注册机后台唯一支持) |
| -ttl-refresh |  | 服务TTL刷新频率(注册机后台唯一支持) |
| -useIdFromLabel |  | 使用存储在给定label的IP地址，这个label在容器中设置，用以注册Consul。 |

如果设置了-internal选项，Registrator会注册docker内部IP和端口，而不是映射到主机的端口。

默认情况下，注册服务时，Registrator会尝试解析当前主机名来设置服务地址。如果你想强制指定服务地址为某个特定地址，你可以指定-ip参数。

对于支持TTL超期的注册机后端，Registrator支持设置和刷新服务的TTLs，使用-ttl和-ttl-refresh。

如果你想无限制的重连尝试，可以使用-retry-attempts -1。

-resync选项控制Registrator查询Docker中所有容器并且注册所有服务的频率。这个选项允许Registrator和服务注册机重新找到掉出同步的服务。要谨慎使用这个选项，它会通知已经注册到你服务上的所有观察者，可能会迅速淹没你的系统（比如consul-template就大量使用监测）。

## **Consul ACL令牌**

如果consul配置要求提供ACL令牌，Registrator需要知道，或者你会在consul的docker容器中看到警告。

[WARN] consul.catalog: Register of service 'redis' on 'hostname' denied due to ACLs

ACL令牌通过一个环境变量传入docker:CONSUL\_HTTP\_TOKEN。

$ docker run -d \

--name=registrator \

--net=host \

--volume=/var/run/docker.sock:/tmp/docker.sock \

-e CONSUL\_HTTP\_TOKEN=<your acl token> \

gliderlabs/registrator:latest \

consul://localhost:8500

## **注册URI**

<backend>://<address>[/<path>]

注册机后台使用URI来定义。架构是支持的注册机名字。地址是用来连接注册机的一个主机或者主机和端口。一些注册机支持一个定义的路径，比如作为前缀在服务定义中供基于注册机的key-value使用。

所以支持的后端参考，查看[注册机后端](https://www.cnblogs.com/zhangdk/p/Registrator_reference.html" \l "%E6%B3%A8%E5%86%8C%E6%9C%BA%E5%90%8E%E7%AB%AF)

# **服务模型**

Registrator主要关注的那些要被添加到服务发现注册机的服务。对我们而言，所有监听在某个端口的程序都是服务。如果一个容器监听了多个端口，它就又多个服务。

服务，包括来自容器的信息和用户在容器上定义的元数据被创建成一个服务对象。这个服务对象随后被传递给注册机后端，尝试放置到一个特定的注册项。

type Service struc {

ID string //unique service instance ID

Name string //service name

IP string //IP address service is located at

Port int //Port service is listening on

Tags []string //extra tags to classify service

Attrs map[string]string //extra attribute metadata

}

## **容器覆盖**

Name,Tags,Attrs,ID字段可以被用户自定义的容器元数据覆盖。你可以使用前缀SERVICE\_或者SERVICE\_x\_,其中x是内部暴露端口的环境变量或者标签设置这些值。例如，SERVICE\_NAME=customerdb和SERVICE\_80\_NAME=api

你在这些环境变量中使用的端口指的是在这个端口上的特定服务。名字中没有使用端口的元数据变量用作所有服务的缺省值，或者便捷的指向暴露的单个服务。

Attrs字段集合所有使用其他字段名的关键字，例如，SERVICE\_REGION=us-east。

因为元数据被存储为环境变量或者标签，因此容器作者可以在Dockerfile中包含他们自己的元数据定义。操作者仍然能够覆盖这些作者定义的缺省值。

## **发现服务**

缺省情况下，你可以期望Registrator从那些已经显式发布端口（例如使用'-p'或者-P）的容器中获取服务。这对于那些以主机网络模式运行的容器也是正确的，因此你必须发布端口，即使它没什么做任何网络智慧的事情。

$ docker run --net=host -p 8080:8080 -p 8443:8443 ...

如果使用-internal选项运行，相反它将寻找暴露的端口。这些可以在Dockerfile中隐式设置或者使用docker run --expose=8080...显式设定。

你也可以通过设置一个叫SERVICE\_IGNORE的标签或者环境变量告诉Registrator忽略一个容器。

如果你需要在某些容器中忽略几个服务，你可以使用SERVICE\_<port>\_IGNORE=true。

## **服务名称**

服务名是你在服务发现查找中使用的。缺省情况下，服务名按下面的格式确定：

<base(container-image)>[-<exposed-port> if >1 ports]

使用容器镜像的基础，如果镜像是gliderlabs/footbar，服务名就是footbar。如果镜像是redis,服务名就是简单的redis。

而且如果一个容器有多个暴露端口，它将各自追加内部暴露端口以区别。例如，一个镜像nginx有两个暴露端口，80和443,将产生两个服务，分别命名nginx-80和nginx-443。

你可以使用标签或者环境变量，SERVICE\_NAME或者'SERVICE\_x\_NAME'，其中x是内部暴露端口，覆盖这些缺省名字。注意如果一个容器有多个暴露端口，设置SERVICE\_NAME会导致多个服务命名为SERVICE\_NAME-<exposed port>。

## **IP和端口**

IP和端口组成了服务名解析的地址。这有许多方法Registrator能够依赖你的设置判断IP地址和端口。缺省情况下，端口就是发布的公共端口，IP将是你的主机IP。

由于自动判定正确的IP是困难的，推荐使用-ip选项显式告诉Registrator使用什么IP。

如果你使用-internal选项，Regisrator会使用暴露端口和docker分配的内部容器IP。

## **Tags和Attributes**

Tags和attributes是服务额外的元数据字段。并不是所有的后端都支持他们。事实上，目前consul支持tags，并且最近的v1.0.7以KV元数据的形式添加了对attributes的支持，但是没有其他的后端支持attributes。

Attributes也能被后端用来注册特定的特性，不仅仅是元数据。例如，consul使用他们指定specifying HTTP health checks。

## **Unique ID**

ID是服务示例在集群内的唯一标识。大部分情况下，它是一个实现细节，通常用户使用服务名而不是ID。Registrator使用一个人机友好的字符串，基于下面的格式编码了有用的信息在ID中：

<hostname>:<container-name>:<exposed-port>[:udp if udp]

ID包括了主机名，帮助你识别服务运行的主机。这也是Registrator运行在主机网络模式下或者设置Registrator的主机名为寄宿主机的主机名重要的原因。**否则它将是Registrator容器的ID,那没有什么用处**。

这个服务的容器名称也包含进来了。它使用容器名称代替容器ID，因为它更人性化，并且用户可配置。

为了识别出容器中这个服务，它使用内部暴露端口。这代表这个服务在容器内在这个端口上监听。我们使用这个是因为它比公共的发布端口更好的表达了这个服务。一个公共的端口可能是一个任意的54292，然而暴露的端口可能是80，表示它是一个HTTP服务。

最后，如果服务定义为UDP，这会被包括到ID中与监听在相同端口的TCP服务区别开。

尽管这可以使用容器的SERVICE\_ID或者SERVICE\_x\_ID覆盖，但是不推荐这样做。

## **示例**

### **缺省的单个服务**

$ docker run -d --name redis.0 -p 10000:6379 progrium/redis

Service结果是：

{

"ID": "hostname:redis.0:6379",

"Name": "redis",

"Port": 10000,

"IP": "192.168.1.102",

"Tags": [],

"Attrs": {}

}

### **指定元数据的单个服务**

$ docker run -d --name redis.0 -p 10000:6379 \

-e "SERVICE\_NAME=db" \

-e "SERVICE\_TAGS=master,backups" \

-e "SERVICE\_REGION=us2" progrium/redis

Service结果是:

{

"ID": "hostname:redis.0:6379",

"Name": "db",

"Port": 10000,

"IP": "192.168.1.102",

"Tags": ["master", "backups"],

"Attrs": {"region": "us2"}

}

记住并不是所有的Service对象都会被注册后端使用。例如，目前不支持注册任意attributes。这个字段留作将来使用。  
逗号可能可以通过反斜杠转义，像下面的例子：

$ docker run -d --name redis.0 -p 10000:6379 \

-e "SERVICE\_NAME=db" \

-e "SERVICE\_TAGS=/(;\\,:-\_)/" \

-e "SERVICE\_REGION=us2" progrium/redis

### **缺省的多个服务**

$ docker run -d --name nginx.0 -p 4443:443 -p 8000:80 progrium/nginx

两个Service对象的结果：

[

{

"ID": "hostname:nginx.0:443",

"Name": "nginx-443",

"Port": 4443,

"IP": "192.168.1.102",

"Tags": [],

"Attrs": {},

},

{

"ID": "hostname:nginx.0:80",

"Name": "nginx-80",

"Port": 8000,

"IP": "192.168.1.102",

"Tags": [],

"Attrs": {}

}

]

### **指定元数据的多个服务**

$ docker run -d --name nginx.0 -p 4443:443 -p 8000:80 \

-e "SERVICE\_443\_NAME=https" \

-e "SERVICE\_443\_ID=https.12345" \

-e "SERVICE\_443\_SNI=enabled" \

-e "SERVICE\_80\_NAME=http" \

-e "SERVICE\_TAGS=www" progrium/nginx

两个Service对象的结果：

[

{

"ID": "https.12345",

"Name": "https",

"Port": 4443,

"IP": "192.168.1.102",

"Tags": ["www"],

"Attrs": {"sni": "enabled"},

},

{

"ID": "hostname:nginx.0:80",

"Name": "http",

"Port": 8000,

"IP": "192.168.1.102",

"Tags": ["www"],

"Attrs": {}

}

]

### **使用labels定义元数据**

$ docker run -d --name redis.0 -p 10000:6379 \

-l "SERVICE\_NAME=db" \

-l "SERVICE\_TAGS=master,backups" \

-l "SERVICE\_REGION=us2" dockerfile/redis

Service结果：

{

"ID": "hostname:redis.0:6379",

"Name": "db",

"Port": 10000,

"IP": "192.168.1.102",

"Tags": ["master", "backups"],

"Attrs": {"region": "us2"}

}

# **注册后端**

Registrator支持很多后端注册机。为了发挥Registrator的用途，你需要运行其中一个。下面是被支持后端使用的注册URI和它们的特性文档。

查看[Contributing Backends](https://gliderlabs.com/registrator/latest/dev/backends/)

## **Consul**

consul://<address>:<port>consul-unix://<filepath>consul-tls://<address>:<port>

Consul是推荐的注册机，因为它专门为服务发现提供了健康检查服务。

如果没有指定地址和端口，默认将使用127.0.0.1:8500。

consul支持tags，但不是任意服务attributes。

当使用consul-tls架构，Registrator通过TLS与Consul通信。你必须设置下面的环境变量：CONSUL\_CAERT:CA 文件位置, CONSUL\_TLSKEY:Key位置

了解更多关于Consul检查参数信息，查看[API documentation](https://www.consul.io/api/agent/check.html" \l "register-check)。

### **Consul HTTP Check**

这个特性只能在Consul 0.5及更新版本中使用。容器通过环境变量或者标签指定的这些额外的元数据用来在服务中注册一个HTTP健康检查。

SERVICE\_80\_CHECK\_HTTP=/health/endpoint/path

SERVICE\_80\_CHECK\_INTERVAL=15s

SERVICE\_80\_CHECK\_TIMEOUT=1s # optional, Consul default used otherwise

它适用于任何端口的服务，不仅仅是80端口。如果它是唯一的服务，你也可以使用

SERVICE\_CHECK\_HTTP

### **Consul HTTPS Check**

这个特性只能在Consul 0.5及更新版本中使用。容器通过环境变量或者标签指定的这些额外的元数据用来在服务中注册一个HTTPS健康检查。

SERVICE\_443\_CHECK\_HTTPS=/health/endpoint/path

SERVICE\_443\_CHECK\_INTERVAL=15s

SERVICE\_443\_CHECK\_TIMEOUT=1s # optional, Consul default used otherwise

### **Consul TCP Check**

这个特性只能在Consul 0.6及更新版本中使用。容器通过环境变量或者标签指定的这些额外的元数据用来在服务中注册一个TCP健康检查。

SERVICE\_443\_CHECK\_TCP=trueSERVICE\_443\_CHECK\_INTERVAL=15sSERVICE\_443\_CHECK\_TIMEOUT=3s # optional, Consul default used otherwise

### **Consul Script Check**

这个特性很难用，因为它让你指定一个脚本检查从Consul运行。如果在容器中运行Consul，你就被限制只能在那个容器中运行。例如，curl必须按照以使用这个特性：

SERVICE\_CHECK\_SCRIPT=curl --silent --fail example.com

任务no-TTL检查的缺省间隔是10秒,但是你可以使用\_CHECK\_INTERVAL设置。检查命令可以使用$SERVICE\_IP和$SERVICE\_PORT占位符插值：

SERVICE\_CHECK\_SCRIPT=nc $SERVICE\_IP $SERVICE\_PORT | grep OK

### **Consul TTL Check**

你也可以向consul注册一个TTL Check。记住这意味着Consul将期待一个常规的心跳ping到他的API,用以保持服务标志健康。

SERVICE\_CHECK\_TTL=30s

### **Consul Initial Health Check Status**

缺省的当一个服务在Consul注册时，状态被设置为"critical"。你可以指定初始的健康检查状态：

SERVICE\_CHECK\_INITIAL\_STATUS=passing

### **Consul Critical Service Deregistration**

Consul可能撤销注册一个服务，如果检查在critical状态超过设定的时间值。如果启用，这应该比可期待的恢复故障更长。

SERVICE\_CHECK\_DEREGISTER\_AFTER=10m

## **Consul KV**

consulkv://<address>:<port>/<prefix>

consulkv-unix://<filepath>:/<prefix>

这是一个分离后端，使用Consul的key-value存储代替它的本地服务目录。这种表现更像etcd，因为它有相似的语义，但是目前不支持TTLs。

如果没有指定地址和端口，默认是127.0.0.1:8500。

使用Registry URI前缀，服务定义存储为下面的格式：

<prefix>/<service-name>/<service-id> = <ip>:<port>

## **Etcd**

etcd://<address>:<port>/<prefix>

Etcd工作方式与Consul KV相似，而且支持服务TTLs。它目前也不支持服务attributes和tags。

如果没有指定地址和端口，它将使用默认值127.0.0.1:2379。

使用Registry URI作为前缀，服务定义被储存为下面的格式：

<prefix>/<service-name>/<service-id> = <ip>:<port>

## **SkyDNS 2**

skydns2://<address>:<port>/<domain>

SkyDNS2 使用etcd,因此这个后端写服务定义的格式兼容SkyDNS2。path不能被省略，必须是一个对SkyDNS有效的DNS域名。

如果没有指定地址和端口，它将是默认值：127.0.0.1:2379。

使用Registry URI和域名cluster.local，服务定义存储为下面的格式：

/skydns/local/cluster/<service-name>/<service-id> = {"host":"<ip>","port":<port>}

SkyDNS要求服务ID是一个有效的DNS主机名，因此这个后端要求容器用一个有效的DNS名字覆盖服务ID。例如：

$ docker run -d --name redis-1 -e SERVICE\_ID=redis-1 -p 6379:6379 redis

## **Zookeeper Store**

Zookeeper后端让你发布临时的znodes到zookeeper。这个模式在指定zookeeper路径后可以使用。zookeeper后端支持发布一个json格式的znode body，完成定义服务attributes/tags，也包括服务名和容器id。URI示例：

$ registrator zookeeper://zookeeper.host/basepath

$ registrator zookeeper://192.168.1.100:9999/basepath

在zookeeper URI指定的base path中，registrator将为服务创建下面的包括一个JSON入口路径树：

<service-name>/<service-port> = <JSON>

JSON将包括发布容器服务的所有信息。例如下面的容器启动：

docker run -i -p 80 -e 'SERVICE\_80\_NAME=www' -t ubuntu:14.04 /bin/bash

将产生的zookeeper path和JSON znode body：

/basepath/www/80 = {"Name":"www","IP":"192.168.1.123","PublicPort":49153,"PrivatePort":80,"ContainerID":"9124853ff0d1","Tags":[],"Attrs":{}}

分类: [微服务](https://www.cnblogs.com/zhangdk/category/1259895.html" \t "_blank)

标签: [Registrator](https://www.cnblogs.com/zhangdk/tag/Registrator/), [Registrator用户手册](https://www.cnblogs.com/zhangdk/tag/Registrator%E7%94%A8%E6%88%B7%E6%89%8B%E5%86%8C/), [Registrator中文文档](https://www.cnblogs.com/zhangdk/tag/Registrator%E4%B8%AD%E6%96%87%E6%96%87%E6%A1%A3/), [Consul](https://www.cnblogs.com/zhangdk/tag/Consul/)