Contents

[Service registry and Service discovery 1](#_Toc454973760)

[基本概念 1](#_Toc454973761)

[相关算法 2](#_Toc454973762)

[工具 2](#_Toc454973763)

# Service registry and Service discovery

## 基本概念

A service registry is useful because it enables client-side load-balancing and decouples service providers from consumers without the need for DNS.

1）什么是服务发现？

服务发现组件记录了（大规模）分布式系统中所有服务的信息，人们或者其它服务可以据此找到这些服务。 DNS 就是一个简单的例子,服务发现提供了一种协调机制，方便服务的发布和查找。

2）服务发现应该具备哪些关键特性

服务发现是支撑大规模 SOA 的核心服务，它必须是高可用的，提供注册、目录和查找三大关键特性

服务元数据存储是服务发现的关键，因为复杂的服务提供了多种服务接口和端口，部署环境也比较复杂。一旦服务发现组件存储了大量元数据，它就必须提供强大的查询功能，包括服务健康和其它状态的查询。

3）服务发现带来的主要好处

服务发现的主要好处是「零配置」：不用使用硬编码的网络地址，只需服务的名字（有时甚至连名字都不用）就能使用服务。

服务发现组件必须提供查询所有服务的部署状态和集中控制所有服务实例的手段

服务发现解决方案: Consul, etcd 和 doozerd,ZooKeeper

服务发现门户的工作方式是：当每一个服务启动上线之后，他们通过发现工具来注册自身信息。它记录了一个相关组件若想使用某服务时的全部必要信息。例如，一个MySQL数据库服务会在这注册它运行的ip和端口，如有必要，登录时的用户名和密码也会留下。

当一个服务的消费者上线时，它能够在预设的终端查询该服务的相关信息。然后它就可以基于查到的信息与其需要的组件进行交互。负载均衡就是一个很好的例子，它可以通过查询服务发现得到各个后端节点承受的流量数，然后根据这个信息来调整配置。

这可将配置信息从容器内拿出。一个好处是可以让组件容器更加灵活，并不受限于特定的配置信息。另一个好处是使得组件与一个新的相关服务实例交互时变得简单，可以由管理工具动态进行调整配置。

## 相关算法

强一致性协议:按照某一顺序串行执行存储对象读写操作， 更新存储对象之后， 后续访问总是读到最新值

Raft 算法

## 工具

Consul is a HashiCorp tool for service discovery, service registry, and health checks.

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

1. Consul must first be installed on every node that will be a member of the Consul cluster

推荐使用vagrant+virtualbox+script启动虚机，虚机与主机的共享目录/vagrant，虚机通过终端操作，完整的操作系统

$sudo cp /vagrant/web.json /etc/consul.d

1. RUN THE CONSUL AGENT

The agent can run either in server or client mode

A client is a very lightweight process that registers services, runs health checks, and forwards queries to servers. The agent must be run on every node that is part of the cluster.

$ consul agent -dev #agent is running in server mode and has claimed leadership of the cluster

if error occurs like “Problem: Error starting agent: Failed to get advertise address: Multiple private IPs found”

please refer to <http://blog.csdn.net/sunny_forever/article/details/51148311>

$consul agent -dev -bind your-ip

$ consul members #see the members of the Consul cluster

$ curl localhost:8500/v1/catalog/nodes

Ctrl-C #Stopping the Agent

By gracefully leaving, Consul notifies other cluster members that the node left. If you had forcibly killed the agent process, other members of the cluster would have detected that the node failed. When a member leaves, its services and checks are removed from the catalog. When a member fails, its health is simply marked as critical, but it is not removed from the catalog. Consul will automatically try to reconnect to failed nodes, allowing it to recover from certain network conditions, while left nodes are no longer contacted.

1. REGISTERING SERVICES

A service can be registered either by providing a service definition

etc

consul.d #the .d suffix implies "this directory contains a set of configuration files").

web.json

weg.json

{

"service": {

"name": "web",

"tags": ["rails"],

"port": 80

}

}

$ consul agent -dev -config-dir /etc/consul.d

the agent loaded the service definition from the configuration file, and registered it in the service catalog.

If you wanted to register multiple services, you could create multiple service definition files in the Consul configuration directory.

Querying Services

query the service using either the DNS or HTTP API.

$ dig @127.0.0.1 -p 8600 web.service.consul #the DNS name for services is NAME.service.consul

$ curl http://localhost:8500/v1/catalog/service/web #The catalog API gives all nodes hosting a given service

1. CONSUL CLUSTER

**Starting the Agents**

When a Consul agent is started, it begins without knowledge of any other node: it is an isolated cluster of one. To learn about other cluster members, the agent must join an existing cluster

vagrant@n1:~$ consul agent -server -bootstrap-expect 1 \

-data-dir /tmp/consul -node=agent-one -bind=172.20.20.10 \

-config-dir /etc/consul.d

vagrant@n2:~$ consul agent -data-dir /tmp/consul -node=agent-two \

-bind=172.20.20.11 -config-dir /etc/consul.d

-node: By default, Consul uses the hostname of the machine, but we'll manually override it using the -node

-bind: By default, Consul will listen on the first private IP on a system, bind address that Consul listens on

-server: The first node will act as our sole server in this cluster

-bootstrap-expect: hints to the Consul server the number of additional server nodes we are expecting to join.The purpose of this flag is to delay the bootstrapping of the replicated log until the expected number of servers has successfully joined.

-config-dir: marking where service and check definitions can be found.

two Consul agents running: one server and one client. The two Consul agents still don't know anything about each other and are each part of their own single-node clusters. You can verify this by running consul members against each agent and noting that only one member is visible to each agent.

**Joining a Cluster**

vagrant@n1:~$ consul join 172.20.20.11 #tell the first agent to join the second agent

If you run consul members against each agent, you'll see that both agents now know about each other.

Remember: To join a cluster, a Consul agent only needs to learn about one existing member. After joining the cluster, the agents gossip with each other to propagate full membership information.

Auto-joining a Cluster on Start

$ consul agent -atlas-join \

-atlas=ATLAS\_USERNAME/infrastructure \

-atlas-token="YOUR\_ATLAS\_TOKEN"

Ideally, whenever a new node is brought up in your datacenter, it should automatically join the Consul cluster without human intervention

Querying Nodes

Just like querying services, Consul has an API for querying the nodes themselves. You can do this via the DNS or HTTP API.For the DNS API, the structure of the names is NAME.node.consul or NAME.node.DATACENTER.consul

vagrant@n1:~$ dig @127.0.0.1 -p 8600 agent-two.node.consul

1. Health Checks

etc

consul.d #the .d suffix implies "this directory contains a set of configuration files").

web.json

ping.json

{

"check": {

"name": "ping",

"script": "ping -c1 google.com >/dev/null",

"interval": "30s"

}

}

vagrant@n1:~$ consul agent -server -bootstrap-expect 1 \

-data-dir /tmp/consul -node=agent-one -bind=172.20.20.10 \

-config-dir /etc/consul.d

Checking Health Status

vagrant@n1:~$ curl http://localhost:8500/v1/health/state/critical | python -m json.tool

1. Key/Value Data

In addition to providing service discovery and integrated health checking, Consul provides an easy to use Key/Value store

$ curl -v http://localhost:8500/v1/kv/?recurse

返回结果为404 response,因为还没有数据

$ curl -X PUT -d 'test' http://localhost:8500/v1/kv/web/key1 #存数据

$ curl http://localhost:8500/v1/kv/?recurse #查询数据

$ curl http://localhost:8500/v1/kv/web/key1

1. Web UI

UIs can be used for viewing all services and nodes, for viewing all health checks and their current status, and for reading and setting key/value data.

set up the self-hosted UI

$ consul agent -ui

http://localhost:8500/ui.