

# 云计算导论实践----云原生应用实践

前面的章节介绍了 K8S 这种云计算管理平台，但是考虑到其安装使用比较复杂，对初学者来说，使用 K8S 部署云原生的应用可能颇具挑战性。故本章使用一种十分轻量化、便捷化的云计算平台及一系列模块化组件来部署一个简单的 **nginx** 云原生应用，以便从中体会到云原生中微服务架构的中种种优势，并对云计算有进一步的体会。

如果使用物理机器设备部署，那么可以简单地使用一个千兆交换机来搭建一个小型局域网，在局域网中的几台机器上都要先安装 **nomad** 软件，之后分别部署一个服务端机器和多个客户端机器。

如果使用云服务器虚拟机的方式（如何创建华为云虚拟机请参考前几章实验），那么需要保证集群中各个机器处在同一个网络中。下面是本实验创建的 3 台华为云虚拟机，其配置及 IP 地址如图 7-3 所示。

<input type="checkbox"/>	名称/ID	监控	安全	状态	可用区	规格/镜像	操作系统...	IP地址
<input type="checkbox"/>	mjq-hw-0001 ae64bcb3-98ac-472f-9...			运行中	可用区7	2vCPUs   4GiB   kc1.large.2 Ubuntu 18.04 server 64bit with ...	Linux	121.36.19.156 (... 192.168.0.6 (私有)
<input type="checkbox"/>	mjq-hw-0002 50542ebf-c25f-4070-8...			运行中	可用区7	2vCPUs   4GiB   kc1.large.2 Ubuntu 18.04 server 64bit with ...	Linux	1.92.115.63 (弹... 192.168.0.66 (私...
<input type="checkbox"/>	mjq-hw-0003 ec1feb11-a746-4b9b-a...			运行中	可用区7	2vCPUs   4GiB   kc1.large.2 Ubuntu 18.04 server 64bit with ...	Linux	120.46.205.196 (... 192.168.0.250 (...

图 7-3 华为云虚拟机的配置及 IP 地址

## 7.1 nomad 安装及部署

**nomad** 是一个高度可扩展的、分布式、多数据中心的任务调度器。它支持容器化、虚拟机和静态二进制应用的调度，提供统一的接口来调度这些不同类型的任务。**nomad** 的设计目标是简单、灵活和高效，旨在简化部署和运维工作，同时保持高性能和可靠性。它能够自动处理任务调度、资源分配、服务注册和健康检查等工作，支持横向扩展以适应不断增长的工作负载。可以将其看作类似于 K8S 的一种云计算管理平台，但是 **nomad** 更加地轻量化。准确地讲，可以将其视为 K8S 的核心调度组件。

### 1. 安装 nomad

在机器上安装 **nomad** 平台的方式有多种，在有网络的情况下可以使用 **apt** 仓库进行快捷安装，或是将针对特定系统架构预编译后的二进制文件手动安装至 **/bin** 目录。

在部分系统（如麒麟系统）中，**apt** 仓库中软件版本较落后，并且无法添加 **nomad** 所在仓库，所以只能选择手动安装二进制文件的方式。

#### （1）apt 安装（部分系统不适用）

步骤 1：安装依赖包，代码如下：

```
sudo apt-get update && \sudo apt-get install wget gpg coreutils
```

步骤 2: 在 apt 管理器中添加 hashicorp (nomad 平台开发公司) 的 key, 代码如下:

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
```

步骤 3: 在 apt 工具中, 添加 hashicorp 的仓库, 代码如下:

```
echo "deb [signed-by = /usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $ (lsb_release -cs)
main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
```

步骤 4: 最后使用 apt install 命令安装, 代码如下:

```
sudo apt-get update && sudo apt-get install nomad
```

## (2) 二进制文件安装

可以在 nomad 官网下载预编译完的二进制执行文件, nomad 官网下载地址如图 7-4 所示, 也可以直接使用本书资源包或百度网盘下载 nomad 文件。

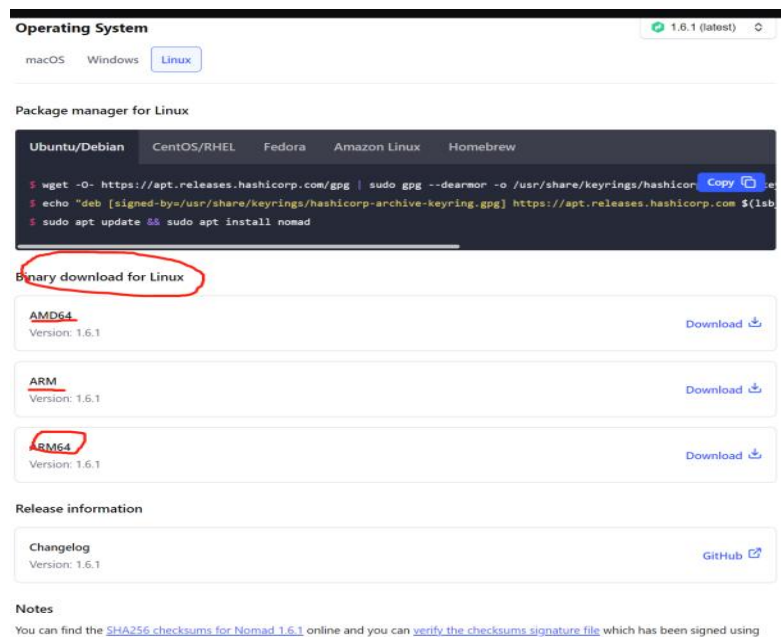


图 7-4 nomad 官网下载地址

将其上传到虚拟机并放到 /usr/bin 中, 再添加执行权限, 然后就可以通过 nomad -v 来查看是否安装成功了, 命令如下, 检查 nomad 安装结果如图 7-5 所示。

```
sudo mv nomad /usr/bin
sudo chmod +x /usr/bin/nomad
```

```
root@mjq-hw-0001:~# sudo mv nomad /usr/bin
root@mjq-hw-0001:~# sudo chmod +x /usr/bin/nomad
root@mjq-hw-0001:~# nomad -v
Nomad v1.6.1
BuildDate 2023-07-21T13:49:42Z
Revision 515895c7690cdc72278018dc5dc58aca41204ccc
root@mjq-hw-0001:~#
```

图 7-5 检查 nomad 安装结果

## 2. 搭建集群

集群分为服务端和客户端，其中服务端负责分发任务（通过 job 文件），客户端负责接收服务端分发的任务，然后在本地选择驱动执行该任务。

注：如果使用的集群机器同时连接到了多个局域网，那么配置文件中需要指定服务/客户端监听和运行的 IP 地址网段。

### （1）服务端部署

服务端的部署需要用特定的硬件描述语言（hand ware clescription language, HCL）配置文件来启动 nomad，可在服务端创建 nomad 文件夹，并在文件夹中创建 server.hcl 文件，server.hcl 文件的内容如下：

```
# Increase log verbosity
log_level = "DEBUG"
# Setup data dir
data_dir = "/tmp/server1"
# Give the agent a unique name. Defaults to hostname
name = "server1"
# Enable the server
server {
  enabled = true
  # Self-elect, should be 3 or 5 for production
  bootstrap_expect = 1
}
```

在 nomad 目录下打开终端执行以下命令，然后让该终端一直运行。

```
sudo nomad agent -config- server.hcl
```

### （2）客户端部署

客户端的部署需要用特定的 HCL 配置文件来启动 nomad。在所有需要充当客户端的虚拟机/设备上，创建 nomad 文件夹，使用命令 vim client1.hcl 创建文件，然后写入配置信息，命令内容如下，客户端 nomad 部署完成的结果如图 7-6 所示。

```
# Increase log verbosity
log_level = "DEBUG"
```

```
# Setup data dir
data_dir = "/tmp/client1"

# Give the agent a unique name. Defaults to hostname
name = "client1"

# Enable the client
client {
  enabled = true

  # For demo assume we are talking to server1 - in my case this is IP 192.168.0.6
  # For production, this should be like "nomad.service.consul:4647" and a system
  # like Consul used for service discovery.
  servers = ["192.168.0.6"]
  options = {
    "driver.raw_exec.enable" = "1"
  }

  host_volume "scratch" {
    path      = "/opt/nomad/scratch"
    read_only = false
  }
}

# Modify our port to avoid a collision with server1
ports {
  http = 5656
}

# Because we will potentially have two clients talking to the same
# Docker daemon, we have to disable the dangling container cleanup,
# otherwise they will stop each other's work thinking it was orphaned.

plugin "docker" {
  config {
    gc {
```



```
2024-02-18T09:08:21.489Z [DEBUG] http: UI is enabled
2024-02-18T09:08:21.489Z [DEBUG] http: UI is enabled
2024-02-18T09:08:21.582Z [DEBUG] client.server_mgr: new server list: new_servers=[10.0.0.56:4647] old_servers=[192.168.89.47:4647]
2024-02-18T09:08:21.582Z [INFO] client: node registration complete
2024-02-18T09:08:21.582Z [DEBUG] client: updated allocations: index=1 total=0 pulled=0 filtered=0
2024-02-18T09:08:21.587Z [DEBUG] client: allocation updates: added=0 removed=0 updated=0 ignored=0
2024-02-18T09:08:21.587Z [DEBUG] client: allocation updates applied: added=0 removed=0 updated=0 ignored=0 errors=0
2024-02-18T09:08:21.636Z [DEBUG] client: state updated: node status=ready
2024-02-18T09:08:29.751Z [DEBUG] client: state changed, updating node and re-registering
2024-02-18T09:08:29.865Z [INFO] client: node registration complete
=> Newer Nomad version available: 1.7.5 (currently running: 1.6.1)
2024-02-18T09:13:25.162Z [DEBUG] client: updated allocations: index=1 total=0 pulled=0 filtered=0
2024-02-18T09:13:25.162Z [DEBUG] client: allocation updates: added=0 removed=0 updated=0 ignored=0
2024-02-18T09:13:25.162Z [DEBUG] client: allocation updates applied: added=0 removed=0 updated=0 ignored=0 errors=0
```

图 7-7 客户端部署完毕的结果

到此为止，已经完成了一个客户端的部署，如果需要创建多个客户端，仿照上述步骤即可。例如若要创建 client2，只需要将 client1.hcl 文件中的 name 及 data\_dir 字段修改为 client2 即可。

3. 查看集群状态

(1) 浏览器 GUI 查看

在客户端运行起来后，可以通过浏览器进行访问，访问 url 为 http://[server\_ip]:4646/ui。如果是在本机上运行的服务端，可以将[server\_ip]替换为 localhost。在其他机器上，也可以通过 IP 地址或域名访问，只要将[server\_ip]替换为服务端运行的 IP 地址即可（在本例中为 121.36.19.156）。CVI 查看服务端状态如图 7-8 所示，CUI 查看客户端状态如图 7-9 所示，CVI 查看集群性能情况如图 7-10 所示。通过这些方式可以查看任务、当前客户端、服务端，以及任务所占空间以及执行状态。

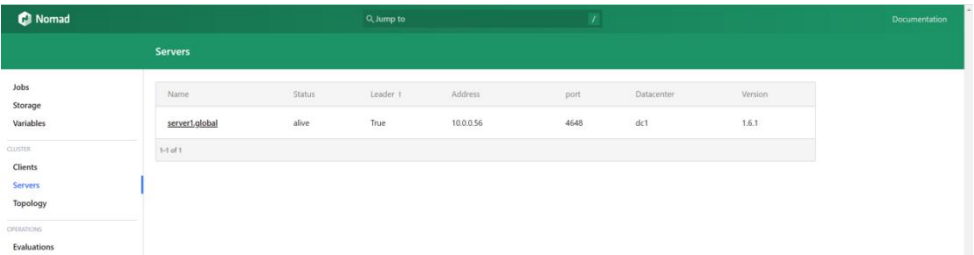


图 7-8 GUI 查看服务端状态

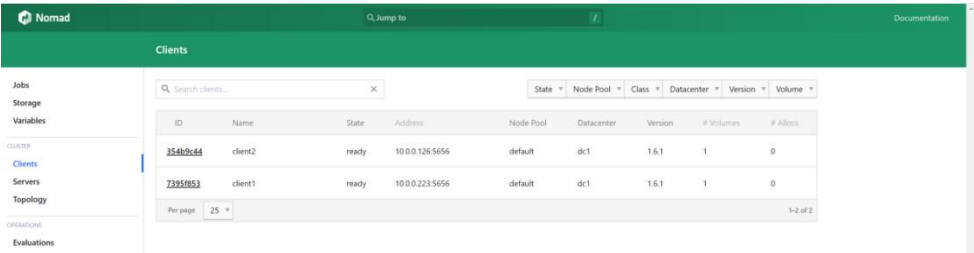


图 7-9 GUI 查看客户端状态

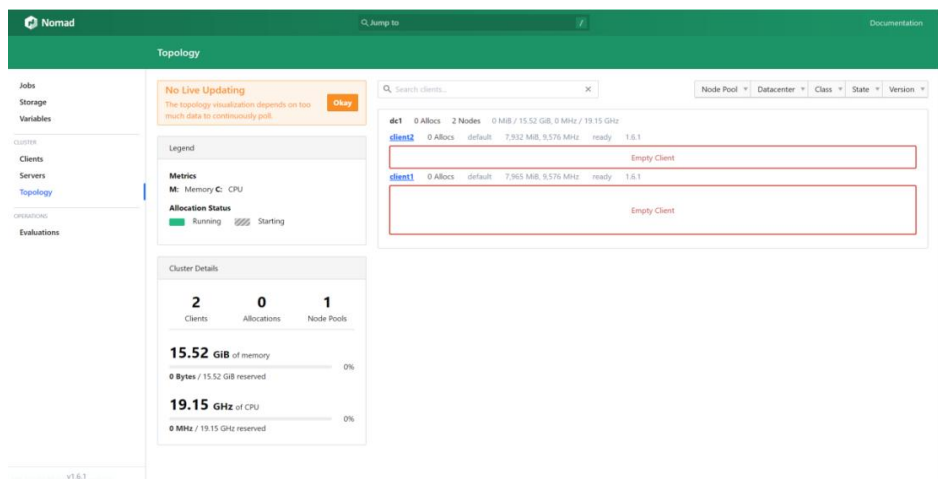


图 7-10 GUI 查看集群性能情况

## (2) 命令行查看

可以使用以下命令查看当前集群的状态。要注意将下文命令中的地址替换为实际运行的服务器的地址，命令查看集群状态如图 7-11 所示。

```
nomad node status -address = http://192.168.0.6:4646
```

```
root@mq-hw-0001:~/consul# nomad node status -address=http://192.168.0.6:4646
ID      Node Pool DC Name Class Drain Eligibility Status
3b9e3234 default dc1 client1 <none> false eligible ready
06770610 default dc1 client1 <none> false eligible ready
```

图 7-11 命令查看集群状态

## 7.2 consul 安装及部署

consul 是一个服务网格解决方案，提供了服务发现、健康检查、键值存储、安全的服务间通信和多云数据中心支持。consul 使构建分布式、微服务架构的应用变得更加简单和可靠，通过自动化服务注册和发现，减少了配置的复杂性和维护工作。Consul 与 nomad 是同一家公司开发的软件，可以将 consul 视作 nomad 的配套云计算组件，相当于 K8S 中的服务管理组件。

### 1. consul 安装

#### (1) 二进制文件安装

安装过程如 nomad 的二进制安装过程，可从官网下载 consul 的二进制文件。

步骤 1：先向虚拟机/设备上传 consul 的二进制文件，如图 7-12 所示。

```
root@mq-hw-0001:~/consul# ls
consul server.json
root@mq-hw-0001:~/consul#
```

图 7-12 向虚拟机上传 consul 的二进制文件

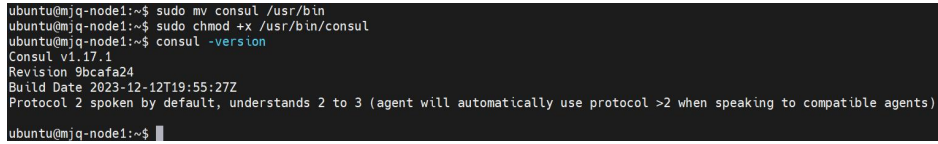
步骤 2：执行 mv 命令将其放到 bin 下，并增加执行权限，代码如下：

```
sudo mv consul /usr/bin
sudo chmod +x /usr/bin/consul
```



步骤 3：检查是否安装成功，代码如下，consul 安装成功的显示如图 7-13 所示。

```
consul -version
```



```
ubuntu@mq-node1:~$ sudo mv consul /usr/bin
ubuntu@mq-node1:~$ sudo chmod +x /usr/bin/consul
ubuntu@mq-node1:~$ consul -version
Consul v1.17.1
Revision 9bcafa24
Build Date 2023-12-12T19:55:27Z
Protocol 2 spoken by default, understands 2 to 3 (agent will automatically use protocol >2 when speaking to compatible agents)
ubuntu@mq-node1:~$
```

图 7-13 consul 安装成功的显示

## 2. 服务端部署

步骤 1：先创建挂载目录，代码如下：

```
sudo mkdir -p /var/lib/consul
```

步骤 2：同样编写服务端的配置文件，文件名为 `server.json`（此处示例的服务端用 192.168.0.6），内容如下：

```
{
  "server": true,
  "bootstrap_expect": 1,
  "ui_config": {
    "enabled": true
  },
  "data_dir": "/var/lib/consul",
  "bind_addr": "192.168.0.6",
  "advertise_addr": "192.168.0.6",
  "node_name": "consul-server-1",
  "datacenter": "dc1",
  "log_level": "INFO",
  "enable_syslog": true,
  "addresses": {
    "http": "0.0.0.0"
  }
}
```

步骤 3：启动 consul，代码如下：

```
sudo consul agent -config-file = server.json
```

## 3. 客户端部署

步骤 1：先创建挂载目录，代码如下：

```
sudo mkdir -p /var/lib/consul
```



步骤 2：同样编写各个客户端的配置文件 `client.json`（主要设置节点 IP 地址和服务端 IP 地址），在此例中，192.168.0.66 为本客户端 IP 地址，192.168.0.6 为服务端 IP 地址，具体内容如下：

```
{
  "datacenter": "dc1",
  "data_dir": "/var/lib/consul",
  "server": false,
  "bind_addr": "192.168.0.66",
  "advertise_addr": "192.168.0.66",
  "retry_join": ["192.168.0.6"]
}
```

步骤 3：启动 `consul`，检查部署是否成功，代码如下，启动 Consul 的结果如图 7-14 所示。

```
sudo consul agent -config-file = client.json
```

```
root@mq-hw-0003:~# sudo chmod +x /usr/bin/consul
root@mq-hw-0003:~# sudo mkdir -p /var/lib/consul
root@mq-hw-0003:~# vi client.json
root@mq-hw-0003:~# sudo consul agent -config-file=client.json
=> Starting Consul agent:
    Version: '1.17.1'
    Build Date: '2023-12-12 19:55:27 +0000 UTC'
    Node ID: '6f3acbc8-60a0-87f5-6f45-26d7ce2acff0'
    Node name: 'mq-hw-0003'
    Datacenter: 'dc1' (Segment: '')
    Server: false (Bootstrap: false)
    Client Addr: '192.168.0.66' (HTTP: 8500, HTTPS: -1, gRPC: -1, DNS: 8600)
    Cluster Addr: '192.168.0.250' (LAN: 8301, WAN: 8302)
    Gossip Encryption: false
    Auto-Encrypt-TLS: false
    ACL Enabled: false
    ACL Default Policy: allow
    HTTP: TLS: Verify Incoming: false, Verify Outgoing: false, Min Version: TLSv1.2
    gRPC: TLS: Verify Incoming: false, Min Version: TLSv1.2
    Internal RPC: TLS: Verify Incoming: false, Verify Outgoing: false (Verify Hostname: false), Min Version: TLSv1.2
=> Log data will now stream in as it occurs:
2024-02-18T22:30:40.923+0800 [INFO] agent.client.serf.lw: serf: EventMemberJoin: mq-hw-0003 192.168.0.250
2024-02-18T22:30:40.923+0800 [INFO] agent.router: Initializing LAN area manager
2024-02-18T22:30:40.924+0800 [INFO] agent: Started DNS server: address=127.0.0.1:8600 network=udp
2024-02-18T22:30:40.924+0800 [INFO] agent: Started DNS server: address=127.0.0.1:8600 network=tcp
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=demo/v1/artist/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/computedroutes/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/catalog/v2beta1/healthstatus/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/tenancy/v1alpha1/namespaces/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/demo/v1/executive/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/proxyconfiguration/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/catalog/v2beta1/node/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/demo/v1/concept/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/computedexplicitdestinations/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/auth/v2beta1/workloadidentity/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/catalog/v2beta1/service/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/demo/v1/recordlabel/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/catalog/v2beta1/serviceendpoints/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/auth/v2beta1/trafficpermissions/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/internal/v1/tombstone/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/demo/v1/album/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/demo/v2/artist/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/destinations/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/httproute/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/tcproute/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/grpcroute/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/destinationpolicy/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/catalog/v2beta1/workload/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/catalog/v2beta1/failoverpolicy/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/auth/v2beta1/computedtrafficpermissions/
2024-02-18T22:30:40.924+0800 [INFO] agent.http: Registered resource endpoint: endpoint=/mesh/v2beta1/proxytemplate/
```

图 7-14 启动 consul 的结果

以上便是创建了一个 `consul` 的客户端，仿照此过程可添加其他节点进入 `consul` 集群，要注意修改配置文件中的 IP 地址为该虚拟机节点的 IP 地址。

#### 4. GUI 查看集群

服务器和客户端都启动成功后，如果是局域网，可在局域网内的任何一台机器上，通过 8500 端口访问到服务器运行的 `consul` GUI。如果使用的是华为云服务器，则可以

通过浏览器访问虚拟机公网 IP 地址来访问管理页面，代码如下：

```
http://121.36.19.156:8500/ui/
```

在运行 nomad 与 consul 后，nomad 云计算管理平台的节点、客户端均以微服务的形式注册到了 consul 集群中，受到 consul 的管理，consul 主页面如图 7-15 所示，可以查看集群状态或微服务状态等信息，在这些界面中，可以查看到之后查看 Consul 集群状态如图 7-16 所示，查看微服务状态如图 7-17 所示。注册的微服务。

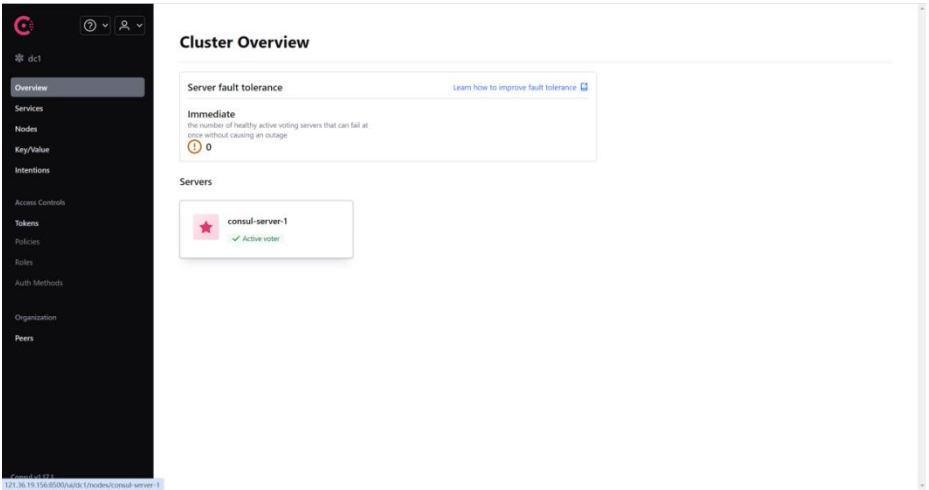


图 7-15 consul 主页面

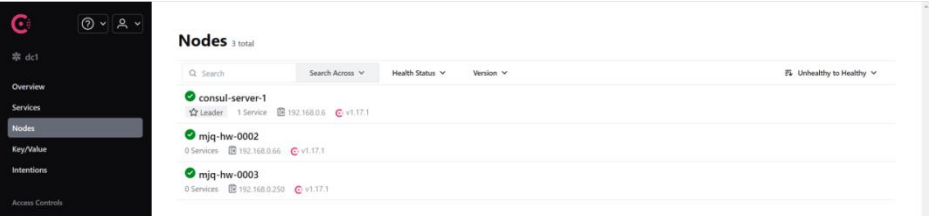


图 7-16 查看 consul 集群状态

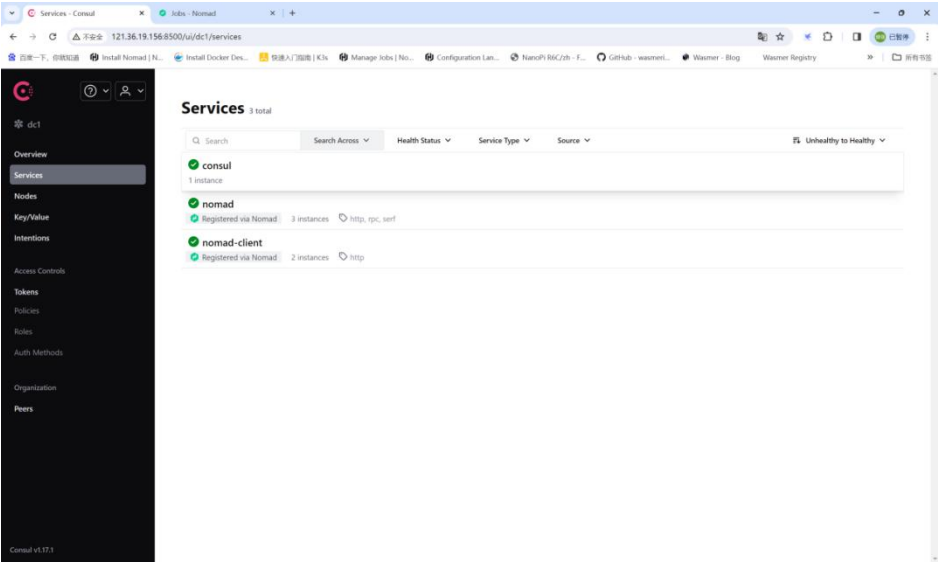


图 7-17 查看微服务状态

7.3 nomad 任务部署流程

系统动态管理调度流程如图 7-18 所示。由用户（如地面云）发出任务请求，将任务提交至主控机器节点后，由主控机器节点维护资源池，并分配任务至资源池中的某一具体机器进行执行，然后为其分配合适的资源。在用户看来，整个集群是统一的，用户不必考虑集群的操作系统和处理器架构差异，只需要向云计算集群提出任务请求即可。

其中，任务的执行方式（驱动方式）有多种，如虚拟机、docker 容器、java 等多种方式。本实验主要采用 docker 容器来运行任务，所以在部署 nomad-job 文件前，需要确保 docker 处于运行状态。

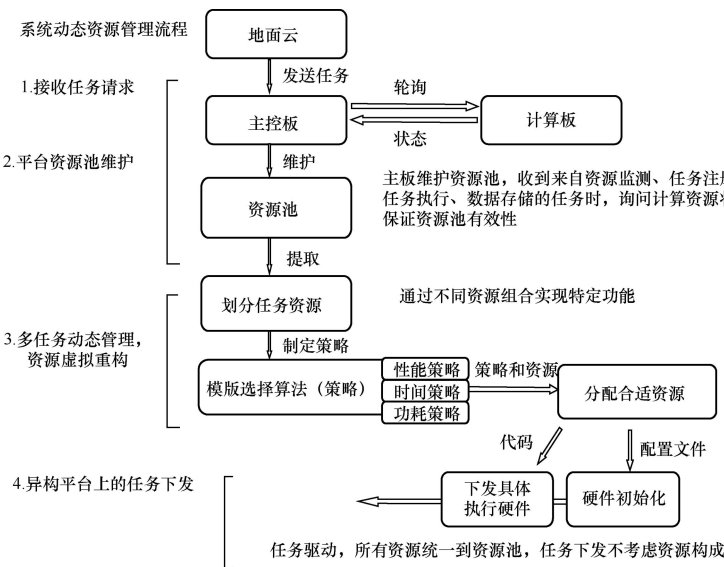


图 7-18 系统动态管理调度流程

nomad 任务调度简化流程如图 7-19 所示。在 nomad 平台下，可以将其简化为 3 个步骤：（1）用户提交规范的 job 文件（可能包含多个任务组及任务）至服务主控节点；（2）服务节点根据 job 文件的配置及要求，分配任务（以组为单位）至客户节点；（3）客户节点作为实际执行任务的硬件平台，完成任务并反馈。

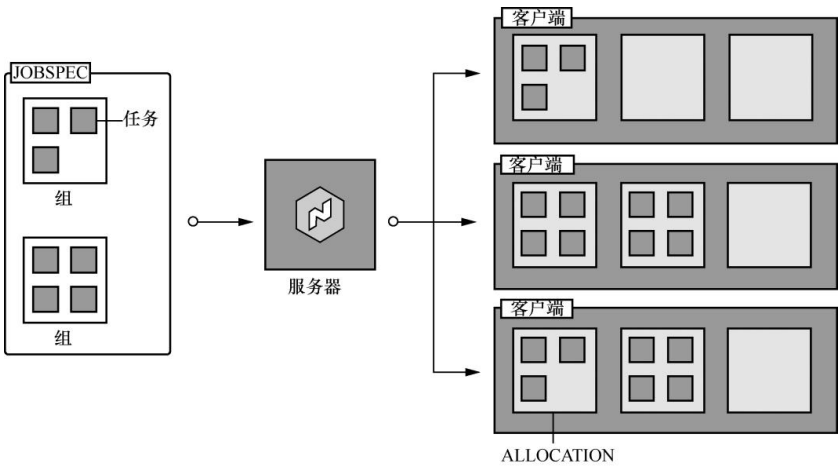


图 7-19 nomad 任务调度简化流程

## 1. 提交 job 文件

在 **nomad** 中，用户只需要一个简单的 **job** 文件就可以提交一个任务至云计算平台，一个 **job** 文件可以就指定任务执行的方式、内容（如计算程序或运行 **nginx** 网络服务器）、以及参数丰富的配置功能。提交 **job** 文件有命令行提交和使用 **nomad** 浏览器界面提交两种方法。

方法 1：命令行提交。

在能访问到集群的某一个用户端上，编写一个简单的 **job** 文件，命名并存储为 **docker\_hello.hcl**，内容如下：

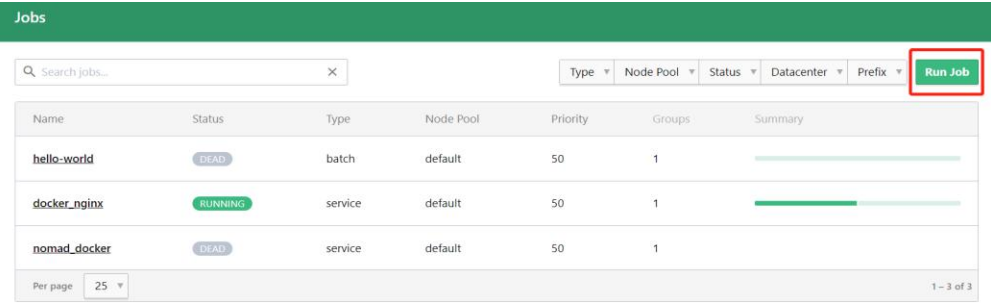
```
job "hello-world" {
  datacenters = ["dc1"]
  type = "batch"
  group "example" {
    count = 1
    task "hello-world-task" {
      driver = "docker"
      config {
        image = "hello-world"
      }
      resources {
        cpu    = 100
        memory = 128
      }
    }
  }
}
```

此 **job** 文件要求使用 **docker** 驱动方式（与之相对应的还有 **Java**、虚拟机、直接运行等方式）执行一个 **hello-world** 镜像，并且规定了该任务占用的资源大小，然后运行命令提交该任务即可，命令如下：

```
export NOMAD_ADDR = http://121.36.19.156:4646
nomad job run docker_hello.hcl
```

方法 2：使用 **nomad** 浏览器界面提交。

在浏览器 **GUI** 中，选择左侧的 **jobs** 栏，单击 **run job** 按钮，进入任务编写界面，在此可以直接提交符合规范的任务。浏览器 **job** 界面如图 7-20 所示，浏览器 **GUI** 提交任务界面如图 7-21 所示。



Name	Status	Type	Node Pool	Priority	Groups	Summary
hello-world	DEAD	batch	default	50	1	
docker_nginx	RUNNING	service	default	50	1	
nomad_docker	DEAD	service	default	50	1	

图 7-20 浏览器 job 界面

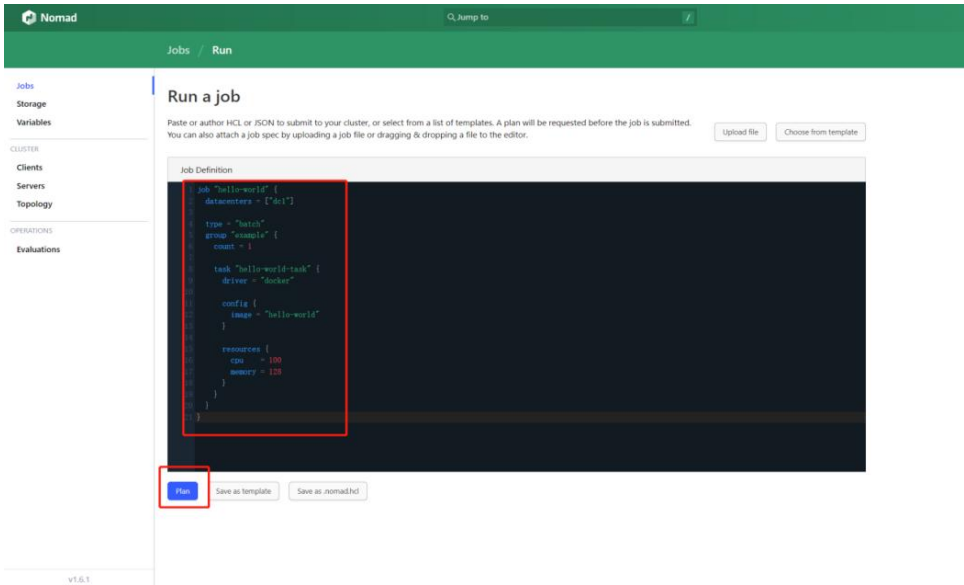


图 7-21 浏览器 GUI 提交任务界面

(2) 服务端分配任务

刚刚将任务提交到了集群的服务端，接下来服务端需要将该任务分配给集群中某一客户端机器来实际执行。在 **nomad** 管理页面上能够查看到服务器分配该任务的详细情况。

查看服务节点分配任务的方式有两种，其中一种是使用命令行提交 **job** 后，会在控制台输出任务分配情况，以及该任务执行的预计截止时间。超时则认为任务分配或执行失败。控制台查看分配如图 7-22 所示。

```
root@mj-q-hw-0001:~/nomad# export NOMAD_ADDR=http://121.36.19.156:4646
root@mj-q-hw-0001:~/nomad# nomad job run docker_hello.hcl
==> 2024-02-21T17:02:10+08:00: Monitoring evaluation "257b0816"
2024-02-21T17:02:10+08:00: Evaluation triggered by job "hello-world"
2024-02-21T17:02:10+08:00: Allocation "18f1f46a" created: node "06770610", group "example"
2024-02-21T17:02:11+08:00: Evaluation status changed: "pending" -> "complete"
==> 2024-02-21T17:02:11+08:00: Evaluation "257b0816" finished with status "complete"
root@mj-q-hw-0001:~/nomad#
```

图 7-22 控制台查看分配

相比而言，使用浏览器 GUI 查看更加方便，浏览器 CUI 查看分配如图 7-23 所示。分配成功的任务会显示 **complete**，失败或阻塞的任务也会显示相应的状态。

b3e6e419	<a href="#">docker_nginx2</a>	50	Sep 22 03:21:33 +0800	deployment-watcher	complete	False
33698336	<a href="#">docker_nginx2</a>	50	Sep 22 03:21:17 +0800	job-register	complete	False
312e9ea0	<a href="#">docker_nginx</a>	50	Sep 22 03:20:40 +0800	job-deregister	complete	False
098649ef	<a href="#">hello-world</a>	50	Sep 22 03:02:00 +0800	job-register	complete	False

图 7-23 浏览器 GUI 查看分配

对于不同类型的任务，服务端会执行不同的操作。对于 **batch** 类任务 **hello-world**，**nomad** 仅执行 **job-register**，而后在客户端执行一次后即结束。对于 **service** 类的任务 **docker\_nginx2**，**nomad** 在完成 **register** 操作后，会持续运行并监控该服务，而不是像 **batch** 型任务默认结束。

### (3) 客户节点执行任务

在服务端将任务分配至具体的某一客户节点后，可以通过任务界面来查询监控任务执行的具体情况，如任务状态、控制台输出结果、报错信息等，**hello-world** 的执行结果如图 7-24 所示。

如本例中，对于 **hello-world** 任务而言，客户节点在接收到此任务后，仅执行一次，然后便认为任务执行完毕，向 **Server** 节点返回 **complete** 状态。右侧控制台可以看到该程序的正确输出结果。

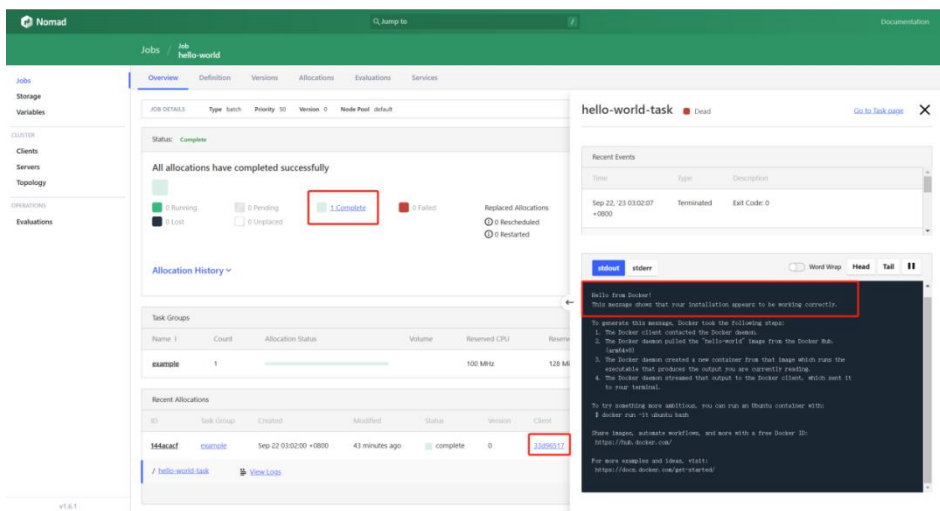


图 7-24 hello-world 的执行结果

## 7.4 运行 nginx 微服务

本例采用 **docker** 容器作为云计算任务的驱动，即应用程序以 **docker** 容器的方式运行。可以用如下命令查看 **docker** 是否正在作为系统服务运行。

```
sudo systemctl status docker
```

在网络环境不好的情况下，可能会出现 **nomad** 部署任务失败的情况。由于 **nginx** 的镜像文件较大，如果直接使用 **nomad** 发布任务，客户端很可能会因为下载时间太长，而被 **nomad** 误以为是执行失败，从而中断整个任务。所以可以先在客户端手动下载 **nginx**，

这样，当客户端收到 nomad 分配的任务后，就不用从 docker 网站上花时间下载了。

在客户端执行命令下载 nginx 镜像，命令如下：

```
sudo docker pull nginx
```

在客户端下载成功后，就可以在服务端编写 job 文件，提交任务至服务端了。job 文件内容如下：

```
job "nomad_docker_nginx" {

  datacenters = ["dc1"]
  group "nomad_docker_nginx_group" {
    count = 2
    network {
      port "docker_port" {
        static = 8765
        to = 80
      }
    }
  }
  service {
    name = "nginx"
    port = "docker_port"

    check {
      type      = "http"
      path      = "/"
      interval = "2s"
      timeout   = "2s"
    }
  }
  task "nomad_docker_nginx" {
    driver = "docker"

    config {
      image = "nginx:latest"
      ports = ["docker_port"]
    }
  }
}
```



将该job 文件提交到 nomad 服务端，控制台提交成功后的结果如图 7-25 所示。

```
export NOMAD_ADDR = http://192.168.0.6:4646
nomad job run docker_nginx.hcl
```

```
root@mq-hw-0001:~/nomad# export NOMAD_ADDR=http://192.168.0.6:4646
root@mq-hw-0001:~/nomad# nomad job run docker_nginx.hcl
==> 2024-02-19T11:52:43+08:00: Monitoring evaluation "e31d13bc"
2024-02-19T11:52:44+08:00: Evaluation triggered by job "nomad_docker_nginx_2"
2024-02-19T11:52:44+08:00: Evaluation within deployment: "27f984b5"
2024-02-19T11:52:44+08:00: Allocation "3cb515fe" created: node "06770610", group "nomad_docker_nginx_group"
2024-02-19T11:52:44+08:00: Allocation "607140a5" created: node "3b9e3234", group "nomad_docker_nginx_group"
2024-02-19T11:52:44+08:00: Evaluation status changed: "pending" -> "complete"
==> 2024-02-19T11:52:44+08:00: Evaluation "e31d13bc" finished with status "complete"
==> 2024-02-19T11:52:44+08:00: Monitoring deployment "27f984b5"
Deployment "27f984b5" successful

2024-02-19T11:52:59+08:00
ID = 27f984b5
Job ID = nomad_docker_nginx_2
Job Version = 4
Status = successful
Description = Deployment completed successfully

Deployed
Task Group Desired Placed Healthy Unhealthy Progress Deadline
nomad_docker_nginx_group 2 2 2 0 2024-02-19T12:02:58+08:00
root@mq-hw-0001:~/nomad#
```

图 7-25 控制台提交成功后的结果

nginx 服务运行成功如图 7-26 所示。

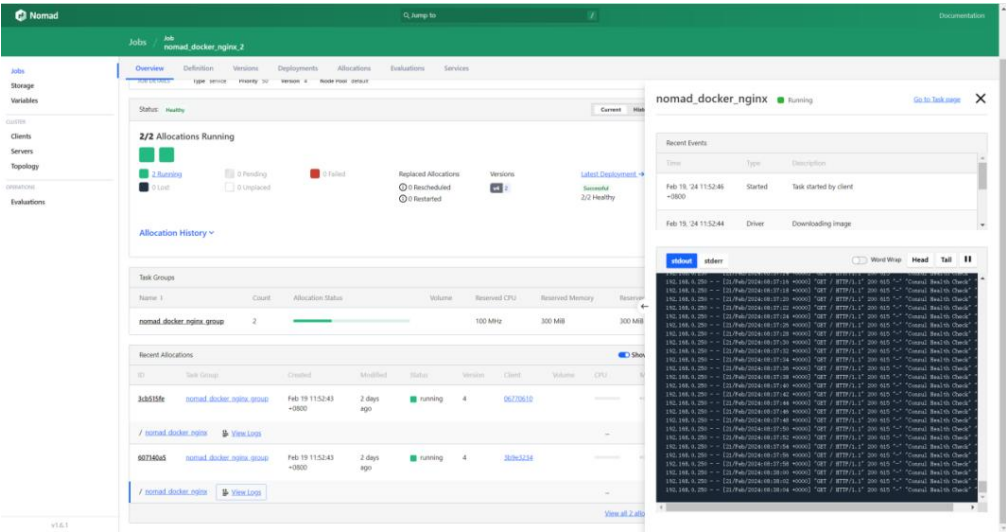


图 7-26 nginx 服务运行成功

集群状态如图 7-27 所示，service 型任务部署了多个 nginx 服务，服务将持续运行在 client1 及 client2 上，并占用一定的集群资源。

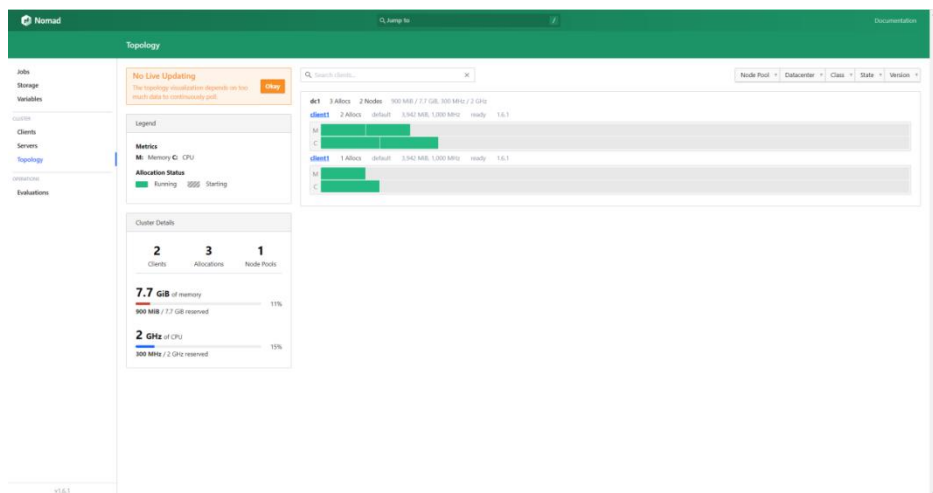


图 7-27 集群状态

此时，仅仅是云计算平台部署了 **nginx** 服务，并未将其纳入 **consul** 的微服务管理中。不过由于在任务部署时已经制定了将其注册到 **consul** 微服务，因此可以在 **consul** 的微服务管理页面上查看到部署的 **nginx** 服务。查看 **nginx** 微服务如图 7-28 所示，刚刚部署完毕运行的 **nginx** 任务已经注册到了 **consul** 中，作为微服务提供。

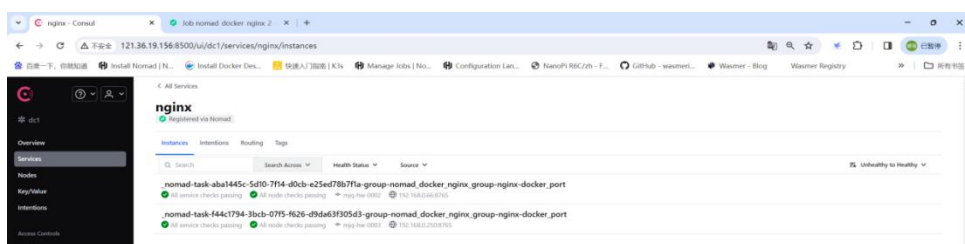


图 7-28 查看 nginx 微服务

此时，通过公网 IP 地址及 8765 端口（本例在 **job** 文件中设置为 8765）可以访问该客户端上的 **nginx** 页面，如图 7-29 所示。



图 7-29 访问客户端上的 nginx 页面

## 7.5 运行负载均衡微服务

负载均衡是一种技术策略，旨在通过分散工作负载（如网络流量、应用请求等）到多个服务器或资源上，以提高服务的可用性、可靠性和性能。在高流量的网络环境中，单一服务器可能难以应对大量的请求，这时候负载均衡就显得尤为重要。通过负载均衡，可以确保没有单一的服务点因为超载而成为系统瓶颈，从而避免服务中断和性能下降。

与之前的 **nginx** 微服务类似，可以运行一个 **nginx** 负载均衡服务器，并将其注册到

consul 微服务。这个负载均衡的微服务将通过调用之前在 consul 中注册的 nginx 微服务，为其提供负载均衡的功能。

此处是云原生的显著特点之一：微服务架构。每个微服务代表一个特定的业务能力。这种模块性确保服务可以独立发展，并允许团队自主工作。

为了部署该 nginx 负载均衡微服务，可以部署以下的 nomad-job，同样需要创建一个 load\_balance.hcl 文件，内容如下所示：

```
job "load_balance" {
  datacenters = ["dc1"]
  group "nginx" {
    count = 1
    network {
      port "http" {
        static = 8989
      }
    }
    service {
      name = "load_balance"
      port = "http"
    }
    task "nginx" {
      driver = "docker"
      config {
        image = "nginx"
        ports = ["http"]

        volumes = [
          "local:/etc/nginx/conf.d",
        ]
      }
      template {
        data = <<EOF
        upstream backend {
          {{ range service "nginx" }}
            server {{ .Address }}:{{ .Port }};
          {{ else }}server 127.0.0.1:65535; # force a 502
          {{ end }}
        }
      }
    }
  }
}
```

```

server {
    listen 8989;
    location / {
        proxy_pass http://backend;
    }
}
EOF

destination = "local/load-balancer.conf"
change_mode = "signal"
change_signal = "SIGHUP"
}
}
}
}

```

将该 job 文件提交到 nomad 服务端，命令如下：

```

export NOMAD_ADDR=http://192.168.0.6:4646
nomad job run load_balance.hcl

```

之后，可以看到 nomad 中该任务成功运行，负载均衡任务部署如图 7-30 所示。也可以在 consul 的微服务中看到它，负载均衡微服务如图 7-31 所示。

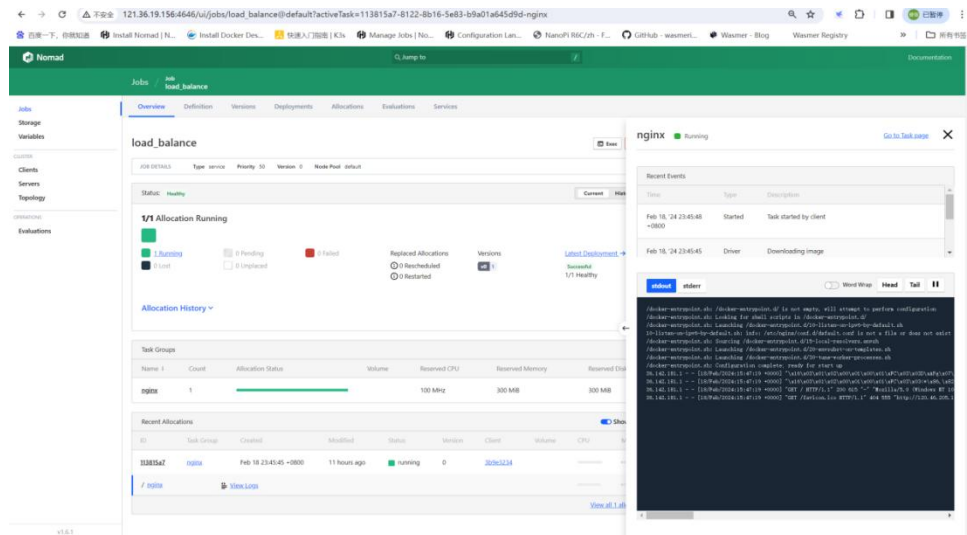


图 7-30 负载均衡任务部署

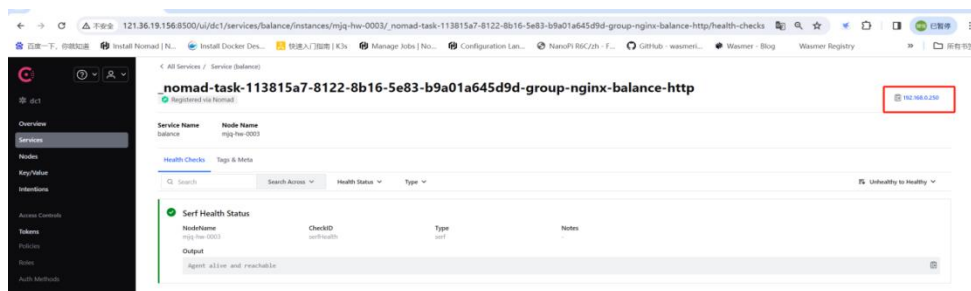


图 7-31 负载均衡微服务

通过负载均衡服务器访问 **nginx** 资源提供服务如图 7-32 所示。这与直接访问资源服务器的方式不同。直接访问资源服务器并未用到负载均衡功能，但此处访问是通过负载均衡服务器（并未提供实际的资源）提供的入口，进而访问某一台实际提供资源的服务器。可以仔细查看访问的网址，图 7-32 与 7.9.4 节中的访问网址并不相同。



图 7-32 通过负载均衡服务器访问 **nginx** 资源提供服务

负载均衡微服务为部署的两个 **nginx** 服务器提供了统一的访问入口。它会判断后端接入的两个 **nginx** 微服务的工作负载，偏向选择负载较低的微服务作为目的服务器。

在本节的实践中，通过各个独立功能的微服务进行组合，实现了部署简单、功能多样的 **nginx** 服务器，并提供了负载均衡的实用功能。

传统的部署方式需要在不同节点上手动安装 **nginx** 服务，并且进行服务器配置。对于作为访问端点的 **nginx** 服务器而言，**nginx** 配置一旦完成，如果修改配置则需要重启整个 **nginx** 服务器。如果对 **nginx** 服务器做出修改，则需要重新指定地址。所以，一旦某个服务器发生了变化，则需要各个部分均做出相应的修改、重启。

在本例的微服务架构中可以看到，通过将 **nginx** 服务器和负载均衡服务器作为微服务注册到 **consul**，可以实现各模块之间的解耦合。对某一模块的修改更新，并不会影响到其他模块与之对接的接口。并且由于采用了 **docker** 容器的部署方式，**nginx** 服务器的版本更新也只需要修改 **job** 文件即可。在这样的情况下，可以向当前服务器添加更多的功能模块，但不会使整个系统变的过于臃肿复杂。