基于 Nginx 的 API 部署方案

本节核心内容

- 介绍 Nginx
- 介绍如何安装 Nginx
- 介绍如何配置 Nginx

> 本小节源码下载路径：demo14 (https://github.com/lexkong/apiserver_demos/tree/mas
>
> 可先下载源码到本地，结合源码理解后续内容，边学边练。
>
> 本小节的代码是基于 demo13 (https://github.com/lexkong/apiserver_demos/tree/mas 来开发的。

Nginx 介绍

Nginx 是一个自由、开源、高性能及轻量级的 HTTP 服务器和反向代理服务器，它有很多功能，主要功能为：

1. 正向代理
2. 反向代理
3. 负载均衡
4. HTTP 服务器（包含动静分离）

本小册使用了 Nginx 反向代理和负载均衡的功能。

> Nginx 的更详细介绍可以参考 [nginx简易教程 (https://www.cnblogs.com/jingmoxukong/p/5945200.h](https://www.cnblogs.com/jingmoxukong/p/5945200.h)

Nginx 反向代理功能

Nginx 最常用的功能之一是作为一个反向代理服务器。反向代理（Reverse Proxy）是指以代理服务器来接收 Internet 上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给 Internet 上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器（摘自百度百科）。

为什么需要反向代理呢？在实际的生产环境中，服务部署的网络（内网）跟外部网络（外网）通常是不通的，需要通过一台既能够访问内网又能够访问外网的服务器来做中转，这种服务器就是反向代理服务器。Nginx 作为反向代理服务器，简单的配置如下：

```
    server {
        listen       80;
        server_name  apiserver.com;
        client_max_body_size 1024M;

        location / {
            proxy_set_header Host $http_host;
            proxy_set_header X-Forwarded-Host
$http_host;
            proxy_set_header X-Real-IP
$remote_addr;
            proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
            proxy_pass  http://127.0.0.1:8080/;
            client_max_body_size 100m;
        }
    }
```

Nginx 在做反向代理服务器时，能够根据不同的配置规则转发到后端不同的服务器上。

Nginx 负载均衡功能

Nginx 另一个常用的功能是负载均衡，所谓的负载均衡就是指当 Nginx 收到一个 HTTP 请求后，会根据负载策略将请求转发到不同的后端服务器上。比如，apiserver 部署在两台服务器 A 和 B 上，当请求到达 Nginx 后，Nginx 会根据 A 和 B 服务器上的负载情况，将请求转发到负载较小的那台服务器上。这里要求 apiserver 是无状态的服务。

安装和启动 Nginx（需要切换到 root 用户）

1. 安装 Nginx（CentOS 7.x 安装流程）

```
$ yum -y install nginx
```

2. 确认 Nginx 安装成功

```
$ nginx -v
```

3. 启动 Nginx

```
$ systemctl start nginx
```

4. 设置开机启动

```
$ systemctl enable nginx
```

5. 查看 Nginx 启动状态

```
$ systemctl status nginx
```

Nginx 常用命令

Nginx 常用命令如下（执行 which nginx 可以找到 Nginx 命令所在的路径）：

```
nginx -s stop        快速关闭 Nginx，可能不保存相关信息，并迅速终止 Web 服务
nginx -s quit        平稳关闭 Nginx，保存相关信息，有安排的结束 Web 服务
nginx -s reload      因改变了 Nginx 相关配置，需要重新加载配置而重载
nginx -s reopen      重新打开日志文件
nginx -c filename    为 Nginx 指定一个配置文件，来代替默认的
nginx -t             不运行，而仅仅测试配置文件。Nginx 将检查配置文件的语法的正确性，并尝试打开配置文件中所引用到的文件
nginx -v             显示 Nginx 的版本
nginx -V             显示 Nginx 的版本、编译器版本和配置参数
```

> Nginx 默认监听 80 端口，启动 Nginx 前要确保 80 端口没有被占用。当然你也可以通过修改 Nginx 配置文件 /etc/nginx/nginx.conf 改 Nginx 监听端口。

配置 Nginx 作为反向代理

假定要访问的 API 服务器域名为 apiserver.com，在 /etc/nginx/nginx.conf 配置 API 服务器的 server 入口：

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;

    server {
        listen      80;
        server_name  apiserver.com;
        client_max_body_size 1024M;

        location / {
            proxy_set_header Host $http_host;
            proxy_set_header X-Forwarded-Host $http_host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass  http://127.0.0.1:8080/;
            client_max_body_size 100m;
        }
    }
}
```

完成 nginx.conf 内容如下：

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
```

```
    log_format  main  '$remote_addr -
$remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent
"$http_referer" '
                      '"$http_user_agent"
"$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;

    server {
        listen      80;
        server_name  apiserver.com;
        client_max_body_size 1024M;

        location / {
            proxy_set_header Host $http_host;
            proxy_set_header X-Forwarded-Host
$http_host;
            proxy_set_header X-Real-IP
$remote_addr;
            proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
            proxy_pass  http://127.0.0.1:8080/;
            client_max_body_size 5m;
```

```
        }
    }
}
```

配置说明

- 由于 Nginx 默认允许客户端请求的最大单文件字节数为 1MB，实际生产环境中可能太小，所以这里将此限制改为 5MB（client_max_body_size 5m）
- server_name：说明使用哪个域名来访问
- proxy_pass：反向代理的路径（这里是本机的 API 服务，所以IP为 127.0.0.1。端口要和 API 服务端口一致：8080）

> 如果需要上传图片之类的，可能需要设置成更大的值，比如 50m。
>
> 因为 Nginx 配置选项比较多，跟实际需求和环境有关，所以这里的配置是基础的、未经优化的配置，在实际生产环境中，需要读者再做调节。

测试

1. 配置完 Nginx 后重启 Nginx

```
$ systemctl restart nginx
```

2. 在编译完 apiserver 后，启动 API 服务器

```
$ ./apiserver
```

3. 在 /etc/hosts 中添加一行：127.0.0.1 apiserver.com
4. 发送 HTTP 请求

```
$ curl -XGET -H "Content-Type: application/json"
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1
MjgwMTY5MjIsImlkIjowLCJuYmYiOjE1MjgwMTY5MjIsInVzZ
XJuYW1lIjoiYWRtaW4ifQ.LjxrK9DuAwAzUD8-
9v43NzWBN7HXsSLfebw92DKd1JQ"
http://apiserver.com/v1/user

{
  "code": 0,
  "message": "OK",
  "data": {
    "totalCount": 1,
    "userList": [
      {
        "id": 0,
        "username": "admin",
        "sayHello": "Hello Jypl3DSig",
        "password":
"$2a$10$veGcArz47VGj7l9xN7g2iuT9TF21jLI1YGXarGzvA
RNdnt4inC9PG",
        "createdAt": "2018-05-28 00:25:33",
        "updatedAt": "2018-05-28 00:25:33"
      }
    ]
  }
}
```

可以看到成功通过代理访问后端的 API 服务。

请求流程说明

在用 curl 请求 http://apiserver.com/v1/user 后，后端的请求流程实际上是这样的：

1. 因为在 /etc/hosts 中配置了 127.0.0.1 apiserver.com，所以请求 http://apiserver.com/v1/use 实际上是请求本机的 Nginx 端口（127.0.0.1:80）
2. Nginx 在收到请求后，解析到请求域名为 apiserver.com，根据请求域名去匹配 Nginx 的 server 配置，匹配到 server_name apiserver.com 配置
3. 匹配到 server 后，把请求转发到该 server 的 proxy_pass 路径
4. 等待 API 服务器返回结果，并返回客户端

配置 Nginx 作为负载均衡

负载均衡的演示需要多个后端服务，为此我们在同一个服务器上启动多个 apiserver，配置不同的端口（8080、8082），并采用 Nginx 默认的轮询转发策略（轮询：每个请求按时间顺序逐一分配到不同的后端服务器）。

在 /etc/nginx/nginx.conf 中添加 upstream 配置：

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
    upstream apiserver.com {
        server 127.0.0.1:8080;
        server 127.0.0.1:8082;
    }

    server {
        listen      80;
        server_name  apiserver.com;
        client_max_body_size 1024M;

        location / {
            proxy_set_header Host $http_host;
            proxy_set_header X-Forwarded-Host $http_host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass  http://apiserver.com/;
            client_max_body_size 100m;
        }
    }
}
```

配置说明

- 因为有多个后端，所以需要将之前固定的后端 proxy_pass http://127.0.0.1:8080/ 换成具有多个后端的 apiserver.com（通过 upstream）
- upstream 配置中配置多个后端（ip:port）

```
    upstream apiserver.com {
        server 127.0.0.1:8080;
        server 127.0.0.1:8082;
    }
```

测试

1. 配置完 Nginx 后重启 Nginx

```
$ systemctl restart nginx
```

2. 这里需要构建并发请求，编写测试脚本 test.sh，内容为：

```
#!/bin/bash


for n in $(seq 1 1 10)
do
    nohup curl -XGET -H "Content-Type:
application/json" -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1
MjgwMTY5MjIsImlkIjowLCJuYmYiOjE1MjgwMTY5MjIsInVzZ
XJuYW1lIjoiYWRtaW4ifQ.LjxrK9DuAwAzUD8-
9v43NzWBN7HXsSLfebw92DKd1JQ"
http://apiserver.com/v1/user &>/dev/null
done
```

3. 为了展示哪个 API 被调用，需要在查询用户列表的入口函数（handler/user/list.go文件中的 List() 函数）中添加日志打印信息：

```go
package user

import (
    . "apiserver/handler"
    "apiserver/pkg/errno"
    "apiserver/service"

    "github.com/gin-gonic/gin"
    "github.com/lexkong/log"
)

// List list the users in the database.
func List(c *gin.Context) {
    log.Info("List function called.")
    var r ListRequest
    if err := c.Bind(&r); err != nil {
        SendResponse(c, errno.ErrBind, nil)
        return
    }

    infos, count, err := service.ListUser(r.Username, r.Offset, r.Limit)
    if err != nil {
        SendResponse(c, err, nil)
        return
    }

    SendResponse(c, nil, ListResponse{
        TotalCount: count,
        UserList:   infos,
    })
}
```

4. 在相同服务器上启动两个不同的 HTTP 端口：8080 和 8082
5. 执行 test.sh 脚本

```
$ ./test.sh
```

观察 API 日志，可以看到请求被均衡地转发到后端的两个服务：

apiserver1（8080 端口）：

```
[api@centos apiserver]$ ./apiserver
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] POST   /login                    --> apiserver/handler/user.Login (5 handlers)
[GIN-debug] POST   /v1/user                  --> apiserver/handler/user.Create (6 handlers)
[GIN-debug] DELETE /v1/user/:id              --> apiserver/handler/user.Delete (6 handlers)
[GIN-debug] PUT    /v1/user/:id              --> apiserver/handler/user.Update (6 handlers)
[GIN-debug] GET    /v1/user                  --> apiserver/handler/user.List (6 handlers)
[GIN-debug] GET    /v1/user/:username        --> apiserver/handler/user.Get (6 handlers)
[GIN-debug] GET    /sd/health                --> apiserver/handler/sd.HealthCheck (5 handlers)
[GIN-debug] GET    /sd/disk                  --> apiserver/handler/sd.DiskCheck (5 handlers)
[GIN-debug] GET    /sd/cpu                   --> apiserver/handler/sd.CPUCheck (5 handlers)
[GIN-debug] GET    /sd/ram                   --> apiserver/handler/sd.RAMCheck (5 handlers)
2018-06-05 14:03:01.395 INFO apiserver/main.go:85 Start to listening the incoming requests on http address: :8080
2018-06-05 14:03:01.396 INFO apiserver/main.go:80 Start to listening the incoming requests on https address: :8081
2018-06-05 14:03:01.396 INFO apiserver/main.go:81 listen tcp :8081: bind: address already in use
2018-06-05 14:03:01.396 INFO apiserver/main.go:72 The router has been deployed successfully.
2018-06-05 14:04:49.835 INFO user/list.go:14 List function called.
2018-06-05 14:04:49.847 INFO user/list.go:14 List function called.
2018-06-05 14:04:49.858 INFO user/list.go:14 List function called.
2018-06-05 14:04:49.868 INFO user/list.go:14 List function called.
2018-06-05 14:04:49.881 INFO user/list.go:14 List function called.
```

收到5个请求

apiserver2（8082 端口）：

```
[api@centos apiserver]$ ./apiserver -c config.yaml
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
 - using env:   export GIN_MODE=release
 - using code:  gin.SetMode(gin.ReleaseMode)

[GIN-debug] POST   /login                    --> apiserver/handler/user.Login (5 handlers)
[GIN-debug] POST   /v1/user                  --> apiserver/handler/user.Create (6 handlers)
[GIN-debug] DELETE /v1/user/:id              --> apiserver/handler/user.Delete (6 handlers)
[GIN-debug] PUT    /v1/user/:id              --> apiserver/handler/user.Update (6 handlers)
[GIN-debug] GET    /v1/user                  --> apiserver/handler/user.List (6 handlers)
[GIN-debug] GET    /v1/user/:username        --> apiserver/handler/user.Get (6 handlers)
[GIN-debug] GET    /sd/health                --> apiserver/handler/sd.HealthCheck (5 handlers)
[GIN-debug] GET    /sd/disk                  --> apiserver/handler/sd.DiskCheck (5 handlers)
[GIN-debug] GET    /sd/cpu                   --> apiserver/handler/sd.CPUCheck (5 handlers)
[GIN-debug] GET    /sd/ram                   --> apiserver/handler/sd.RAMCheck (5 handlers)
2018-06-05 14:03:04.018 INFO apiserver/main.go:85 Start to listening the incoming requests on http address: :8082
2018-06-05 14:03:04.018 INFO apiserver/main.go:80 Start to listening the incoming requests on https address: :8081
2018-06-05 14:03:04.020 INFO apiserver/main.go:72 The router has been deployed successfully.
2018-06-05 14:04:49.841 INFO user/list.go:14 List function called.
2018-06-05 14:04:49.852 INFO user/list.go:14 List function called.
2018-06-05 14:04:49.863 INFO user/list.go:14 List function called.
2018-06-05 14:04:49.875 INFO user/list.go:14 List function called.
2018-06-05 14:04:49.886 INFO user/list.go:14 List function called.
```

收到5个请求

小结

在生产环境中，API 服务器所在的网络通常不能直接通过外网访问，需要通过可从外网访问的 Nginx 服务器，将请求转发到内网的 API 服务器。并且随着业务规模越来越大，请求量也会越来越大，这时候需要将 API 横向扩容，也需要 Nginx。所以在实际的 API 服务部署中 Nginx 经常能派上用场。通过本小节的学习，读者可以了解到如何在实际生产环境中部署 API 服务。