[**NginxConfiguration**](#NginxConfiguration)

**[How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 18.04](#_How_To_Serve)**

[**How To Set Up Nginx Server Blocks (Virtual Hosts) on Ubuntu 16.04**](#_How_To_Set)

[**nginxgunicorn部署supervisor部署**](#nginxgunicorn部署supervisor部署)**（监视服务的启动和重启）**

[**CRM项目部署实际步骤**](#CRM项目部署实际步骤)

**[Nginx配置文件 nginx.conf 和default.conf 讲解](https://www.cnblogs.com/panchanggui/p/12067953.html)**

Posted on 2019-12-19 15:15  [panchanggui](https://www.cnblogs.com/panchanggui/)  阅读(6183)  评论(0)  [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=12067953)  [收藏](javascript:void(0))

**nginx.conf**

/etc/nginx/nginx.conf

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195 | ######Nginx配置文件nginx.conf中文详解#####    #定义Nginx运行的用户和用户组  user www www;    #nginx进程数，建议设置为等于CPU总核心数。  worker\_processes 8;    #全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]  error\_log /usr/local/nginx/logs/error.log info;    #进程pid文件  pid /usr/local/nginx/logs/nginx.pid;    #指定进程可以打开的最大描述符：数目  #工作模式与连接数上限  #这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。  #现在在linux 2.6内核下开启文件打开数为65535，worker\_rlimit\_nofile就相应应该填写65535。  #这是因为nginx调度时分配请求到进程并不是那么的均衡，所以假如填写10240，总并发量达到3-4万时就有进程可能超过10240了，这时会返回502错误。  worker\_rlimit\_nofile 65535;    events  {      #参考事件模型，use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型      #是Linux 2.6以上版本内核中的高性能网络I/O模型，linux建议epoll，如果跑在FreeBSD上面，就用kqueue模型。      #补充说明：      #与apache相类，nginx针对不同的操作系统，有不同的事件模型      #A）标准事件模型      #Select、poll属于标准事件模型，如果当前系统不存在更有效的方法，nginx会选择select或poll      #B）高效事件模型      #Kqueue：使用于FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 和 MacOS X.使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。      #Epoll：使用于Linux内核2.6版本及以后的系统。      #/dev/poll：使用于Solaris 7 11/99+，HP/UX 11.22+ (eventport)，IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+。      #Eventport：使用于Solaris 10。 为了防止出现内核崩溃的问题， 有必要安装安全补丁。      use epoll;        #单个进程最大连接数（最大连接数=连接数\*进程数）      #根据硬件调整，和前面工作进程配合起来用，尽量大，但是别把cpu跑到100%就行。每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为。      worker\_connections 65535;        #keepalive超时时间。      keepalive\_timeout 60;        #客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求头的大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。      #分页大小可以用命令getconf PAGESIZE 取得。      #[root@web001 ~]# getconf PAGESIZE      #4096      #但也有client\_header\_buffer\_size超过4k的情况，但是client\_header\_buffer\_size该值必须设置为“系统分页大小”的整倍数。      client\_header\_buffer\_size 4k;        #这个将为打开文件指定缓存，默认是没有启用的，max指定缓存数量，建议和打开文件数一致，inactive是指经过多长时间文件没被请求后删除缓存。      open\_file\_cache max=65535 inactive=60s;        #这个是指多长时间检查一次缓存的有效信息。      #语法:open\_file\_cache\_valid time 默认值:open\_file\_cache\_valid 60 使用字段:http, server, location 这个指令指定了何时需要检查open\_file\_cache中缓存项目的有效信息.      open\_file\_cache\_valid 80s;        #open\_file\_cache指令中的inactive参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive时间内一次没被使用，它将被移除。      #语法:open\_file\_cache\_min\_uses number 默认值:open\_file\_cache\_min\_uses 1 使用字段:http, server, location  这个指令指定了在open\_file\_cache指令无效的参数中一定的时间范围内可以使用的最小文件数,如果使用更大的值,文件描述符在cache中总是打开状态.      open\_file\_cache\_min\_uses 1;        #语法:open\_file\_cache\_errors on | off 默认值:open\_file\_cache\_errors off 使用字段:http, server, location 这个指令指定是否在搜索一个文件是记录cache错误.      open\_file\_cache\_errors on;  }        #设定http服务器，利用它的反向代理功能提供负载均衡支持  http  {      #文件扩展名与文件类型映射表      include /etc/nginx/mime.types;        #默认文件类型      default\_type application/octet-stream;        #默认编码      #charset utf-8;        #服务器名字的hash表大小      #保存服务器名字的hash表是由指令server\_names\_hash\_max\_size 和server\_names\_hash\_bucket\_size所控制的。参数hash bucket size总是等于hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次是在存储单元中查找键 值。因此，如果Nginx给出需要增大hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小.      server\_names\_hash\_bucket\_size 128;        #客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE取得。      client\_header\_buffer\_size 32k;        #客户请求头缓冲大小。nginx默认会用client\_header\_buffer\_size这个buffer来读取header值，如果header过大，它会使用large\_client\_header\_buffers来读取。      large\_client\_header\_buffers 4 64k;        #设定通过nginx上传文件的大小      client\_max\_body\_size 8m;        #开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成off。      #sendfile指令指定 nginx 是否调用sendfile 函数（zero copy 方式）来输出文件，对于普通应用，必须设为on。如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络IO处理速度，降低系统uptime。      sendfile on;        #开启目录列表访问，合适下载服务器，默认关闭。      autoindex on;        #此选项允许或禁止使用socke的TCP\_CORK的选项，此选项仅在使用sendfile的时候使用      tcp\_nopush on;        tcp\_nodelay on;        #长连接超时时间，单位是秒      keepalive\_timeout 120;        #FastCGI相关参数是为了改善网站的性能：减少资源占用，提高访问速度。下面参数看字面意思都能理解。      fastcgi\_connect\_timeout 300;      fastcgi\_send\_timeout 300;      fastcgi\_read\_timeout 300;      fastcgi\_buffer\_size 64k;      fastcgi\_buffers 4 64k;      fastcgi\_busy\_buffers\_size 128k;      fastcgi\_temp\_file\_write\_size 128k;        #gzip模块设置      gzip on; #开启gzip压缩输出      gzip\_min\_length 1k;    #最小压缩文件大小      gzip\_buffers 4 16k;    #压缩缓冲区      gzip\_http\_version 1.0;    #压缩版本（默认1.1，前端如果是squid2.5请使用1.0）      gzip\_comp\_level 2;    #压缩等级      gzip\_types text/plain application/x-javascript text/css application/xml;    #压缩类型，默认就已经包含textml，所以下面就不用再写了，写上去也不会有问题，但是会有一个warn。      gzip\_vary on;        #开启限制IP连接数的时候需要使用      #limit\_zone crawler $binary\_remote\_addr 10m;            #负载均衡配置      upstream piao.jd.com {            #upstream的负载均衡，weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。          server 192.168.80.121:80 weight=3;          server 192.168.80.122:80 weight=2;          server 192.168.80.123:80 weight=3;            #nginx的upstream目前支持4种方式的分配          #1、轮询（默认）          #每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。          #2、weight          #指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。          #例如：          #upstream bakend {          #    server 192.168.0.14 weight=10;          #    server 192.168.0.15 weight=10;          #}          #2、ip\_hash          #每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。          #例如：          #upstream bakend {          #    ip\_hash;          #    server 192.168.0.14:88;          #    server 192.168.0.15:80;          #}          #3、fair（第三方）          #按后端服务器的响应时间来分配请求，响应时间短的优先分配。          #upstream backend {          #    server server1;          #    server server2;          #    fair;          #}          #4、url\_hash（第三方）          #按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。          #例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash\_method是使用的hash算法          #upstream backend {          #    server squid1:3128;          #    server squid2:3128;          #    hash $request\_uri;          #    hash\_method crc32;          #}            #tips:          #upstream bakend{#定义负载均衡设备的Ip及设备状态}{          #    ip\_hash;          #    server 127.0.0.1:9090 down;          #    server 127.0.0.1:8080 weight=2;          #    server 127.0.0.1:6060;          #    server 127.0.0.1:7070 backup;          #}          #在需要使用负载均衡的server中增加 proxy\_pass http://bakend/;            #每个设备的状态设置为:          #1.down表示单前的server暂时不参与负载          #2.weight为weight越大，负载的权重就越大。          #3.max\_fails：允许请求失败的次数默认为1.当超过最大次数时，返回proxy\_next\_upstream模块定义的错误          #4.fail\_timeout:max\_fails次失败后，暂停的时间。          #5.backup： 其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。            #nginx支持同时设置多组的负载均衡，用来给不用的server来使用。          #client\_body\_in\_file\_only设置为On 可以讲client post过来的数据记录到文件中用来做debug          #client\_body\_temp\_path设置记录文件的目录 可以设置最多3层目录          #location对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡      } |

**default.conf**

/etc/nginx/conf.d/default.conf

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131 | server      {          #监听端口          listen 80;            #域名可以有多个，用空格隔开          server\_name www.jd.com jd.com;          index index.html index.htm index.php;          root /data/www/jd;            #对\*\*\*\*\*\*进行负载均衡          location ~ .\*.(php|php5)?$          {              fastcgi\_pass 127.0.0.1:9000;              fastcgi\_index index.php;              include fastcgi.conf;          }            #图片缓存时间设置          location ~ .\*.(gif|jpg|jpeg|png|bmp|swf)$          {              expires 10d;          }            #JS和CSS缓存时间设置          location ~ .\*.(js|css)?$          {              expires 1h;          }            #日志格式设定          #$remote\_addr与$http\_x\_forwarded\_for用以记录客户端的ip地址；          #$remote\_user：用来记录客户端用户名称；          #$time\_local： 用来记录访问时间与时区；          #$request： 用来记录请求的url与http协议；          #$status： 用来记录请求状态；成功是200，          #$body\_bytes\_sent ：记录发送给客户端文件主体内容大小；          #$http\_referer：用来记录从那个页面链接访问过来的；          #$http\_user\_agent：记录客户浏览器的相关信息；          #通常web服务器放在反向代理的后面，这样就不能获取到客户的IP地址了，通过$remote\_add拿到的IP地址是反向代理服务器的iP地址。反向代理服务器在转发请求的http头信息中，可以增加x\_forwarded\_for信息，用以记录原有客户端的IP地址和原来客户端的请求的服务器地址。          log\_format access '$remote\_addr - $remote\_user [$time\_local] "$request" '          '$status $body\_bytes\_sent "$http\_referer" '          '"$http\_user\_agent" $http\_x\_forwarded\_for';            #定义本虚拟主机的访问日志          access\_log  /usr/local/nginx/logs/host.access.log  main;          access\_log  /usr/local/nginx/logs/host.access.404.log  log404;            #对 "/" 启用反向代理          location / {              proxy\_pass http://127.0.0.1:88;              proxy\_redirect off;              proxy\_set\_header X-Real-IP $remote\_addr;                #后端的Web服务器可以通过X-Forwarded-For获取用户真实IP              proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;                #以下是一些反向代理的配置，可选。              proxy\_set\_header Host $host;                #允许客户端请求的最大单文件字节数              client\_max\_body\_size 10m;                #缓冲区代理缓冲用户端请求的最大字节数，              #如果把它设置为比较大的数值，例如256k，那么，无论使用firefox还是IE浏览器，来提交任意小于256k的图片，都很正常。如果注释该指令，使用默认的client\_body\_buffer\_size设置，也就是操作系统页面大小的两倍，8k或者16k，问题就出现了。              #无论使用firefox4.0还是IE8.0，提交一个比较大，200k左右的图片，都返回500 Internal Server Error错误              client\_body\_buffer\_size 128k;                #表示使nginx阻止HTTP应答代码为400或者更高的应答。              proxy\_intercept\_errors on;                #后端服务器连接的超时时间\_发起握手等候响应超时时间              #nginx跟后端服务器连接超时时间(代理连接超时)              proxy\_connect\_timeout 90;                #后端服务器数据回传时间(代理发送超时)              #后端服务器数据回传时间\_就是在规定时间之内后端服务器必须传完所有的数据              proxy\_send\_timeout 90;                #连接成功后，后端服务器响应时间(代理接收超时)              #连接成功后\_等候后端服务器响应时间\_其实已经进入后端的排队之中等候处理（也可以说是后端服务器处理请求的时间）              proxy\_read\_timeout 90;                #设置代理服务器（nginx）保存用户头信息的缓冲区大小              #设置从被代理服务器读取的第一部分应答的缓冲区大小，通常情况下这部分应答中包含一个小的应答头，默认情况下这个值的大小为指令proxy\_buffers中指定的一个缓冲区的大小，不过可以将其设置为更小              proxy\_buffer\_size 4k;                #proxy\_buffers缓冲区，网页平均在32k以下的设置              #设置用于读取应答（来自被代理服务器）的缓冲区数目和大小，默认情况也为分页大小，根据操作系统的不同可能是4k或者8k              proxy\_buffers 4 32k;                #高负荷下缓冲大小（proxy\_buffers\*2）              proxy\_busy\_buffers\_size 64k;                #设置在写入proxy\_temp\_path时数据的大小，预防一个工作进程在传递文件时阻塞太长              #设定缓存文件夹大小，大于这个值，将从upstream服务器传              proxy\_temp\_file\_write\_size 64k;          }              #设定查看Nginx状态的地址          location /NginxStatus {              stub\_status on;              access\_log on;              auth\_basic "NginxStatus";              auth\_basic\_user\_file confpasswd;              #htpasswd文件的内容可以用apache提供的htpasswd工具来产生。          }            #本地动静分离反向代理配置          #所有jsp的页面均交由tomcat或resin处理          location ~ .(jsp|jspx|do)?$ {              proxy\_set\_header Host $host;              proxy\_set\_header X-Real-IP $remote\_addr;              proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;              proxy\_pass http://127.0.0.1:8080;          }            #所有静态文件由nginx直接读取不经过tomcat或resin          location ~ .\*.(htm|html|gif|jpg|jpeg|png|bmp|swf|ioc|rar|zip|txt|flv|mid|doc|ppt|          pdf|xls|mp3|wma)$          {              expires 15d;          }            location ~ .\*.(js|css)?$          {              expires 1h;          }      }  } |

######Nginx配置文件nginx.conf中文详解#####

#定义Nginx运行的用户和用户组

user www www;

#nginx进程数，建议设置为等于CPU总核心数。

worker\_processes 8;

#全局错误日志定义类型，[ debug | info | notice | warn | error | crit ]

error\_log /usr/local/nginx/logs/error.log info;

#进程pid文件

pid /usr/local/nginx/logs/nginx.pid;

#指定进程可以打开的最大描述符：数目

#工作模式与连接数上限

#这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。

#现在在linux 2.6内核下开启文件打开数为65535，worker\_rlimit\_nofile就相应应该填写65535。

#这是因为nginx调度时分配请求到进程并不是那么的均衡，所以假如填写10240，总并发量达到3-4万时就有进程可能超过10240了，这时会返回502错误。

worker\_rlimit\_nofile 65535;

events

{

#参考事件模型，use [ kqueue | rtsig | epoll | /dev/poll | select | poll ]; epoll模型

#是Linux 2.6以上版本内核中的高性能网络I/O模型，linux建议epoll，如果跑在FreeBSD上面，就用kqueue模型。

#补充说明：

#与apache相类，nginx针对不同的操作系统，有不同的事件模型

#A）标准事件模型

#Select、poll属于标准事件模型，如果当前系统不存在更有效的方法，nginx会选择select或poll

#B）高效事件模型

#Kqueue：使用于FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 和 MacOS X.使用双处理器的MacOS X系统使用kqueue可能会造成内核崩溃。

#Epoll：使用于Linux内核2.6版本及以后的系统。

#/dev/poll：使用于Solaris 7 11/99+，HP/UX 11.22+ (eventport)，IRIX 6.5.15+ 和 Tru64 UNIX 5.1A+。

#Eventport：使用于Solaris 10。 为了防止出现内核崩溃的问题， 有必要安装安全补丁。

use epoll;

#单个进程最大连接数（最大连接数=连接数\*进程数）

#根据硬件调整，和前面工作进程配合起来用，尽量大，但是别把cpu跑到100%就行。每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为。

worker\_connections 65535;

#keepalive超时时间。

keepalive\_timeout 60;

#客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求头的大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。

#分页大小可以用命令getconf PAGESIZE 取得。

#[root@web001 ~]# getconf PAGESIZE

#4096

#但也有client\_header\_buffer\_size超过4k的情况，但是client\_header\_buffer\_size该值必须设置为“系统分页大小”的整倍数。

client\_header\_buffer\_size 4k;

#这个将为打开文件指定缓存，默认是没有启用的，max指定缓存数量，建议和打开文件数一致，inactive是指经过多长时间文件没被请求后删除缓存。

open\_file\_cache max=65535 inactive=60s;

#这个是指多长时间检查一次缓存的有效信息。

#语法:open\_file\_cache\_valid time 默认值:open\_file\_cache\_valid 60 使用字段:http, server, location 这个指令指定了何时需要检查open\_file\_cache中缓存项目的有效信息.

open\_file\_cache\_valid 80s;

#open\_file\_cache指令中的inactive参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive时间内一次没被使用，它将被移除。

#语法:open\_file\_cache\_min\_uses number 默认值:open\_file\_cache\_min\_uses 1 使用字段:http, server, location 这个指令指定了在open\_file\_cache指令无效的参数中一定的时间范围内可以使用的最小文件数,如果使用更大的值,文件描述符在cache中总是打开状态.

open\_file\_cache\_min\_uses 1;

#语法:open\_file\_cache\_errors on | off 默认值:open\_file\_cache\_errors off 使用字段:http, server, location 这个指令指定是否在搜索一个文件是记录cache错误.

open\_file\_cache\_errors on;

}

#设定http服务器，利用它的反向代理功能提供负载均衡支持

http

{

#文件扩展名与文件类型映射表

include /etc/nginx/mime.types;

#默认文件类型

default\_type application/octet-stream;

#默认编码

#charset utf-8;

#服务器名字的hash表大小

#保存服务器名字的hash表是由指令server\_names\_hash\_max\_size 和server\_names\_hash\_bucket\_size所控制的。参数hash bucket size总是等于hash表的大小，并且是一路处理器缓存大小的倍数。在减少了在内存中的存取次数后，使在处理器中加速查找hash表键值成为可能。如果hash bucket size等于一路处理器缓存的大小，那么在查找键的时候，最坏的情况下在内存中查找的次数为2。第一次是确定存储单元的地址，第二次是在存储单元中查找键 值。因此，如果Nginx给出需要增大hash max size 或 hash bucket size的提示，那么首要的是增大前一个参数的大小.

server\_names\_hash\_bucket\_size 128;

#客户端请求头部的缓冲区大小。这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE取得。

client\_header\_buffer\_size 32k;

#客户请求头缓冲大小。nginx默认会用client\_header\_buffer\_size这个buffer来读取header值，如果header过大，它会使用large\_client\_header\_buffers来读取。

large\_client\_header\_buffers 4 64k;

#设定通过nginx上传文件的大小

client\_max\_body\_size 8m;

#开启高效文件传输模式，sendfile指令指定nginx是否调用sendfile函数来输出文件，对于普通应用设为 on，如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络I/O处理速度，降低系统的负载。注意：如果图片显示不正常把这个改成off。

#sendfile指令指定 nginx 是否调用sendfile 函数（zero copy 方式）来输出文件，对于普通应用，必须设为on。如果用来进行下载等应用磁盘IO重负载应用，可设置为off，以平衡磁盘与网络IO处理速度，降低系统uptime。

sendfile on;

#开启目录列表访问，合适下载服务器，默认关闭。

autoindex on;

#此选项允许或禁止使用socke的TCP\_CORK的选项，此选项仅在使用sendfile的时候使用

tcp\_nopush on;

tcp\_nodelay on;

#长连接超时时间，单位是秒

keepalive\_timeout 120;

#FastCGI相关参数是为了改善网站的性能：减少资源占用，提高访问速度。下面参数看字面意思都能理解。

fastcgi\_connect\_timeout 300;

fastcgi\_send\_timeout 300;

fastcgi\_read\_timeout 300;

fastcgi\_buffer\_size 64k;

fastcgi\_buffers 4 64k;

fastcgi\_busy\_buffers\_size 128k;

fastcgi\_temp\_file\_write\_size 128k;

#gzip模块设置

gzip on; #开启gzip压缩输出

gzip\_min\_length 1k; #最小压缩文件大小

gzip\_buffers 4 16k; #压缩缓冲区

gzip\_http\_version 1.0; #压缩版本（默认1.1，前端如果是squid2.5请使用1.0）

gzip\_comp\_level 2; #压缩等级

gzip\_types text/plain application/x-javascript text/css application/xml; #压缩类型，默认就已经包含textml，所以下面就不用再写了，写上去也不会有问题，但是会有一个warn。

gzip\_vary on;

#开启限制IP连接数的时候需要使用

#limit\_zone crawler $binary\_remote\_addr 10m;

#负载均衡配置

upstream piao.jd.com {

#upstream的负载均衡，weight是权重，可以根据机器配置定义权重。weigth参数表示权值，权值越高被分配到的几率越大。

server 192.168.80.121:80 weight=3;

server 192.168.80.122:80 weight=2;

server 192.168.80.123:80 weight=3;

#nginx的upstream目前支持4种方式的分配

#1、轮询（默认）

#每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

#2、weight

#指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

#例如：

#upstream bakend {

# server 192.168.0.14 weight=10;

# server 192.168.0.15 weight=10;

#}

#2、ip\_hash

#每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

#例如：

#upstream bakend {

# ip\_hash;

# server 192.168.0.14:88;

# server 192.168.0.15:80;

#}

#3、fair（第三方）

#按后端服务器的响应时间来分配请求，响应时间短的优先分配。

#upstream backend {

# server server1;

# server server2;

# fair;

#}

#4、url\_hash（第三方）

#按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

#例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash\_method是使用的hash算法

#upstream backend {

# server squid1:3128;

# server squid2:3128;

# hash $request\_uri;

# hash\_method crc32;

#}

#tips:

#upstream bakend{#定义负载均衡设备的Ip及设备状态}{

# ip\_hash;

# server 127.0.0.1:9090 down;

# server 127.0.0.1:8080 weight=2;

# server 127.0.0.1:6060;

# server 127.0.0.1:7070 backup;

#}

#在需要使用负载均衡的server中增加 proxy\_pass http://bakend/;

#每个设备的状态设置为:

#1.down表示单前的server暂时不参与负载

#2.weight为weight越大，负载的权重就越大。

#3.max\_fails：允许请求失败的次数默认为1.当超过最大次数时，返回proxy\_next\_upstream模块定义的错误

#4.fail\_timeout:max\_fails次失败后，暂停的时间。

#5.backup： 其它所有的非backup机器down或者忙的时候，请求backup机器。所以这台机器压力会最轻。

#nginx支持同时设置多组的负载均衡，用来给不用的server来使用。

#client\_body\_in\_file\_only设置为On 可以讲client post过来的数据记录到文件中用来做debug

#client\_body\_temp\_path设置记录文件的目录 可以设置最多3层目录

#location对URL进行匹配.可以进行重定向或者进行新的代理 负载均衡

}

# How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 18.04

[Nginx](https://www.digitalocean.com/community/tags/nginx)[Ubuntu](https://www.digitalocean.com/community/tags/ubuntu)[Python](https://www.digitalocean.com/community/tags/python)[Python Frameworks](https://www.digitalocean.com/community/tags/python-frameworks)[Ubuntu 18.04](https://www.digitalocean.com/community/tags/ubuntu-18-04)

<https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-18-04>

[[](https://www.digitalocean.com/community/users/katjuell)](https://www.digitalocean.com/community/users/katjuell)

* **By**[**Justin Ellingwood**](https://www.digitalocean.com/community/users/jellingwood)**and**[**Kathleen Juell**](https://www.digitalocean.com/community/users/katjuell)

**PostedJuly 13, 2018 297.6kviews**

### Introduction

In this guide, you will build a Python application using the Flask microframework on Ubuntu 18.04. The bulk of this article will be about how to set up the [Gunicorn application server](http://gunicorn.org/) and how to launch the application and configure [Nginx](https://www.nginx.com/) to act as a front-end reverse proxy.

## Prerequisites

Before starting this guide, you should have:

* A server with Ubuntu 18.04 installed and a non-root user with sudo privileges. Follow our [initial server setup guide](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-18-04) for guidance.
* Nginx installed, following Steps 1 and 2 of [How To Install Nginx on Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-18-04).
* A domain name configured to point to your server. You can purchase one on [Namecheap](https://namecheap.com/) or get one for free on [Freenom](http://www.freenom.com/en/index.html). You can learn how to point domains to DigitalOcean by following the relevant [documentation on domains and DNS](https://www.digitalocean.com/docs/networking/dns/). Be sure to create the following DNS records:
  + An A record with your\_domain pointing to your server’s public IP address.
  + An A record with www.your\_domain pointing to your server’s public IP address.
* Familiarity with the WSGI specification, which the Gunicorn server will use to communicate with your Flask application. [This discussion](https://www.digitalocean.com/community/tutorials/how-to-set-up-uwsgi-and-nginx-to-serve-python-apps-on-ubuntu-14-04#definitions-and-concepts) covers WSGI in more detail.

## Step 1 — Installing the Components from the Ubuntu Repositories

Our first step will be to install all of the pieces we need from the Ubuntu repositories. This includes pip, the Python package manager, which will manage our Python components. We will also get the Python development files necessary to build some of the Gunicorn components.

First, let’s update the local package index and install the packages that will allow us to build our Python environment. These will include python3-pip, along with a few more packages and development tools necessary for a robust programming environment:

* sudo apt update
* sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev python3-setuptools

With these packages in place, let’s move on to creating a virtual environment for our project.

## Step 2 — Creating a Python Virtual Environment

Next, we’ll set up a virtual environment in order to isolate our Flask application from the other Python files on the system.

Start by installing the python3-venv package, which will install the venv module:

* sudo apt install python3-venv

Next, let’s make a parent directory for our Flask project. Move into the directory after you create it:

* mkdir ~/myproject
* cd ~/myproject

Create a virtual environment to store your Flask project’s Python requirements by typing:

* python3.6 -m venv myprojectenv

This will install a local copy of Python and pip into a directory called myprojectenv within your project directory.

Before installing applications within the virtual environment, you need to activate it. Do so by typing:

* source myprojectenv/bin/activate

Your prompt will change to indicate that you are now operating within the virtual environment. It will look something like this: (myprojectenv)user@host:~/myproject$.

## Step 3 — Setting Up a Flask Application

Now that you are in your virtual environment, you can install Flask and Gunicorn and get started on designing your application.

First, let’s install wheel with the local instance of pip to ensure that our packages will install even if they are missing wheel archives:

* pip install wheel

Note

Regardless of which version of Python you are using, when the virtual environment is activated, you should use the pip command (not pip3).

Next, let’s install Flask and Gunicorn:

* pip install gunicorn flask

### Creating a Sample App

Now that you have Flask available, you can create a simple application. Flask is a microframework. It does not include many of the tools that more full-featured frameworks might, and exists mainly as a module that you can import into your projects to assist you in initializing a web application.

While your application might be more complex, we’ll create our Flask app in a single file, called myproject.py:

* nano ~/myproject/myproject.py

The application code will live in this file. It will import Flask and instantiate a Flask object. You can use this to define the functions that should be run when a specific route is requested:

~/myproject/myproject.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def hello():

return "<h1 style='color:blue'>Hello There!</h1>"

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0')

Copy

This basically defines what content to present when the root domain is accessed. Save and close the file when you’re finished.

If you followed the initial server setup guide, you should have a UFW firewall enabled. To test the application, you need to allow access to port 5000:

* sudo ufw allow 5000

Now you can test your Flask app by typing:

* python myproject.py

You will see output like the following, including a helpful warning reminding you not to use this server setup in production:

Output

\* Serving Flask app "myproject" (lazy loading)

\* Environment: production

WARNING: Do not use the development server in a production environment.

Use a production WSGI server instead.

\* Debug mode: off

\* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

Visit your server’s IP address followed by :5000 in your web browser:

http://your\_server\_ip:5000

You should see something like this:

Flask sample app

When you are finished, hit CTRL-C in your terminal window to stop the Flask development server.

### Creating the WSGI Entry Point

Next, let’s create a file that will serve as the entry point for our application. This will tell our Gunicorn server how to interact with the application.

Let’s call the file wsgi.py:

* nano ~/myproject/wsgi.py

In this file, let’s import the Flask instance from our application and then run it:

~/myproject/wsgi.py

from myproject import app

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

Copy

Save and close the file when you are finished.

## Step 4 — Configuring Gunicorn

Your application is now written with an entry point established. We can now move on to configuring Gunicorn.

Before moving on, we should check that Gunicorn can serve the application correctly.

We can do this by simply passing it the name of our entry point. This is constructed as the name of the module (minus the .py extension), plus the name of the callable within the application. In our case, this is wsgi:app.

We’ll also specify the interface and port to bind to so that the application will be started on a publicly available interface:

* cd ~/myproject
* gunicorn --bind 0.0.0.0:5000 wsgi:app

You should see output like the following:

Output

[2018-07-13 19:35:13 +0000] [28217] [INFO] Starting gunicorn 19.9.0

[2018-07-13 19:35:13 +0000] [28217] [INFO] Listening at: http://0.0.0.0:5000 (28217)

[2018-07-13 19:35:13 +0000] [28217] [INFO] Using worker: sync

[2018-07-13 19:35:13 +0000] [28220] [INFO] Booting worker with pid: 28220

Visit your server’s IP address with :5000 appended to the end in your web browser again:

http://your\_server\_ip:5000

You should see your application’s output:

Flask sample app

When you have confirmed that it’s functioning properly, press CTRL-C in your terminal window.

We’re now done with our virtual environment, so we can deactivate it:

* deactivate

Any Python commands will now use the system’s Python environment again.

Next, let’s create the systemd service unit file. Creating a systemd unit file will allow Ubuntu’s init system to automatically start Gunicorn and serve the Flask application whenever the server boots.

Create a unit file ending in .service within the /etc/systemd/system directory to begin:

* sudo nano /etc/systemd/system/myproject.service

Inside, we’ll start with the [Unit] section, which is used to specify metadata and dependencies. Let’s put a description of our service here and tell the init system to only start this after the networking target has been reached:

/etc/systemd/system/myproject.service

[Unit]

Description=Gunicorn instance to serve myproject

After=network.target

Next, let’s open up the [Service] section. This will specify the user and group that we want the process to run under. Let’s give our regular user account ownership of the process since it owns all of the relevant files. Let’s also give group ownership to the www-data group so that Nginx can communicate easily with the Gunicorn processes. Remember to replace the username here with your username:

/etc/systemd/system/myproject.service

[Unit]

Description=Gunicorn instance to serve myproject

After=network.target

[Service]

User=sammy

Group=www-data

Next, let’s map out the working directory and set the PATH environmental variable so that the init system knows that the executables for the process are located within our virtual environment. Let’s also specify the command to start the service. This command will do the following:

* Start 3 worker processes (though you should adjust this as necessary)
* Create and bind to a Unix socket file, myproject.sock, within our project directory. We’ll set an umask value of 007 so that the socket file is created giving access to the owner and group, while restricting other access
* Specify the WSGI entry point file name, along with the Python callable within that file (wsgi:app)

Systemd requires that we give the full path to the Gunicorn executable, which is installed within our virtual environment.

Remember to replace the username and project paths with your own information:

/etc/systemd/system/myproject.service

[Unit]

Description=Gunicorn instance to serve myproject

After=network.target

[Service]

User=sammy

Group=www-data

WorkingDirectory=/home/sammy/myproject

Environment="PATH=/home/sammy/myproject/myprojectenv/bin"

ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock -m 007 wsgi:app

Finally, let’s add an [Install] section. This will tell systemd what to link this service to if we enable it to start at boot. We want this service to start when the regular multi-user system is up and running:

/etc/systemd/system/myproject.service

[Unit]

Description=Gunicorn instance to serve myproject

After=network.target

[Service]

User=sammy

Group=www-data

WorkingDirectory=/home/sammy/myproject

Environment="PATH=/home/sammy/myproject/myprojectenv/bin"

ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock -m 007 wsgi:app

[Install]

WantedBy=multi-user.target

With that, our systemd service file is complete. Save and close it now.

We can now start the Gunicorn service we created and enable it so that it starts at boot:

* sudo systemctl start myproject
* sudo systemctl enable myproject

Let’s check the status:

* sudo systemctl status myproject

You should see output like this:

Output

● myproject.service - Gunicorn instance to serve myproject

Loaded: loaded (/etc/systemd/system/myproject.service; enabled; vendor preset: enabled)

Active: active (running) since Fri 2018-07-13 14:28:39 UTC; 46s ago

Main PID: 28232 (gunicorn)

Tasks: 4 (limit: 1153)

CGroup: /system.slice/myproject.service

├─28232 /home/sammy/myproject/myprojectenv/bin/python3.6 /home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock -m 007

├─28250 /home/sammy/myproject/myprojectenv/bin/python3.6 /home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock -m 007

├─28251 /home/sammy/myproject/myprojectenv/bin/python3.6 /home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock -m 007

└─28252 /home/sammy/myproject/myprojectenv/bin/python3.6 /home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock -m 007

If you see any errors, be sure to resolve them before continuing with the tutorial.

## Step 5 — Configuring Nginx to Proxy Requests

Our Gunicorn application server should now be up and running, waiting for requests on the socket file in the project directory. Let’s now configure Nginx to pass web requests to that socket by making some small additions to its configuration file.

Begin by creating a new server block configuration file in Nginx’s sites-available directory. Let’s call this myproject to keep in line with the rest of the guide:

* sudo nano /etc/nginx/sites-available/myproject

Open up a server block and tell Nginx to listen on the default port 80. Let’s also tell it to use this block for requests for our server’s domain name:

/etc/nginx/sites-available/myproject

server {

listen 80;

server\_name your\_domain www.your\_domain;

}

Next, let’s add a location block that matches every request. Within this block, we’ll include the proxy\_params file that specifies some general proxying parameters that need to be set. We’ll then pass the requests to the socket we defined using the proxy\_pass directive:

/etc/nginx/sites-available/myproject

server {

listen 80;

server\_name your\_domain www.your\_domain;

location / {

include proxy\_params;

proxy\_pass http://unix:/home/sammy/myproject/myproject.sock;

}

}

Save and close the file when you’re finished.

To enable the Nginx server block configuration you’ve just created, link the file to the sites-enabled directory:

* sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled

With the file in that directory, you can test for syntax errors:

* sudo nginx -t

If this returns without indicating any issues, restart the Nginx process to read the new configuration:

* sudo systemctl restart nginx

Finally, let’s adjust the firewall again. We no longer need access through port 5000, so we can remove that rule. We can then allow full access to the Nginx server:

* sudo ufw delete allow 5000
* sudo ufw allow 'Nginx Full'

You should now be able to navigate to your server’s domain name in your web browser:

http://your\_domain

You should see your application’s output:

Flask sample app

If you encounter any errors, trying checking the following:

* sudo less /var/log/nginx/error.log: checks the Nginx error logs.
* sudo less /var/log/nginx/access.log: checks the Nginx access logs.
* sudo journalctl -u nginx: checks the Nginx process logs.
* sudo journalctl -u myproject: checks your Flask app’s Gunicorn logs.

## Step 6 — Securing the Application

To ensure that traffic to your server remains secure, let’s get an SSL certificate for your domain. There are multiple ways to do this, including getting a free certificate from [Let’s Encrypt](https://letsencrypt.org/), [generating a self-signed certificate](https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-in-ubuntu-18-04), or [buying one from another provider](https://www.digitalocean.com/community/tutorials/how-to-install-an-ssl-certificate-from-a-commercial-certificate-authority) and configuring Nginx to use it by following Steps 2 through 6 of  [How to Create a Self-signed SSL Certificate for Nginx in Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-in-ubuntu-18-04#step-2-%E2%80%93-configuring-nginx-to-use-ssl). We will go with option one for the sake of expediency.

First, add the Certbot Ubuntu repository:

* sudo add-apt-repository ppa:certbot/certbot

You’ll need to press ENTER to accept.

Install Certbot’s Nginx package with apt:

* sudo apt install python-certbot-nginx

Certbot provides a variety of ways to obtain SSL certificates through plugins. The Nginx plugin will take care of reconfiguring Nginx and reloading the config whenever necessary. To use this plugin, type the following:

* sudo certbot --nginx -d your\_domain -d www.your\_domain

This runs certbot with the --nginx plugin, using -d to specify the names we’d like the certificate to be valid for.

If this is your first time running certbot, you will be prompted to enter an email address and agree to the terms of service. After doing so, certbot will communicate with the Let’s Encrypt server, then run a challenge to verify that you control the domain you’re requesting a certificate for.

If that’s successful, certbot will ask how you’d like to configure your HTTPS settings:

Output

Please choose whether or not to redirect HTTP traffic to HTTPS, removing HTTP access.

-------------------------------------------------------------------------------

1: No redirect - Make no further changes to the webserver configuration.

2: Redirect - Make all requests redirect to secure HTTPS access. Choose this for

new sites, or if you're confident your site works on HTTPS. You can undo this

change by editing your web server's configuration.

-------------------------------------------------------------------------------

Select the appropriate number [1-2] then [enter] (press 'c' to cancel):

Select your choice then hit ENTER. The configuration will be updated, and Nginx will reload to pick up the new settings. certbot will wrap up with a message telling you the process was successful and where your certificates are stored:

Output

IMPORTANT NOTES:

- Congratulations! Your certificate and chain have been saved at:

/etc/letsencrypt/live/your\_domain/fullchain.pem

Your key file has been saved at:

/etc/letsencrypt/live/your\_domain/privkey.pem

Your cert will expire on 2018-07-23. To obtain a new or tweaked

version of this certificate in the future, simply run certbot again

with the "certonly" option. To non-interactively renew \*all\* of

your certificates, run "certbot renew"

- Your account credentials have been saved in your Certbot

configuration directory at /etc/letsencrypt. You should make a

secure backup of this folder now. This configuration directory will

also contain certificates and private keys obtained by Certbot so

making regular backups of this folder is ideal.

- If you like Certbot, please consider supporting our work by:

Donating to ISRG / Let's Encrypt: https://letsencrypt.org/donate

Donating to EFF: https://eff.org/donate-le

If you followed the Nginx installation instructions in the prerequisites, you will no longer need the redundant HTTP profile allowance:

* sudo ufw delete allow 'Nginx HTTP'

To verify the configuration, navigate once again to your domain, using https://:

https://your\_domain

You should see your application output once again, along with your browser’s security indicator, which should indicate that the site is secured.

## Conclusion

In this guide, you created and secured a simple Flask application within a Python virtual environment. You created a WSGI entry point so that any WSGI-capable application server can interface with it, and then configured the Gunicorn app server to provide this function. Afterwards, you created a systemd service file to automatically launch the application server on boot. You also created an Nginx server block that passes web client traffic to the application server, relaying external requests, and secured traffic to your server with Let’s Encrypt.

Flask is a very simple, but extremely flexible framework meant to provide your applications with functionality without being too restrictive about structure and design. You can use the general stack described in this guide to serve the flask applications that you design.

# How To Set Up Nginx Server Blocks (Virtual Hosts) on Ubuntu 16.04

[Nginx](https://www.digitalocean.com/community/tags/nginx)[Ubuntu](https://www.digitalocean.com/community/tags/ubuntu)[Getting Started](https://www.digitalocean.com/community/tags/getting-started)[Ubuntu 16.04](https://www.digitalocean.com/community/tags/ubuntu-16-04)

<https://www.digitalocean.com/community/tutorials/how-to-set-up-nginx-server-blocks-virtual-hosts-on-ubuntu-16-04>

* **By**[**Justin Ellingwood**](https://www.digitalocean.com/community/users/jellingwood)

**PostedMay 19, 2016 1.4mviews**

### Introduction

When using the Nginx web server, **server blocks** (similar to the virtual hosts in Apache) can be used to encapsulate configuration details and host more than one domain off of a single server.

In this guide, we’ll discuss how to configure server blocks in Nginx on an Ubuntu 16.04 server.

## Prerequisites

We’re going to be using a non-root user with sudo privileges throughout this tutorial. If you do not have a user like this configured, you can create one by following our [Ubuntu 16.04 initial server setup](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-16-04) guide.

You will also need to have Nginx installed on your server. The following guides cover this procedure:

* [How To Install Nginx on Ubuntu 16.04](https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-16-04): Use this guide to set up Nginx on its own.
* [How To Install Linux, Nginx, MySQL, PHP (LEMP stack) in Ubuntu 16.04](https://www.digitalocean.com/community/tutorials/how-to-install-linux-nginx-mysql-php-lemp-stack-in-ubuntu-16-04): Use this guide if you will be using Nginx in conjunction with MySQL and PHP.

When you have fulfilled these requirements, you can continue on with this guide.

## Example Configuration

For demonstration purposes, we’re going to set up two domains with our Nginx server. The domain names we’ll use in this guide are **example.com** and **test.com**.

You can find a guide on how to set up domain names with DigitalOcean [here](https://www.digitalocean.com/community/articles/how-to-set-up-a-host-name-with-digitalocean). If you do not have two spare domain names to play with, use dummy names for now and we’ll show you later how to configure your local computer to test your configuration.

## Step One: Set Up New Document Root Directories

By default, Nginx on Ubuntu 16.04 has one server block enabled by default. It is configured to serve documents out of a directory at /var/www/html.

While this works well for a single site, we need additional directories if we’re going to serve multiple sites. We can consider the /var/www/html directory the default directory that will be served if the client request doesn’t match any of our other sites.

We will create a directory structure within /var/www for each of our sites. The actual web content will be placed in an html directory within these site-specific directories. This gives us some additional flexibility to create other directories associated with our sites as siblings to the html directory if necessary.

We need to create these directories for each of our sites. The -p flag tells mkdir to create any necessary parent directories along the way:

* sudo mkdir -p /var/www/example.com/html
* sudo mkdir -p /var/www/test.com/html

Now that we have our directories, we will reassign ownership of the web directories to our normal user account. This will let us write to them without sudo.

Note

Depending on your needs, you might need to adjust the permissions or ownership of the folders again to allow certain access to the www-data user. For instance, dynamic sites will often need this. The specific permissions and ownership requirements entirely depend on what your configuration. Follow the recommendations for the specific technology you’re using.

We can use the $USER environmental variable to assign ownership to the account that we are currently signed in on (make sure you’re not logged in as root). This will allow us to easily create or edit the content in this directory:

* sudo chown -R $USER:$USER /var/www/example.com/html
* sudo chown -R $USER:$USER /var/www/test.com/html

The permissions of our web roots should be correct already if you have not modified your umask value, but we can make sure by typing:

* sudo chmod -R 755 /var/www

Our directory structure is now configured and we can move on.

## Step Two: Create Sample Pages for Each Site

Now that we have our directory structure set up, let’s create a default page for each of our sites so that we will have something to display.

Create an index.html file in your first domain:

* nano /var/www/example.com/html/index.html

Inside the file, we’ll create a really basic file that indicates what site we are currently accessing. It will look like this:

/var/www/example.com/html/index.html

<html>

<head>

<title>Welcome to Example.com!</title>

</head>

<body>

<h1>Success! The example.com server block is working!</h1>

</body>

</html>

Save and close the file when you are finished.

Since the file for our second site is basically going to be the same, we can copy it over to our second document root like this:

* cp /var/www/example.com/html/index.html /var/www/test.com/html/

Now, we can open the new file in our editor:

* nano /var/www/test.com/html/index.html

Modify it so that it refers to our second domain:

/var/www/test.com/html/index.html

<html>

<head>

<title>Welcome to Test.com!</title>

</head>

<body>

<h1>Success! The test.com server block is working!</h1>

</body>

</html>

Save and close this file when you are finished. We now have some pages to display to visitors of our two domains.

## Step Three: Create Server Block Files for Each Domain

Now that we have the content we wish to serve, we need to actually create the server blocks that will tell Nginx how to do this.

By default, Nginx contains one server block called default which we can use as a template for our own configurations. We will begin by designing our first domain’s server block, which we will then copy over for our second domain and make the necessary modifications.

### Create the First Server Block File

As mentioned above, we will create our first server block config file by copying over the default file:

* sudo cp /etc/nginx/sites-available/default /etc/nginx/sites-available/example.com

Now, open the new file you created in your text editor with sudo privileges:

* sudo nano /etc/nginx/sites-available/example.com

Ignoring the commented lines, the file will look similar to this:

/etc/nginx/sites-available/example.com

server {

listen 80 default\_server;

listen [::]:80 default\_server;

root /var/www/html;

index index.html index.htm index.nginx-debian.html;

server\_name \_;

location / {

try\_files $uri $uri/ =404;

}

}

First, we need to look at the listen directives. **Only one of our server blocks on the server can have the default\_server option enabled.** This specifies which block should serve a request if the server\_name requested does not match any of the available server blocks. This shouldn’t happen very frequently in real world scenarios since visitors will be accessing your site through your domain name.

You can choose to designate one of your sites as the “default” by including the default\_server option in the listen directive, or you can leave the default server block enabled, which will serve the content of the /var/www/html directory if the requested host cannot be found.

In this guide, we’ll leave the default server block in place to serve non-matching requests, so we’ll remove the default\_server from this and the next server block. You can choose to add the option to whichever of your server blocks makes sense to you.

/etc/nginx/sites-available/example.com

server {

listen 80;

listen [::]:80;

. . .

}

Note

You can check that the default\_server option is only enabled in a single active file by typing:

* grep -R default\_server /etc/nginx/sites-enabled/

If matches are found uncommented in more than on file (shown in the leftmost column), Nginx will complain about an invalid configuration.

The next thing we’re going to have to adjust is the document root, specified by the root directive. Point it to the site’s document root that you created:

/etc/nginx/sites-available/example.com

server {

listen 80;

listen [::]:80;

root /var/www/example.com/html;

}

Next, we need to modify the server\_name to match requests for our first domain. We can additionally add any aliases that we want to match. We will add a www.example.com alias to demonstrate.

When you are finished, your file will look something like this:

/etc/nginx/sites-available/example.com

server {

listen 80;

listen [::]:80;

root /var/www/example.com/html;

index index.html index.htm index.nginx-debian.html;

server\_name example.com www.example.com;

location / {

try\_files $uri $uri/ =404;

}

}

That is all we need for a basic configuration. Save and close the file to exit.

### Create the Second Server Block File

Now that we have our initial server block configuration, we can use that as a basis for our second file. Copy it over to create a new file:

* sudo cp /etc/nginx/sites-available/example.com /etc/nginx/sites-available/test.com

Open the new file with sudo privileges in your editor:

* sudo nano /etc/nginx/sites-available/test.com

Again, make sure that you do not use the default\_server option for the listen directive in this file if you’ve already used it elsewhere. Adjust the root directive to point to your second domain’s document root and adjust the server\_name to match your second site’s domain name (make sure to include any aliases).

When you are finished, your file will likely look something like this:

/etc/nginx/sites-available/test.com

server {

listen 80;

listen [::]:80;

root /var/www/test.com/html;

index index.html index.htm index.nginx-debian.html;

server\_name test.com www.test.com;

location / {

try\_files $uri $uri/ =404;

}

}

When you are finished, save and close the file.

## Step Four: Enable your Server Blocks and Restart Nginx

Now that we have our server block files, we need to enable them. We can do this by creating symbolic links from these files to the sites-enabled directory, which Nginx reads from during startup.

We can create these links by typing:

* sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/
* sudo ln -s /etc/nginx/sites-available/test.com /etc/nginx/sites-enabled/

These files are now in the enabled directory. We now have three server blocks enabled, which are configured to respond based on their listen directive and the server\_name (you can read more about how Nginx processes these directives [here](https://www.digitalocean.com/community/tutorials/understanding-nginx-server-and-location-block-selection-algorithms)):

* example.com: Will respond to requests for example.com and www.example.com
* test.com: Will respond to requests for test.com and www.test.com
* default: Will respond to any requests on port 80 that do not match the other two blocks.

In order to avoid a possible hash bucket memory problem that can arise from adding additional server names, we will go ahead and adjust a single value within our /etc/nginx/nginx.conf file. Open the file now:

* sudo nano /etc/nginx/nginx.conf

Within the file, find the server\_names\_hash\_bucket\_size directive. Remove the # symbol to uncomment the line:

/etc/nginx/nginx.conf

http {

. . .

server\_names\_hash\_bucket\_size 64;

. . .

}

Save and close the file when you are finished.

Next, test to make sure that there are no syntax errors in any of your Nginx files:

* sudo nginx -t

If no problems were found, restart Nginx to enable your changes:

* sudo systemctl restart nginx

Nginx should now be serving both of your domain names.

## Step Five: Modify Your Local Hosts File for Testing(Optional)

If you have not been using domain names that you own and instead have been using dummy values, you can modify your local computer’s configuration to let you to temporarily test your Nginx server block configuration.

This will not allow other visitors to view your site correctly, but it will give you the ability to reach each site independently and test your configuration. This basically works by intercepting requests that would usually go to DNS to resolve domain names. Instead, we can set the IP addresses we want our local computer to go to when we request the domain names.

Note

Make sure you are operating on your local computer during these steps and not your VPS server. You will need to have root access, be a member of the administrative group, or otherwise be able to edit system files to do this.

If you are on a Mac or Linux computer at home, you can edit the file needed by typing:

* sudo nano /etc/hosts

If you are on Windows, you can [find instructions for altering your hosts file](http://www.thewindowsclub.com/hosts-file-in-windows) here.

You need to know your server’s public IP address and the domains you want to route to the server. Assuming that my server’s public IP address is 203.0.113.5, the lines I would add to my file would look something like this:

/etc/hosts

127.0.0.1 localhost

. . .

203.0.113.5 example.com www.example.com

203.0.113.5 test.com www.test.com

This will intercept any requests for example.com and test.com and send them to your server, which is what we want if we don’t actually own the domains that we are using.

Save and close the file when you are finished.

## Step Six: Test your Results

Now that you are all set up, you should test that your server blocks are functioning correctly. You can do that by visiting the domains in your web browser:

http://example.com

You should see a page that looks like this:

Nginx first server block

If you visit your second domain name, you should see a slightly different site:

http://test.com

Nginx second server block

If both of these sites work, you have successfully configured two independent server blocks with Nginx.

At this point, if you adjusted your hosts file on your local computer in order to test, you’ll probably want to remove the lines you added.

If you need domain name access to your server for a public-facing site, you will probably want to purchase a domain name for each of your sites. You can learn how to [set them up to point to your server](https://www.digitalocean.com/community/articles/how-to-set-up-a-host-name-with-digitalocean) here.

## Conclusion

You should now have the ability to create server blocks for each domain you wish to host from the same server. There aren’t any real limits on the number of server blocks you can create, so long as your hardware can handle the traffic.

**[nginx+gunicorn项目部署](https://www.cnblogs.com/gaidy/p/9784919.html)**

**1.1安装虚拟环境**

**创建文件夹**

mkdir data 目录文件夹

cd data 进入data文件夹

mkdir nginx 创建安装nginx的文件夹

mkdir server 存放代码的文件夹

mkdir logs 存放日志的文件夹

mkdir backup 备份代码的文件夹

mkdir softs 软件存放的位置

mkdir virtual 虚拟环境的位置

mkdir scripts 脚本的运行位置

mkdir scp\_codes 上传的代码

**安装虚拟环境**

apt-get install python-virtualenv （ubuntu已经安装过，可省略）

virtualenv -p /usr/bin/python3 api\_server (采用他，安装在本文件夹,)

mkvirtualenv api\_server(不要用它，它会自己安装到其他地方)

source api\_server/bin/activate (在虚拟环境的active进入虚拟环境)

安装的软件会在虚拟环境下的bin目录下

**安装nginx的环境**

解压

cd ~/data/softs/

tar xf pcre-8.39.tar.gz

配置

cd ~/data/softs/pcre-8.39

./configure

编译

make

安装

sudo make install

**安装nginx**

解压

cd ~/data/softs/

tar xf nginx-1.10.2.tar.gz

配置

cd nginx-1.10.2/

./configure --prefix=/root/data/nginx 安装在root/data下面的nginx文件夹

编译

make

安装

make install

查看进程

ps aux | grep nginx

**nginx简单基础操作**

检查

sudo ~/data/nginx/sbin/nginx -t

开启

sudo ~/data/nginx/sbin/nginx

关闭

sudo ~/data/nginx/sbin/nginx -s stop

重载

sudo ~/data/nginx/sbin/nginx -s reload

**1、首先安装Gunicorn**

pip install gunicorn

复制代码

**2、在入口文件的app.run()加上以下内容**

from werkzeug.contrib.fixers import ProxyFix

app.wsgi\_app = ProxyFix(app.wsgi\_app)

复制代码

例：

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello\_world():

return 'Hello World!'

if \_\_name\_\_ == '\_\_main\_\_':

from werkzeug.contrib.fixers import ProxyFix

app.wsgi\_app = ProxyFix(app.wsgi\_app)

app.run()

复制代码

**3、启动Gunicorn**

最简单的方式为

gunicorn 入口文件名:app

复制代码

默认是监听127.0.0.1:8000

如果是要处理高并发则要开多个进程和修改监听端口的画

gunicorn -w 4 -b 127.0.0.1:8000 入口文件名:app

复制代码保存在supervisord 里面的etc/supervisord.d目录下添加一个logo.ini配置文件

**添加配置文件**

vim /etc/supervisord.d/logo.ini

[program:logo\_api\_server]

directory = /data/api-service #代码存放的地方

command = /usr/local/python3/bin/gunicorn -w 4 -b :5005 LogoSeverApi:app #-w的参数根据CPU的核数来定，不要超过CPU的核数

#process\_name = %(process\_num)s ; process\_name expr (default %(program\_name)s)

#process\_name = %(process\_num)s

#numprocs = 4 ; number of processes copies to start (def 1)

numprocs\_start = 1

autostart = true ; start at supervisord start (default: true)

autorestart = unexpected ; whether/when to restart (default: unexpected)

startsecs = 10 ; number of secs prog must stay running (def. 1)

startretries = 3 ; max # of serial start failures (default 3)

user = dev

redirect\_stderr = true

stdout\_logfile\_maxbytes = 20MB

stdout\_logfile\_backups = 20

stdout\_logfile = /data/api-service/logs/supervisor.log

这样就可以启动4个进程同时处理HTTP请求，提高系统的使用效率及性能。 还可以把端口8000改为其他

而在实际应用中，应使用后台执行的方式启动服务

nohup 启动服务的命令 &

复制代码

即

nohup gunicorn -w 4 -b 127.0.0.1:8000 入口文件名:app &

复制代码

这时你可以在本机的浏览器上访问127.0.0.1:8000，浏览器上就会出现Hello World!

**注意：如果是想通过外网访问的话就要把ip改为内网的ip**

**4、配置nginx**

配置改为

server {

listen 80;

server\_name example.org; # 这是HOST机器的外部域名，用地址也行

location / {

proxy\_pass http://127.0.0.1:8000; # 这里是指向 gunicorn host 的服务地址

proxy\_set\_header Host $host;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

复制代码

这样启动之后80端口就可以转发到8000端口了

**5、检查配置**

nginx -t

复制代码

若出现一下内容，则表示配置成功

nginx: the configuration file /etc/nginx/nginx.conf syntax is ok

nginx: configuration file /etc/nginx/nginx.conf test is successful

复制代码

**6、更新nginx配置**

nginx -s reload

复制代码

在已经运行了Gunicorn的前提下，在浏览器访问127.0.0.1就会出现Hello World!了

**7、重启Gunicorn**

输入命令

pstree -ap|grep gunicorn

**1.2：安装wget(如果已经安装跳过)**

yum install wget

**1.3：安装python3.6（如果已经安装跳过）**

2.1.下载python3.6

wget <http://cdn.npm.taobao.org/dist/python/3.6.5/Python-3.6.5.tgz>

2.2.解压安装包

tar -zxvf Python-3.6.5.tgz

---------------------

**2.3安装python3.6的依赖包**

yum install -y gcc zlib\*

yum -y install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel gdbm-devel db4-devel libpcap-devel xz-devel

---------------------

**2.4.配置安装路径为/usr/local/python3**

./configure --prefix=/usr/local/python3 --with-ssl

make && make install

**2.5.将python3.6的路径加入到PATH中**

echo "export PATH=$PATH:/usr/local/python3/bin/" >>/etc/profile

source /etc/profile

**2.6.建立软连接**

ln -s /usr/local/python3/bin/python3 /usr/bin/python3

ln -s /usr/local/python3/bin/pip3 /usr/bin/pip3

---------------------

**2.7.安装supervisor**

yum install supervisor -y

如果报如下错误，安装yum的扩展源

安装yum的扩展源

yum install epel-\*

**2.8.添加supervisor的配置**

在/etc/supervisord.d目录下添加一个logo.ini配置文件

vim /etc/supervisord.d/logo.ini

内容如下：

[program:logo\_api\_server]

directory = /data/api-service #代码存放的地方

command = /usr/local/python3/bin/gunicorn -w 4 -b :5005 LogoSeverApi:app #-w的参数根据CPU的核数来定，不要超过CPU的核数

\#process\_name = %(process\_num)s ; process\_name expr (default %(program\_name)s)

\#process\_name = %(process\_num)s

\#numprocs = 4 ; number of processes copies to start (def 1)

numprocs\_start = 1

autostart = true ; start at supervisord start (default: true)

autorestart = unexpected ; whether/when to restart (default: unexpected)

startsecs = 10 ; number of secs prog must stay running (def. 1)

startretries = 3 ; max # of serial start failures (default 3)

user = dev

redirect\_stderr = true

stdout\_logfile\_maxbytes = 20MB

stdout\_logfile\_backups = 20

stdout\_logfile = /data/api-service/logs/supervisor.log

**2.9创建data目录**

mkdir /data

**2.10从git上面将代码下载下来，git在内网，如果是线上请将git下载下来然后通过ftp上传到线上坏境**

cd /data

git clone http://zhangxiaoyang:messi1020@git.epweike.net:3000/graphics/logo\_api api-servicels -

**2.11安装程序依赖包**

cd /data/api-service/

yum install cairo-devel

pip3 install --upgrade pip

pip3 install -r requirements.txt

CRM项目部署实际步骤：

（注意：需要添加user ‘ritchie‘ 到www-group，并且需要设置/var/www/html目录读写属性）

2020-10-17 CRM-server Ubuntu Desktop 18.04

(1) root@ubuntu:/home/ritchie# python3 -V

Python 3.6.9

(2) apt install python3-pip

(3) root@ubuntu:/home/ritchie# pip3 -V

pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)

(4) root@ubuntu:/home/ritchie# apt install python3-venv

(5) root@ubuntu:/home/ritchie# cd Documents

root@ubuntu:/home/ritchie/Documents# mkdir shares

root@ubuntu:/home/ritchie/Documents# /usr/bin/vmhgfs-fuse .host:/ /home/ritchie/Documents/shares -o subtype=vmhgfs-fuse,allow\_other

(6) root@ubuntu:/home/ritchie# cd /var

root@ubuntu:/var#

root@ubuntu:/var# mkdir www

root@ubuntu:/var# cd www

root@ubuntu:/var/www# mkdir html

root@ubuntu:/var/www# cd html

(7) Copy files:

root@ubuntu:/var/www/html# cp -r /home/ritchie/Documents/shares/financial-computing-app/requirements.txt /var/www/html/

root@ubuntu:/var/www/html# cp -r /home/ritchie/Documents/shares/financial-computing-app/app /var/www/html/

root@ubuntu:/var/www/html# cp -r /home/ritchie/Documents/shares/financial-computing-app/financial\_computing\_app.py /var/www/html/

(8) root@ubuntu:/var/www/html# python3 -m venv venv

(9) root@ubuntu:/var/www/html# source venv/bin/activate

(venv) root@ubuntu:/var/www/html#

(10) (venv) root@ubuntu:/var/www/html# pip3 install wheel

(venv) root@ubuntu:/var/www/html# pip3 install openpyxl

(venv) root@ubuntu:/var/www/html# pip3 install -r requirement.txt

(11) (venv) root@ubuntu:/var/www/html# python3 financial\_computing\_app.py

只显示下时间就结束了，原因在financial\_computing\_app.py文件

(venv) root@ubuntu:/var/www/html# nano financial\_computing\_app.py

from app import app

if \_\_name\_\_ == "\_\_main\_\_":

添加下面两行：

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

然后contr + o保存，contr + x退出

再试(venv) root@ubuntu:/var/www/html# python3 financial\_computing\_app.py 正常运行

(12) root@ubuntu:/home/ritchie# apt install net-tools 安装后ifconfig就可以运行，查看ip地址

(13) 再试(venv) root@ubuntu:/var/www/html# flask run --host=0.0.0.0 正常运行

(14) (venv) root@ubuntu:/var/www/html# pip3 --version和pip --version是一样的

所以安装gunicorn也可以用 pip install： (venv) root@ubuntu:/var/www/html# pip install gunicorn

(15) (venv) root@ubuntu:/var/www/html# gunicorn -w 4 -b 192.168.174.133:5000 --access-logfile log financial\_computing\_app:app

-w 4表示开启4个进程，-b表示绑定在哪个ip和端口，--access-logfile 文件目录及文件名 表示把log写在哪个文件，financial\_computing\_app:app表示入口文件和module（app=Flask(\_\_name\_\_)就是这个'app')）

另外： 目前是在终端情况下运行，必须保持terminal目前状态，不能做其他事情

如想在后台运行，则加入 -D参数即可

(venv) root@ubuntu:/var/www/html# gunicorn -w 4 -b 192.168.174.133:5000 -D --access-logfile log financial\_computing\_app:app

用(venv) root@ubuntu:/var/www/html# ps aux | grep gunicorn检查下，发现5个线程在后台运行，是1个主进程+4个进程，所以是5个

关闭后台运行，需要用kill：

(venv) root@ubuntu:/var/www/html# ps aux | grep gunicorn

root 17377 0.0 0.9 66364 19396 ? S 16:46 0:00 /var/www/html/venv/bin/python3 /var/www/html/venv/bin/gunicorn -w 4 -b 192.168.174.133:5000 -D --access-logfile log financial\_computing\_app:app

root 17381 0.1 2.7 120216 55720 ? S 16:46 0:00 /var/www/html/venv/bin/python3 /var/www/html/venv/bin/gunicorn -w 4 -b 192.168.174.133:5000 -D --access-logfile log financial\_computing\_app:app

root 17382 0.1 2.7 120216 55720 ? S 16:46 0:00 /var/www/html/venv/bin/python3 /var/www/html/venv/bin/gunicorn -w 4 -b 192.168.174.133:5000 -D --access-logfile log financial\_computing\_app:app

root 17383 0.1 2.7 120216 55724 ? S 16:46 0:00 /var/www/html/venv/bin/python3 /var/www/html/venv/bin/gunicorn -w 4 -b 192.168.174.133:5000 -D --access-logfile log financial\_computing\_app:app

root 17408 1.1 2.7 120224 55744 ? S 16:58 0:00 /var/www/html/venv/bin/python3 /var/www/html/venv/bin/gunicorn -w 4 -b 192.168.174.133:5000 -D --access-logfile log financial\_computing\_app:app

root 17422 0.0 0.0 14424 1084 pts/0 S+ 17:00 0:00 grep --color=auto gunicorn

(venv) root@ubuntu:/var/www/html# kill -9 17377

关了主进程17377，其他的4个进程也跟着关掉了（需要点时间）

(16) 如何同时开启2个或多个终端呢：把port端口写成不同即可，如（之前是5000，现在写成5001）

(venv) root@ubuntu:/var/www/html# gunicorn -w 4 -b 192.168.174.133:5001

(17) (venv) root@ubuntu:/var/www/html# apt install nginx

(18) root@ubuntu:/home/ritchie# ps aux | grep nginx 查看正在运行

(19) root@ubuntu:/home/ritchie# nano /etc/nginx/sites-available/default 修改这个文件进行设置

server\_name \_;

location / {

# First attempt to serve request as file, then

# as directory, then fall back to displaying a 404.

try\_files $uri $uri/ =404;

插入下面6行：(注意句尾的；号)

proxy\_pass http://localhost:5000/;

proxy\_redirect off;

proxy\_set\_header Host $http\_post;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

(20) 重启 nginx： root@ubuntu:/var/www/html# /etc/init.d/nginx restart

[ ok ] Restarting nginx (via systemctl): nginx.service.

(21) root@ubuntu:/var/www/html# /etc/init.d/nginx stop

-----------------

CRM-Server-UbuntuDesktop18.04

(1) Install UbuntuDesktop18.04 & Update software & Change timezone

(2) ritchie@ubuntu:~$ sudo apt-get install python3.8 >>> ritchie@ubuntu:~$ python3 --version >>> Python 3.6.9

(3) ritchie@ubuntu:~$ sudo apt install apache2

(4) ritchie@ubuntu:~$ sudo apt install python3-pip virtualenv >>> ritchie@ubuntu:~$ pip3 --version

pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)

(5) ritchie@ubuntu:~$ cd /var/www/html/

(6) ritchie@ubuntu:/var/www/html$ sudo mkdir site

(7) ritchie@ubuntu:/var/www/html$ cd site

(8) ritchie@ubuntu:/var/www/html/site$ sudo virtualenv venv

The path python2 (from --python=python2) does not exist <<<报错

(9) ritchie@ubuntu:/var/www/html/site$ sudo virtualenv --python=python3 venv <<<解决

(10) 共享文件夹，先在vmvare里面设置，然后ubuntu里面要使用mount命令：

---> ritchie@ubuntu:~/Documents$ sudo mkdir shares

---> ritchie@ubuntu:~/Documents$ sudo /usr/bin/vmhgfs-fuse .host:/ /home/ritchie/Documents/shares -o subtype=vmhgfs-fuse,allow\_other

ritchie@ubuntu:~/Documents$ cd /var/www/html/site

(11) Copy files:

---> ritchie@ubuntu:/var/www/html/site$ sudo cp -r /home/ritchie/Documents/shares/financial-computing-app/requirements.txt /var/www/html/site

---> ritchie@ubuntu:/var/www/html/site$ sudo cp -r /home/ritchie/Documents/shares/financial-computing-app/app /var/www/html/site

(12) ritchie@ubuntu:/var/www/html/site$ . vevn/bin/activate 运行virtual环境

(13) 安装requirement.txt中的plugin： sudo pip3 install -r requirement.txt

测试一下：

(14) ritchie@ubuntu:/var/www/html/site$ python3 >>> import flask >>> exit()

(15) ritchie@ubuntu:/var/www/html/site$ python3 init >>> 应该成功启动server

codeworked@ubuntu:/$ sudo apt install apache2 进入browser键入localhost网址显示apache默认界面，就是/var/www/index.html

codeworked@ubuntu:/$ python3 --version

codeworked@ubuntu:/$ sudo apt install python3-pip virtualenv

codeworked@ubuntu:/$ pip3 --version

codeworked@ubuntu:/$ cd /var/www/html

codeworked@ubuntu:/var/www/html$ sudo mkdir site

codeworked@ubuntu:/var/www/html$ cd site

codeworked@ubuntu:/var/www/html/site$ sudo mkdir static templates

codeworked@ubuntu:/var/www/html/site$ sudo virtualenv venv

codeworked@ubuntu:/var/www/html/site$ sudo nano init.py

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

retrun "Hello World"

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

codeworked@ubuntu:/var/www/html/site$ . venv/bin/activate 注意点后面有个空格

codeworked@ubuntu:/var/www/html/site$ python3 init.py >>> run successfully

codeworked@ubuntu:/var/www/html/site$ sudo apt install libapache2-mod-wsgi-py3

codeworked@ubuntu:/var/www/html/site$ sudo nano /etc/apache2/sites-available/000-defautl.conf

WSGIScriptAlias / / var/www/html/site/app.wsgi

<Directory /var/www/html/site>

Order allow,deny

Allow from all

</Directory>

codeworked@ubuntu:/var/www/html/site$ sudo nano app.wsgi

import sys

sys.path.inset(0, "/var/www/html/site")

from init import app as application

codeworked@ubuntu:/var/www/html/site$ sudo service apache2 restart

------ advanced -----

codeworked@ubuntu:/var/www/html/site$ sudo nano init.py

activate\_this\_file = "/var/www/html/site/venv/bin/activate\_this.py"

with open(activate\_this\_file) as \_file:

exec(\_file.read(), dict(\_\_file\_\_ = activate\_this\_file))

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def index():

retrun "Hello World"

@app.route("/done")

def done():

return "we are done with setup of flask ...."

# if \_\_name\_\_ == "\_\_main\_\_":

# app.run()

codeworked@ubuntu:/var/www/html/site$ sudo service apache2 restart

sudo systemctl status apache2.service -l --no-pager

sudo systemctl status httpd.service -l --no-pager

sudo journalctl -u apache2.service --since today --no-pager

sudo journalctl -u httpd.service --since today --no-pager

/etc/apache2/

sudo apachectl configtest

sudo nano /etc/apache2/apache2.conf