

将ASP.NET Core应用程序部署至生产环境中（CentOS7）

KAnts 开发Web那点事 2017-03-08

阅读目录

- 环境说明
- 准备你的ASP.NET Core应用程序
- 安装CentOS7
- 安装.NET Core SDK for CentOS7。
- 部署ASP.NET Core应用程序
- 配置Nginx
- 配置守护服务（Supervisor）

环境说明：

服务器系统：CentOS 7.2.1511

相关工具：Xshel、Xftp

服务器软件软件：.netcore、nginx、supervisor、polycoreutils-python

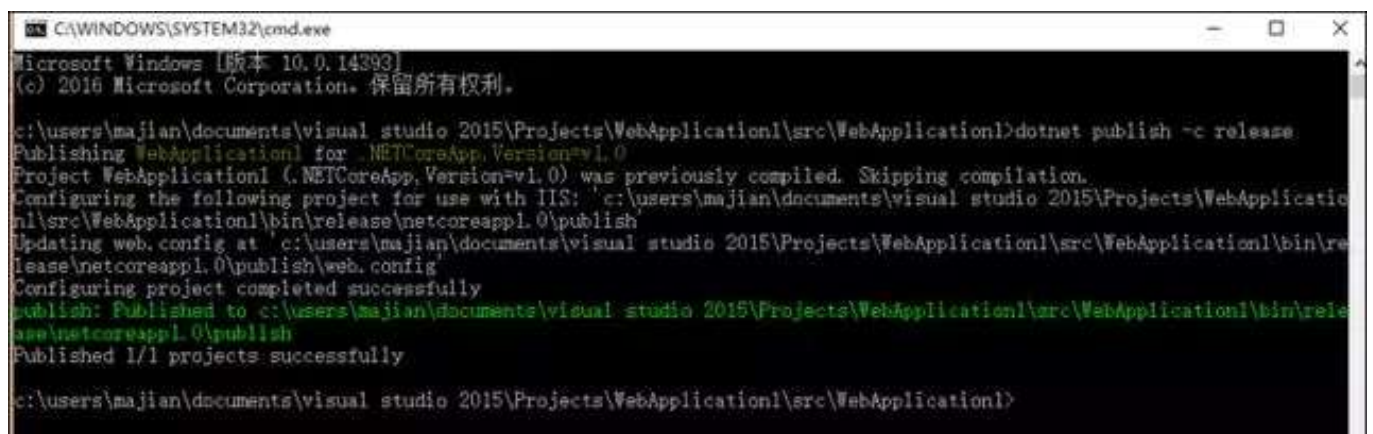
准备你的Asp.Net Core 应用程序

首先将你的应用程序以便携的模式进行发布。

ps:这边我使用一个空的Web项目来进行演示，因为本篇主要介绍生产环境的部署，与应用无关。

发布命令为：**dotnet publish -c release**

运行命令：**dotnet WebApplication1.dll**

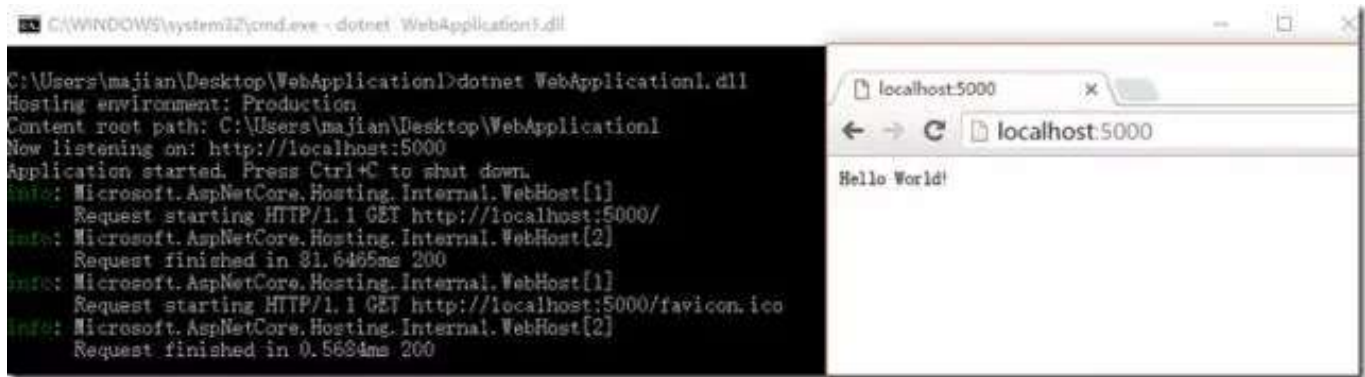


```
CA\WINDOWS\SYSTEM32\cmd.exe
Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation. 保留所有权利。

c:\users\majian\documents\visual studio 2015\Projects\WebApplication1\src\WebApplication1>dotnet publish -c release
Publishing WebApplication1 for .NETCoreApp,Version=v1.0
Project WebApplication1 (.NETCoreApp,Version=v1.0) was previously compiled. Skipping compilation.
Configuring the following project for use with IIS: 'c:\users\majian\documents\visual studio 2015\Projects\WebApplication1\src\WebApplication1\bin\release\netcoreapp1.0\publish'
Updating web.config at 'c:\users\majian\documents\visual studio 2015\Projects\WebApplication1\src\WebApplication1\bin\release\netcoreapp1.0\publish\web.config'
Configuring project completed successfully
publish: Published to c:\users\majian\documents\visual studio 2015\Projects\WebApplication1\src\WebApplication1\bin\release\netcoreapp1.0\publish
Published 1/1 projects successfully

c:\users\majian\documents\visual studio 2015\Projects\WebApplication1\src\WebApplication1>
```

确保这份发布应用可以在windows上运行，以减少后续的问题。



为什么不用自宿主的方式进行部署？

自宿主的发布方式进行部署会简单很多，为什么生产环境要使用便携的方式进行发布呢？

原因1：性能比便携式的低（主）。

原因2：微软给出的建议（次）。

口说无凭，有图有真相。

However, given its unique and cross-platform nature, .NET Core also has another angle through which to observe the type of the application and that is the application's *portability*. Portability essentially means where you can run your application and what prerequisites you need to satisfy in order for your application to run on a given machine. This document deals with this angle, portability, and outlines the two main types of portability that .NET Core enables.

An important thing to note is that in the .NET Core Tools Preview 2 timeframe, the self-contained application is being published from the NuGet cache on your machine. This means that all dependencies, including the actual .NET Core runtime and libraries, is not ready-to-run optimized, which means that it will have lower overall performance than portable applications. This is due to the fact that portable applications run against the installed .NET Core runtime and libraries which are ready-to-run.

参考地址：<https://docs.microsoft.com/zh-cn/dotnet/articles/core/app-types>

so,既然是用于生产环境的，当然我们要追求更高的性能。

安装CentOS7

这个就不细说了，网上教程很多，这边我使用了VMWare虚拟了CentOS7。

安装.NET Core SDK for CentOS7。

sudo yum install libunwind libicu（安装libicu依赖）

```
已安装:
libicu.x86_64 0:50.1.2-15.el7          libunwind.x86_64 2:1.1-5.el7_2.2

完毕!
[root@bogon ~]#
```

curl -sSL -o dotnet.tar.gz <https://go.microsoft.com/fwlink/?linkid=843449> (下载sdk压缩包)

sudo mkdir -p /opt/dotnet && sudo tar xzf dotnet.tar.gz -C /opt/dotnet (解压缩)

sudo ln -s /opt/dotnet/dotnet /usr/local/bin (创建链接)

```
[root@bogon ~]# curl -sSL -o dotnet.tar.gz https://go.microsoft.com/fwlink/?LinkID=809131
[root@bogon ~]# sudo mkdir -p /opt/dotnet && sudo tar xzf dotnet.tar.gz -C /opt/dotnet
[root@bogon ~]# sudo ln -s /opt/dotnet/dotnet /usr/local/bin
[root@bogon ~]#
```

输入 **dotnet --info** 来查看是否安装成功

```
[root@bogon ~]# dotnet --info
.NET Command Line Tools (1.0.0-preview2-003121)

Product Information:
  Version:           1.0.0-preview2-003121
  Commit SHA-1 hash: 1e9d529bc5

Runtime Environment:
  OS Name:            centos
  OS Version:         7
  OS Platform:        Linux
  RID:                centos.7-x64
[root@bogon ~]#
```

如果可以执行则表明.NET Core SDK安装成功。

参考资料: <https://www.microsoft.com/net/core#centos>

部署ASP.NET Core应用程序

上传之前发布的文件夹至/home/wwwroot/WebApplication1。

检查是否能运行

命令: dotnet /home/wwwroot/WebApplication1/WebApplication1.dll

```
Hosting environment: Production
Content root path: /root
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

如果出现这些信息则表示成功运行。

这时候我们是无法访问到这个页面的，这时候我们需要部署一个web容器来进行转发。

安装配置Nginx

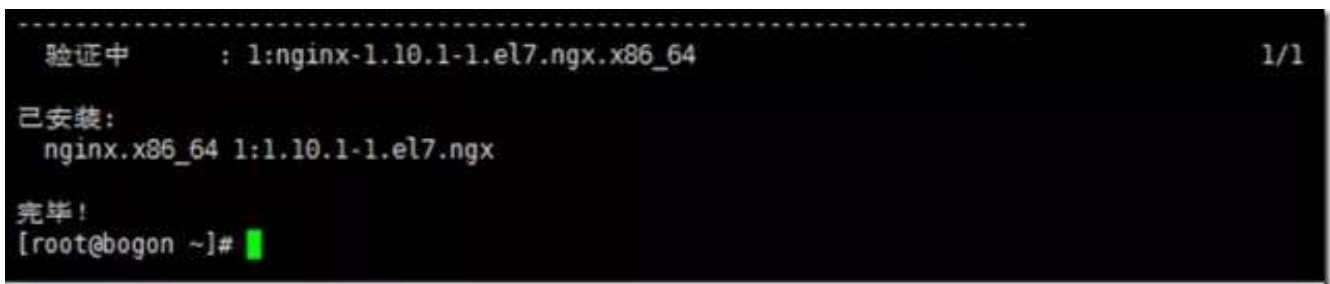
```
curl -o nginx.rpm http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7ngx.noarch.rpm
```



```
[root@bogon ~]# curl -o nginx.rpm http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7ngx.noarch.rpm
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 4680 100 4680    0     0  2446      0  0:00:01  0:00:01 --:--:-- 2446
[root@bogon ~]#
```

```
rpm -ivh nginx.rpm
```

```
yum install nginx
```



```
-----
验证中      : 1:nginx-1.10.1-1.el7ngx.x86_64                                1/1

已安装:
nginx.x86_64 1:1.10.1-1.el7ngx

完毕!
[root@bogon ~]#
```

安装成功！

输入：**systemctl start nginx** 来启动nginx。

输入：**systemctl enable nginx** 来设置nginx的开机启动（linux宕机、重启会自动运行nginx不需要连上去输入命令）。

配置防火墙

命令：**firewall-cmd --zone=public --add-port=80/tcp --permanent**（开放80端口）

命令：**systemctl restart firewalld**（重启防火墙以使配置即时生效）

测试nginx是否可以访问。



配置Nginx对Asp.Net Core应用的转发

修改 `/etc/nginx/conf.d/default.conf` 文件。

将文件内容替换为

```
server {  
    listen 80;  
    location / {  
        proxy_pass http://localhost:5000;  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection keep-alive;  
        proxy_set_header Host $host;  
        proxy_cache_bypass $http_upgrade;  
    }  
}
```

上传至CentOS进行覆盖。

执行：**nginx -s reload** 使其即时生效

运行ASP.NET Core应用程序

```
Hosting environment: Production  
Content root path: /home/wwwroot  
Now listening on: http://localhost:5000  
Application started. Press Ctrl+C to shut down.  
█
```

命令：**dotnet /home/wwwroot/WebApplication1/WebApplication1.dll**

这时候再次尝试访问。



想哭的心都有。。。经过后续了解，这个问题是由于SELinux保护机制所导致，我们需要将nginx添加至SELinux的白名单。

接下来我们通过一些命令解决这个问题。。

```
yum install policycoreutils-python
```

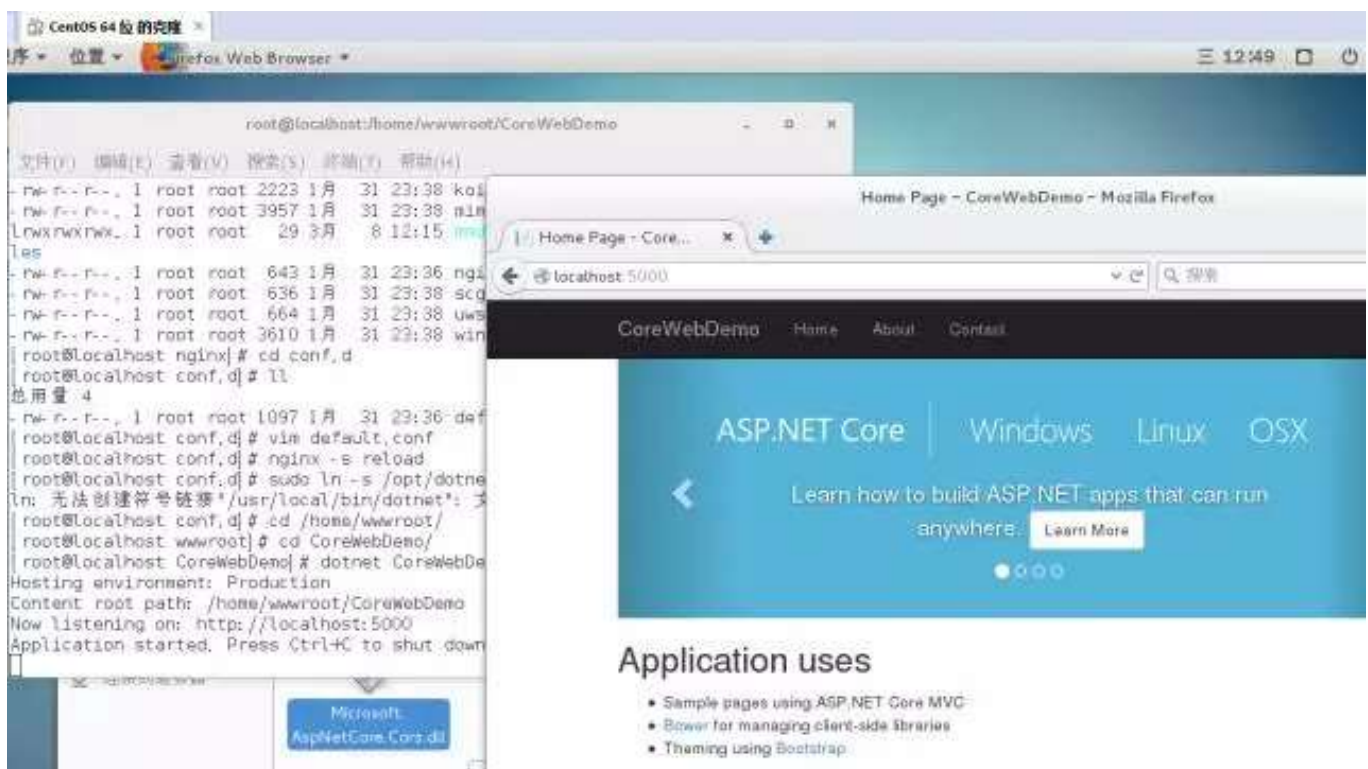
```
sudo cat /var/log/audit/audit.log | grep nginx | grep denied | audit2allow -M mynginx
```

```
sudo semodule -i mynginx.pp
```

```
完毕！
[root@bogon ~]# sudo cat /var/log/audit/audit.log | grep nginx | grep denied | audit2allow
-M mynginx
***** IMPORTANT *****
To make this policy package active, execute:
semodule -i mynginx.pp

[root@bogon ~]# sudo semodule -i mynginx.pp
[root@bogon ~]#
```

再次尝试访问。



至此基本完成了部署。

配置守护服务

目前存在三个问题

问题1：ASP.NET Core应用程序运行在shell之中，如果关闭shell则会发现ASP.NET Core应用被关闭，从而导致应用无法访问，这种情况当然是我们不想遇到的，而且生产环境对这种情况是零容忍的。

问题2：如果ASP.NET Core进程意外终止那么需要人为连进shell进行再次启动，往往这种操作都不够及时。

问题3：如果服务器宕机或需要重启我们则还是需要连入shell进行启动。

为了解决这个问题，我们需要有一个程序来监听ASP.NET Core 应用程序的状况。在应用程序停止运行的时候立即重新启动。这边我们用到了Supervisor这个工具，Supervisor使用Python开发的。

安装Supervisor

```
yum install python-setuptools
```

```
easy_install supervisor
```

配置Supervisor

```
mkdir /etc/supervisor  
echo_supervisord_conf > /etc/supervisor/supervisord.conf
```

修改supervisord.conf文件，将文件尾部的配置

```
;  
[include]  
;files = relative/directory/*.ini  
|
```

修改为

```
[include]  
files = conf.d/*.conf  
|
```

ps:如果服务已启动，修改配置文件可用“supervisorctl reload”命令来使其生效

配置对Asp.Net Core应用的守护

创建一个 WebApplication1.conf文件，内容大致如下

```
[program:WebApplication1]  
command=dotnet WebApplication1.dll ; 运行程序的命令  
directory=/home/wwwroot/WebApplication1/ ; 命令执行的目录
```

```
autorestart=true ; 程序意外退出是否自动重启
stderr_logfile=/var/log/WebApplication1.err.log ; 错误日志文件
stdout_logfile=/var/log/WebApplication1.out.log ; 输出日志文件
environment=ASPNETCORE_ENVIRONMENT=Production ; 进程环境变量
user=root ; 进程执行的用户身份
stopsignal=INT
```

将文件拷贝至：“/etc/supervisor/conf.d/WebApplication1.conf” 下

运行supervisord，查看是否生效

```
supervisord -c /etc/supervisor/supervisord.conf
ps -ef | grep WebApplication1
```

```
[root@bogon ~]# supervisord -c /etc/supervisor/supervisord.conf
[root@bogon ~]# ps -ef | grep WebApplication1
root      2471   2470   1 11:46 ?        00:00:00 dotnet WebApplication1.dll
root      2484   2148   0 11:46 pts/0    00:00:00 grep --color=auto WebApplication1
[root@bogon ~]#
```

如果存在dotnet WebApplication1.dll 进程则代表运行成功，这时候在使用浏览器进行访问。



至此关于ASP.NET Core应用程序的守护即配置完成。

配置Supervisor开机启动

新建一个“supervisord.service” 文件

```
# dservice for systemd (CentOS 7.0+)
# by ET-CS (https://github.com/ET-CS).
[Unit]
Description=Supervisor daemon

[Service]
Type=forking
ExecStart=/usr/bin/supervisord -c /etc/supervisor/supervisord.conf
ExecStop=/usr/bin/supervisorctl shutdown
```



```
ExecReload=/usr/bin/supervisorctl reload
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target
```

将文件拷贝至: “/usr/lib/systemd/system/supervisord.service”

执行命令: `systemctl enable supervisord`

```
[root@bogon ~]# systemctl enable supervisord
Created symlink from /etc/systemd/system/multi-user.target.wants/supervisord.service to /usr/lib/systemd/system/supervisord.service.
```

执行命令: `systemctl is-enabled supervisord` #来验证是否为开机启动

```
[root@bogon ~]# systemctl is-enabled supervisord
enabled
```



[Read more](#)