Reference URL’s:

<https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?view=aspnetcore-2.1&tabs=aspnetcore2x>

<https://kimsereyblog.blogspot.com/2018/06/asp-net-core-with-nginx.html>

# **Host ASP.NET Core on Linux with Nginx**

This guide:

* Places an existing ASP.NET Core app behind a reverse proxy server.
* Sets up the reverse proxy server to forward requests to the Kestrel web server.
* Ensures the web app runs on startup as a daemon.
* Configures a process management tool to help restart the web app.

## **Prerequisites**

1. Access to an Ubuntu 16.04/18.04 server with a standard user account with sudo privilege.
2. Install the .NET Core runtime on the server.
   1. Visit the [.NET Core All Downloads page](https://www.microsoft.com/net/download/all).
   2. Select the latest non-preview runtime from the list under **Runtime**.
   3. Select and follow the instructions for Ubuntu that match the Ubuntu version of the server.
3. An existing ASP.NET Core app.

## **Publish and over the app**

## Add kestrel enpoints in appsettings.json file before publish.

"Kestrel": {

"Endpoints": {

"Http": {

"Url": "http://localhost:6015"

}

}

}

Also make sure you have either removed all codes specific to “Development” environment or put that code under development or debug logic before publishing your app for production release.

Run [dotnet publish](https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-publish) from the development environment to package an app into a directory (for example, **bin/Release/<target\_framework\_moniker>/publish**) that can run on the server:

console

dotnet publish --configuration Release

the ASP.NET Core app to the server using a tool that integrates into the organization's workflow (for example, ***SCP, SFTP***). It's common to locate web apps under the **var** directory (for example, ***var/aspnetcore/dotnetcore-webapi***).

**Note**

*Under a production deployment scenario, a continuous integration workflow does the work of publishing the app and ing the assets to the server.*

Test the app:

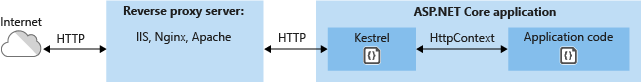
1. From the command line, run the app: ***dotnet <app\_assembly>.dll***.
2. In a browser, navigate to ***http://<serveraddress>:<port>*** to verify the app works on Linux locally.

## **Configure a reverse proxy server**

A reverse proxy is a common setup for serving dynamic web apps. A reverse proxy terminates the HTTP request and forwards it to the ASP.NET Core app.

**Note**

Either configuration—with or without a reverse proxy server—is a valid and supported hosting configuration for ASP.NET Core 2.0 or later apps. For more information, see **[When to use Kestrel with a reverse proxy](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-2.1" \l "when-to-use-kestrel-with-a-reverse-proxy)**.



### **Use a reverse proxy server**

Kestrel is great for serving dynamic content from ASP.NET Core. However, the web serving capabilities aren't as feature rich as servers such as IIS, Apache, or Nginx. A reverse proxy server can offload work such as serving static content, caching requests, compressing requests, and SSL termination from the HTTP server. A reverse proxy server may reside on a dedicated machine or may be deployed alongside an HTTP server.

For the purposes of this guide, a single instance of Nginx is used. It runs on the same server, alongside the HTTP server. Based on requirements, a different setup may be chosen.

Because requests are forwarded by reverse proxy, use the [***Forwarded Headers Middleware***](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/proxy-load-balancer?view=aspnetcore-2.1) from the [***Microsoft.AspNetCore.HttpOverrides***](https://www.nuget.org/packages/Microsoft.AspNetCore.HttpOverrides/) package. The middleware updates the ***Request.Scheme, using the X-Forwarded-Proto*** header, so that redirect URIs and other security policies work correctly.

C# – Startup.cs file under Configure() function

app.UseForwardedHeaders(new ForwardedHeadersOptions

{

ForwardedHeaders = ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto

});

//app.UseAuthentication();

### **Install Nginx**

Use ***apt-get*** to install Nginx. The installer creates a **systemd** init script that runs Nginx as daemon on system startup.

bash

sudo -s

nginx=stable # use nginx=development for latest development version

add-apt-repository ppa:nginx/$nginx

apt-get update

apt-get install nginx

Since Nginx was installed for the first time, explicitly start it by running:

bash

sudo service nginx start

Verify a browser displays the default landing page for Nginx. The landing page is reachable at ***http://<server\_IP\_address>/index.nginx-debian.html***.

### **Configure Nginx**

To configure Nginx as a reverse proxy to forward requests to your ASP.NET Core app, modify ***/etc/nginx/sites-available/mytestwebsite.com***. Open it in a text editor, and replace the contents with the following:

nginx

server {

listen 610;

listen [::]:610;

server\_name localhost 127.0.0.1;

location / {

proxy\_pass http://localhost:6010;

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;

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

proxy\_set\_header X-Forwarded-Proto $scheme;

}

}

Once the Nginx configuration is established, run **sudo nginx -t** to verify the syntax of the configuration files. If the configuration file test is successful, force Nginx to pick up the changes by running ***sudo nginx -s reload***.

Those configurations will not yet be used by nginx. We need to enable them by having them in the sites-enablefolder. This is achieved by creating a symlink, if we navigate to /etc/nginx/sites-enabled, we can find the symlink of default.

ln -s /etc/nginx/sites-available/mytestwebsite.com /etc/nginx/sites-enabled/mytestwebsite.com

To directly run the app on the server:

1. Navigate to the app's directory.
2. Run the app's executable: ***dotnet ./<app\_executable>***.

If a permissions error occurs, change the permissions:

console

chmod u+x <app\_executable>

If the app runs on the server but fails to respond over the Internet, check the server's firewall and confirm that port 80 is open. If using an Azure Ubuntu VM, add a Network Security Group (NSG) rule that enables inbound port 80 traffic. There's no need to enable an outbound port 80 rule, as the outbound traffic is automatically granted when the inbound rule is enabled.

When done testing the app, shut the app down with **Ctrl+C** at the command prompt.

## **Monitoring the app**

The server is setup to forward requests made to ***http://<serveraddress>:610*** on to the ASP.NET Core app running on Kestrel at ***http://127.0.0.1:6010***. However, Nginx isn't set up to manage the Kestrel process. **systemd** can be used to create a service file to start and monitor the underlying web app. **systemd** is an init system that provides many powerful features for starting, stopping, and managing processes.

### **Create the service file**

Create the service definition file:

bash

sudo nano /etc/systemd/system/kestrel-dotnetcore-webapi.service

The following is an example service file for the app:

ini

[Unit]

Description=Example .NET Web API App running on Ubuntu

[Service]

WorkingDirectory=/var/aspnetcore/dotnetcore-webapi

ExecStart=/usr/bin/dotnet /var/aspnetcore/dotnetcore-webapi/dotnetcore-webapi.dll

Restart=always

# Restart service after 10 seconds if the dotnet service crashes:

RestartSec=10

SyslogIdentifier=dotnetcore-webapi

User=www-data

Environment=ASPNETCORE\_ENVIRONMENT=Production

Environment=DOTNET\_PRINT\_TELEMETRY\_MESSAGE=false

[Install]

WantedBy=multi-user.target

If the user ***www-data*** isn't used by the configuration, the user defined here must be created first and given proper ownership for files.

Linux has a case-sensitive file system. Setting ***ASPNETCORE\_ENVIRONMENT*** to "***Production***" results in searching for the configuration file ***appsettings.Production.json, not appsettings.production.json***.

Save the file and enable the service.

bash

systemctl enable kestrel-dotnetcore-webapi.service

Start the service and verify that it's running.

systemctl start kestrel-dotnetcore-webapi.service

systemctl status kestrel-dotnetcore-webapi.service

● kestrel-dotnetcore-webapi.service - Example .NET Web API App running on Ubuntu

Loaded: loaded (/etc/systemd/system/kestrel-dotnetcore-webapi.service; enabled)

Active: active (running) since Thu 2016-10-18 04:09:35 NZDT; 35s ago

Main PID: 9021 (dotnet)

CGroup: /system.slice/kestrel-dotnetcore-webapi.service

└─9021 /usr/local/bin/dotnet /var/aspnetcore/dotnetcore-webapi/dotnetcore-webapi.dll

With the reverse proxy configured and Kestrel managed through systemd, the web app is fully configured and can be accessed from a browser on the local machine at http://localhost. It's also accessible from a remote machine, barring any firewall that might be blocking.

Inspecting the response headers, the Server header shows the ASP.NET Core app being served by Nginx.

**Connection:**

keep-alive

**Content-Type:**

application/json; charset=utf-8

**Date:**

Fri, 14 Sep 2018 19:55:31 GMT

**Server:**

nginx/1.14.0

**Transfer-Encoding:**

chunked

### **Viewing logs**

Since the web app using Kestrel is managed using ***systemd***, all events and processes are logged to a centralized journal. However, this journal includes all entries for all services and processes managed by ***systemd***. To view the ***kestrel-dotnetcore-webapi.service-specific*** items, use the following command:

bash

sudo journalctl -fu kestrel-dotnetcore-webapi.service

For further filtering, time options such as --since today, --until 1 hour ago or a combination of these can reduce the amount of entries returned.

bash

sudo journalctl -fu kestrel-dotnetcore-webapi.service --since "2016-10-18" --until "2016-10-18 04:00"