Installing ASP.NET Core 2.1 on Ubuntu 18.4 Linux

Daniel Opitz
17 Jul 2018

The LAMA stack - Linux, Apache, MySQL, ASP.NET Core

Table of contents

Requirements

- Ubuntu 18.04
- root permissions

Register Microsoft key and feed

Before installing .NET, you'll need to register the Microsoft key, register the product repository, and install required dependencies. This only needs to be done once per machine.

Open a command prompt and run the following commands:

```
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft
sudo mv microsoft.asc.gpg /etc/apt/trusted.gpg.d/
wget -q https://packages.microsoft.com/config/ubuntu/18.04/prod.list
sudo mv prod.list /etc/apt/sources.list.d/microsoft-prod.list
```

```
sudo chown root:root /etc/apt/trusted.gpg.d/microsoft.asc.gpg
sudo chown root:root /etc/apt/sources.list.d/microsoft-prod.list
```

Install the .NET Core 2.1 Runtime

This commands will install the .NET Core Hosting Bundle, which includes the .NET Core runtime and the ASP.NET Core runtime. To install just the .NET Core runtime, use the `dotnet-runtime-2.1` package.

In your command prompt, run the following commands:

```
sudo apt-get install apt-transport-https
sudo apt-get update
sudo apt-get install aspnetcore-runtime-2.1
```

Source: https://www.microsoft.com/net/download/linux-package-manager/ubuntu18-04/runtime-2.1.0

Install the .NET SDK

In your command prompt, run the following commands:

```
sudo apt-get install dotnet-sdk-2.1
```

> **Notice:** It is not necessary to install the .NET SDK on a productive server. You only need to install the SDK in the development environment.

Source: https://www.microsoft.com/net/learn/get-started/linux/ubuntu18-04

Test the installation

To make sure it works after you've set it up, just run:

```
dotnet --info
```

Expected output:

```
.NET Core SDK (reflecting any global.json):
 Version:   2.1.302
 Commit:    9048955601

Runtime Environment:
 OS Name:     ubuntu
 OS Version:  18.04
 OS Platform: Linux
 RID:         ubuntu.18.04-x64
```

```
 Base Path:   /usr/share/dotnet/sdk/2.1.302/

Host (useful for support):
  Version: 2.1.2
  Commit:  811c3ce6c0

.NET Core SDKs installed:
  2.1.302 [/usr/share/dotnet/sdk]

.NET Core runtimes installed:
  Microsoft.AspNetCore.All 2.1.2 [/usr/share/dotnet/shared/Microsoft.AspNetCore.All]
  Microsoft.AspNetCore.App 2.1.2 [/usr/share/dotnet/shared/Microsoft.AspNetCore.App]
  Microsoft.NETCore.App 2.1.2 [/usr/share/dotnet/shared/Microsoft.NETCore.App]

To install additional .NET Core runtimes or SDKs:
  https://aka.ms/dotnet-download
```

Create a new ASP.NET Core web application

Let's create our first ASP.NET Core "hello world" web app. Just run:

```
cd ~
sudo dotnet new web -o hellomvc
cd ~/hellomvc/
sudo dotnet run
```

The `dotnet run` command should start the webserver on a new open port (5001 for https and 5000 for http).

Expected output:

```
Using launch settings from /home/user/hellomvc/Properties/launchSettings.json...
Hosting environment: Development
Content root path: /home/user/hellomvc
Now listening on: https://localhost:5001
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Now open your browser with the url you can see in the console output:

- For example: http://localhost:5000 or https://localhost:5001
- You should see a message like: `Hello World!`

Change the listening address via command line

The following command allows access to the kestrel webserver from anywhere. This parameter is usful to access the website from another host via docker or VirtualBox.

```
sudo dotnet run --urls "http://*:5000;https://*:5001"
```

Please also check the current firewall status with `sudo ufw status`

Publish and copy over the app

Run `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:

```
cd ~/hellomvc
sudo dotnet publish --configuration Release
```

Copy 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/hellomvc`).

```
sudo mkdir /var/aspnetcore/
cd /var/aspnetcore/
```

Note: In a production deployment scenario, a continuous integration workflow does the work of publishing the app and copying the assets to the server.

Test the app

- From the command line, run the app: `dotnet <app_assembly>.dll`.
- In a browser, navigate to `http://<serveraddress>:<port>` to verify the app works on Linux locally.

Install MySQL

For our LAMA stack we need MySQL:

```
sudo apt-get update
sudo apt-get install mysql-server mysql-client libmysqlclient-dev -y
```

Install Apache as reverse proxy

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

Your application could technically run only with the Kestrel web server and without a reverse proxy. But if you want to install and manage your own SSL (https://) certifactes you should use a reverse proxy server (Nginx, Apache, IIS).

Update Ubuntu packages to their latest stable versions:

```
sudo apt-get update
```

Install vim

```
sudo apt-get install vim -y
```

Install the Apache web server on Ubuntu with a single command:

```
sudo apt-get install apache2 -y
```

Enable the required apache modules:

```
sudo a2enmod rewrite
sudo a2enmod proxy
sudo a2enmod proxy_http
sudo a2enmod headers
sudo a2enmod ssl
sudo service apache2 restart
```

Configure SSL

You only need to set up a virtual host for TLS/SSL. The following example assumes that an SSL certificate is located at `/etc/ssl/certs/localhost.crt` and an associated key is located at `/etc/ssl/private/localhost.key`.

> **Security warning** For demonstration purposes I create a self-signed certificate here. If you are using a real certificate (like https://letsencrypt.org/), the browser should not complain about the certificate. Don't use a self-signed certificate in your production environment!

Create and install an Apache self signed certificate:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:4096 -keyout /etc/ssl/private/local
```

Fill out the prompts appropriately. The most important line is the one that requests the Common Name (e.g. server FQDN or YOUR name). You need to enter the domain name associated with your server or, more likely, your server's public IP address.

Output

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:New York
Locality Name (eg, city) []:New York City
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Bouncy Castles, Inc.
Organizational Unit Name (eg, section) []:Ministry of Water Slides
```

```
Common Name (e.g. server FQDN or YOUR name) []:server_IP_address
Email Address []:admin@your_domain.com
```

More infos: How To Create a Self-Signed SSL Certificate for Apache in Ubuntu

Configure Apache

Add a server's fully qualified domain name:

```
sudo vim /etc/apache2/apache2.conf
```

Add the following line to apache2.conf:

```
ServerName localhost
```

Configuration website files for Apache are located within the `/etc/apache2/sites-available/` directory. Any file with the `.conf` extension is processed in alphabetical order.

Create a configuration file, named `/etc/apache2/sites-available/hellomvc.conf` , for the app:

```
cd /etc/apache2/sites-available/
sudo vim hellomvc.conf
```

Copy and save this content to `hellomvc.conf`

```
<VirtualHost *:*>
    RequestHeader set "X-Forwarded-Proto" expr=%{REQUEST_SCHEME}
</VirtualHost>

<VirtualHost *:80>
    RewriteEngine On
    RewriteCond %{HTTPS} !=on
    RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
    ProxyPreserveHost On
    ProxyPass / http://127.0.0.1:5000/
    ProxyPassReverse / http://127.0.0.1:5000/
    ErrorLog /var/log/apache2/hellomvc-error.log
    CustomLog /var/log/apache2/hellomvc-access.log common
    SSLEngine on
    SSLProtocol all -SSLv2
    SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:!RC4+RSA:+HIGH:+MEDIUM:!LOW:!RC4
    SSLCertificateFile /etc/ssl/private/localhost.crt
    SSLCertificateKeyFile /etc/ssl/private/localhost.key
</VirtualHost>
```

See Name-based virtual host support for more information. Requests are proxied at the root to port `5000` of the server at `127.0.0.1`.

For bi-directional communication, `ProxyPass` and `ProxyPassReverse` are required.

To change Kestrel's IP/port, see Kestrel: Endpoint configuration.

Save the file and test the configuration.

Disable the `000-default` site:

```
sudo a2dissite 000-default.conf
```

Activate the `hellomvc` site:

```
sudo a2ensite hellomvc.conf
```

Test the configuration. If everything passes, the result should be `OK`.

```
sudo apachectl configtest
```

Restart Apache:

```
sudo service apache2 restart
```

Start your dotnet application:

```
sudo dotnet run
```

If you open http://localhost/ you should be redirected to https://localhost/.

After you have accepted the self-signed certificate you should see the `Hello World!` message.

More reverse proxy setup informations

- When to use Kestrel with a reverse proxy
- Host ASP.NET Core on Linux with Apache
- Host ASP.NET Core on Linux with Nginx

Monitoring the app

The website isn't up and running on localhost:5000 yet (unless you've run it yourself and kept it running!). So we'll need an service to start it and keep it running.

There's an tool called Supervisor that is good at that so I'll add it.

```
sudo apt-get install supervisor
```

Now publish the application into a final location. Run `dotnet publish`, then copy the content of `publish/` into `/var/aspnetcore` where it will go live.

Build the application:

```
cd ~/hellomvc
sudo dotnet publish --configuration Release
```

Copy the release build into to application directory:

```
sudo mkdir /var/aspnetcore/
sudo cp -R ~/hellomvc/bin/Release/netcoreapp2.1/publish/* /var/aspnetcore/
```

Show me the application files:

```
cd /var/aspnetcore/
ls
```

Create a file `/etc/supervisor/conf.d/hellomvc.conf`

```
sudo vim /etc/supervisor/conf.d/hellomvc.conf
```

Add this content to `hellomvc.conf`

```
[program:dotnettest]
command=/usr/bin/dotnet /var/aspnetcore/hellomvc.dll --urls "http://*:5000"
directory=/var/aspnetcore/
autostart=true
autorestart=true
stderr_logfile=/var/log/hellomvc.err.log
stdout_logfile=/var/log/hellomvc.out.log
environment=ASPNETCORE_ENVIRONMENT=Production
user=www-data
stopsignal=INT
```

Now we start and stop Supervisor and watch/tail its logs to see our app startup!

```
sudo service supervisor stop
sudo service supervisor start
sudo tail -f /var/log/supervisor/supervisord.log
# and the application logs if you like
sudo tail -f /var/log/hellomvc.out.log
```

Output

```
2018-07-13 12:15:08,102 INFO supervisord started with pid 9314
2018-07-13 12:15:09,105 INFO spawned: 'dotnettest' with pid 9317
2018-07-13 12:15:10,336 INFO success: dotnettest entered RUNNING state, process has sta
```

Conclusion

Remember the relationships.

- Linux - Your server
- Apache - Listens on Port 443 / 80 and forwards HTTP/S calls to your website
- MySQL - Keeps your app data
- ASP.NET Core - Runs your app / website
- Supervisor - Keeps your app running