UNIT 6
Hosting and Deploying ASP.NET Core Application

(5 Hours) — Advanced .NET Technology (CACS408)

Note: This PDF may contain errors — please cross-check important points with your syllabus, prescribed textbook and teacher's notes before the exam.

Note: Examples use net8.0 and the 8.0 Docker images. If your installed SDK is newer (for example .NET 10), change the version numbers accordingly — the steps stay the same. Replace MyApp with your project name.

6.1 App Servers and Hosting Models

An ASP.NET Core app is a console application that contains its own web server. When it starts, it listens on a port and handles HTTP requests. The built-in cross-platform server is **Kestrel**. Hosting means running that app on a machine so users can reach it.

Server	Description
Kestrel	Default, fast, cross-platform web server inside every ASP.NET Core app. Can be used directly or behind a proxy.
HTTP.sys	Windows-only server (kernel-mode driver); offers Windows authentication, port sharing and kernel caching without IIS.
IIS	Windows web server that works with ASP.NET Core through the ASP.NET Core Module.
Nginx / Apache	Linux/macOS web servers used as **reverse proxies** in front of Kestrel.

Why use a reverse proxy?

Kestrel can face the internet, but a **reverse proxy** adds features and protection: TLS termination, load balancing, serving static files, request filtering, compression, logging, hosting several apps on the same IP/port 80/443, and restarting the app process if it fails.

bcalink.com

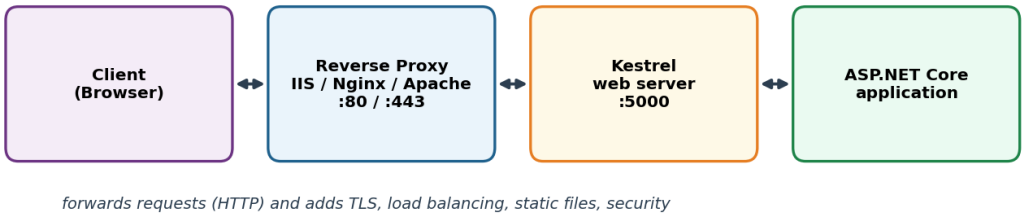


Fig 6.1: Reverse proxy hosting model

Hosting model	Meaning
Self-hosted (Kestrel only)	Run the app directly (dotnet MyApp.dll); good for development, containers and internal services.
Reverse-proxy hosted	Nginx, Apache or IIS forwards requests to Kestrel; recommended for production.
In-process (IIS only)	App runs inside the IIS worker process (w3wp.exe); highest performance.
Out-of-process	App runs in its own dotnet process; IIS acts as a proxy.
Container / cloud hosted	App packaged in Docker or on Azure App Service etc.

bcalink.com

6.2 IIS and the ASP.NET Core Module

The **ASP.NET Core Module (ANCM)** is a native IIS module that connects IIS to the ASP.NET Core app. It starts the app, restarts it if it crashes and forwards requests. It is installed together with the **.NET Hosting Bundle** (runtime + module). IIS does not load the .NET CLR itself, so the application pool must be set to **No Managed Code**.

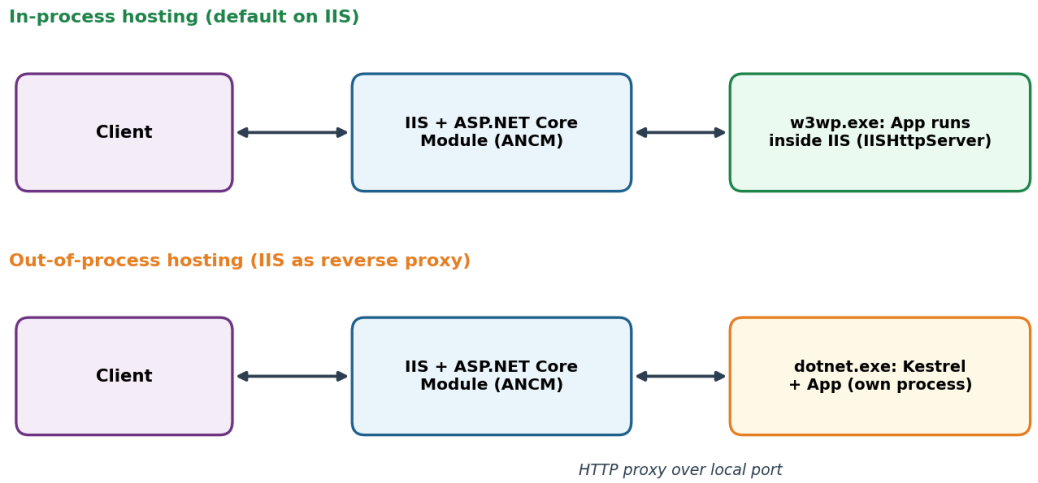


Fig 6.2: In-process and out-of-process hosting with IIS

Point	In-process	Out-of-process
Where the app runs	Inside w3wp.exe (IIS worker)	Separate dotnet.exe process
Web server used	IIS HTTP Server (IISHttpServer)	Kestrel
Performance	Higher — no proxy hop	Slightly lower — loopback HTTP proxy
Setting	`<AspNetCoreHostingModel>InProcess` (default)	`<AspNetCoreHostingModel>OutOfProcess`

Steps to deploy on IIS

- Install the IIS role (Windows features) and the **.NET Hosting Bundle**; restart IIS (`iisreset`).
- Publish the app: `dotnet publish -c Release -o C:\inetpub\MyApp`.
- In IIS Manager create an **Application Pool** with .NET CLR version = **No Managed Code**.
- Add a **Web Site** pointing to the publish folder, assign the pool, set bindings (host name, port, HTTPS certificate).
- Give the pool identity (IIS AppPool\MyApp) read access to the folder, then browse the site.

```
<!-- web.config (generated automatically by dotnet publish) -->
<configuration>
  <system.webServer>
    <handlers>
      <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModuleV2"
           resourceType="Unspecified" />
    </handlers>
    <aspNetCore processPath="dotnet" arguments=".\MyApp.dll"
                stdoutLogEnabled="false" hostingModel="inprocess" />
  </system.webServer>
</configuration>
```

web.config used by the ASP.NET Core Module

pclink.com

6.3 Hosting on Linux with Nginx

On Linux, Kestrel runs as a background service managed by **systemd**, and **Nginx** is configured as the reverse proxy.

- Publish and copy the app to the server, e.g. `/var/www/myapp`.
- Create a systemd service so the app starts on boot and restarts on failure.
- Configure an Nginx server block that forwards requests to Kestrel (port 5000).
- Reload Nginx; open ports 80/443 in the firewall.

```
# /etc/systemd/system/kestrel-myapp.service
[Unit]
Description=My ASP.NET Core App

[Service]
WorkingDirectory=/var/www/myapp
ExecStart=/usr/bin/dotnet /var/www/myapp/MyApp.dll
Restart=always
RestartSec=10
User=www-data
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=ASPNETCORE_URLS=http://localhost:5000

[Install]
WantedBy=multi-user.target

# commands
sudo systemctl enable kestrel-myapp.service
sudo systemctl start kestrel-myapp.service
sudo systemctl status kestrel-myapp.service
```

Program 6.1: systemd service for Kestrel

```
# /etc/nginx/sites-available/myapp
server {
    listen 80;
    server_name example.com *.example.com;
    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_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }
}

# test and reload
sudo nginx -t
sudo nginx -s reload
```

bcalink.com

Program 6.2: Nginx reverse-proxy configuration

Because the proxy changes the apparent client IP and scheme, add the forwarded headers middleware in `Program.cs`:

```
app.UseForwardedHeaders(new ForwardedHeadersOptions
{
    ForwardedHeaders = ForwardedHeaders.XForwardedFor |
                       ForwardedHeaders.XForwardedProto
});
```

Program 6.3: Forwarded headers middleware

6.4 Hosting with Apache

Apache can also be a reverse proxy using modules **mod_proxy**, **mod_proxy_http**, **mod_headers** and **mod_rewrite** (and mod_ssl for HTTPS). The app itself still runs on Kestrel under systemd as above.

```
# enable modules (Debian/Ubuntu)
sudo a2enmod proxy proxy_http proxy_wstunnel headers rewrite ssl

# /etc/apache2/sites-available/myapp.conf
<VirtualHost *:80>
    ServerName www.example.com
    ProxyPreserveHost On
    ProxyPass        / http://127.0.0.1:5000/
    ProxyPassReverse / http://127.0.0.1:5000/
    RequestHeader set X-Forwarded-Proto "http"
    ErrorLog  ${APACHE_LOG_DIR}/myapp-error.log
    CustomLog ${APACHE_LOG_DIR}/myapp-access.log common
</VirtualHost>

sudo a2ensite myapp.conf
sudo systemctl reload apache2
```

Program 6.4: Apache reverse-proxy virtual host

6.5 Publishing an ASP.NET Core Application

Publishing compiles the app in Release mode and collects everything needed to run it (DLLs, static files, config) into a folder.

```
dotnet publish -c Release -o ./publish                # framework-dependent
dotnet publish -c Release -r linux-x64 --self-contained true -o ./publish
dotnet publish -c Release -r win-x64 -p:PublishSingleFile=true --self-contained true
dotnet ./publish/MyApp.dll                            # run the published app
```

Common dotnet publish commands

Option	Framework-dependent	Self-contained
Runtime needed on server	Yes — .NET runtime must be installed	No — runtime is bundled
Size	Small	Large
Portability	Any OS with the runtime	Specific runtime identifier (win-x64, linux-x64…)
Updates	Runtime patched centrally	Must republish to update runtime

Note: Use environment variable ASPNETCORE_ENVIRONMENT (Development, Staging, Production) to choose appsettings.{Environment}.json.

bealink.com

6.6 Docker and Containerization

A **container** packages an application with its runtime and dependencies so it runs identically on any machine. Unlike a virtual machine, containers share the host OS kernel, so they are lightweight and start quickly.

Term	Meaning
Image	Read-only template (app + runtime) built from a Dockerfile.
Container	A running instance of an image.
Dockerfile	Text file with the instructions to build an image.
Registry	Storage for images (Docker Hub, Azure Container Registry).
Docker Compose	Runs multi-container apps (web + database) from one YAML file.

Virtual Machine	Container
Includes a full guest OS — GBs in size, minutes to boot	Shares host kernel — MBs, starts in seconds
Stronger isolation, heavier	Lightweight, high density, good for microservices

bcalink.com

```
# Dockerfile (multi-stage build)
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY *.csproj .
RUN dotnet restore
COPY . .
RUN dotnet publish -c Release -o /app/publish

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime
WORKDIR /app
COPY --from=build /app/publish .
EXPOSE 8080
ENTRYPOINT ["dotnet", "MyApp.dll"]
```

Program 6.5: Dockerfile for an ASP.NET Core app

```
docker build -t myapp:1.0 .                    # build the image
docker run -d -p 8080:8080 --name web myapp:1.0   # run container, map ports
docker ps                                      # running containers
docker logs web                                # view output
docker stop web && docker rm web               # stop and remove
docker tag myapp:1.0 myregistry.azurecr.io/myapp:1.0
docker push myregistry.azurecr.io/myapp:1.0    # upload to a registry
```

Common Docker commands

Note: Since .NET 8 the official ASP.NET images listen on port 8080 by default (earlier versions used port 80). -p 8080:8080 maps host port to container port.

6.7 Publishing to the Azure Cloud

Microsoft Azure is a cloud platform. The easiest place to host an ASP.NET Core app is **Azure App Service**, a managed platform (PaaS) that handles servers, patches, scaling, HTTPS and deployment slots for you. Containers can instead run on **Azure Container Apps** or **App Service for Containers** using an image from **Azure Container Registry**.

Method 1: Visual Studio Publish wizard

- Right-click the project → **Publish**.
- Target: **Azure** → **Azure App Service** (Windows or Linux) → sign in with your Azure account.
- Create a new App Service (name, subscription, resource group, hosting plan) or choose an existing one.
- Click **Publish**; Visual Studio builds, uploads and opens the site at *https://<name>.azurewebsites.net*.

Method 2: Azure CLI

```
az login
az group create --name rg-myapp --location southeastasia
az appservice plan create --name plan-myapp --resource-group rg-myapp \
                  --sku B1 --is-linux
az webapp create --name myapp-demo-123 --resource-group rg-myapp \
                  --plan plan-myapp --runtime "DOTNETCORE:8.0"
dotnet publish -c Release -o ./publish
cd publish && zip -r ../app.zip . && cd ..
az webapp deploy --resource-group rg-myapp --name myapp-demo-123 \
                  --src-path app.zip --type zip

# set a connection string / app setting
az webapp config appsettings set -g rg-myapp -n myapp-demo-123 \
                  --settings ASPNETCORE_ENVIRONMENT=Production
```

Program 6.6: Deploying to Azure App Service with the CLI

Other options

- **CI/CD:** GitHub Actions or Azure DevOps pipelines build and deploy automatically on every push.
- **Configuration:** keep secrets in App Service *Configuration* (application settings, connection strings) or Azure Key Vault, not in source code.
- **Scaling and monitoring:** scale up/out manually or automatically; Application Insights shows logs and performance.
- **Deployment slots** (staging/production) allow testing and a zero-downtime swap.

fptlink.com

- Names of web apps must be globally unique; check the current runtime names with `az webapp list-runtimes`.

Key Takeaways — Unit 6

✓ Kestrel is the built-in cross-platform server; HTTP.sys is Windows-only; production usually puts a reverse proxy in front.

✓ IIS uses the ASP.NET Core Module (from the Hosting Bundle): in-process (w3wp.exe) or out-of-process (Kestrel); app pool = No Managed Code.

✓ Linux: run Kestrel as a systemd service and configure Nginx (proxy_pass) or Apache (ProxyPass); add UseForwardedHeaders.

✓ dotnet publish -c Release creates the deployable output; framework-dependent vs self-contained.

✓ Docker: image built from a Dockerfile (multi-stage), run as a container with docker run -p.

✓ Azure App Service: publish with Visual Studio or az webapp create/deploy; keep secrets in app settings or Key Vault.

— End of Unit 6 —

bcalink.com