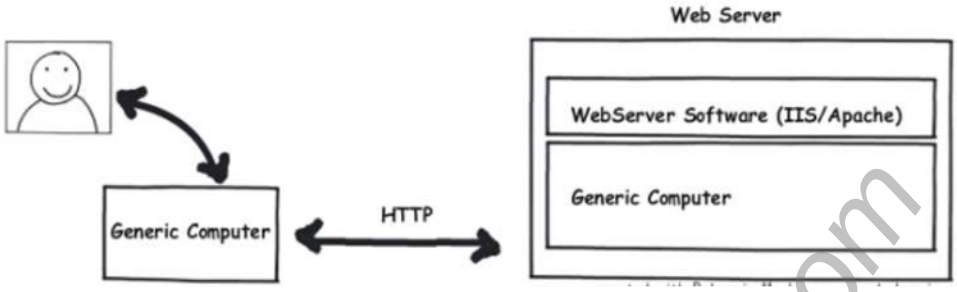
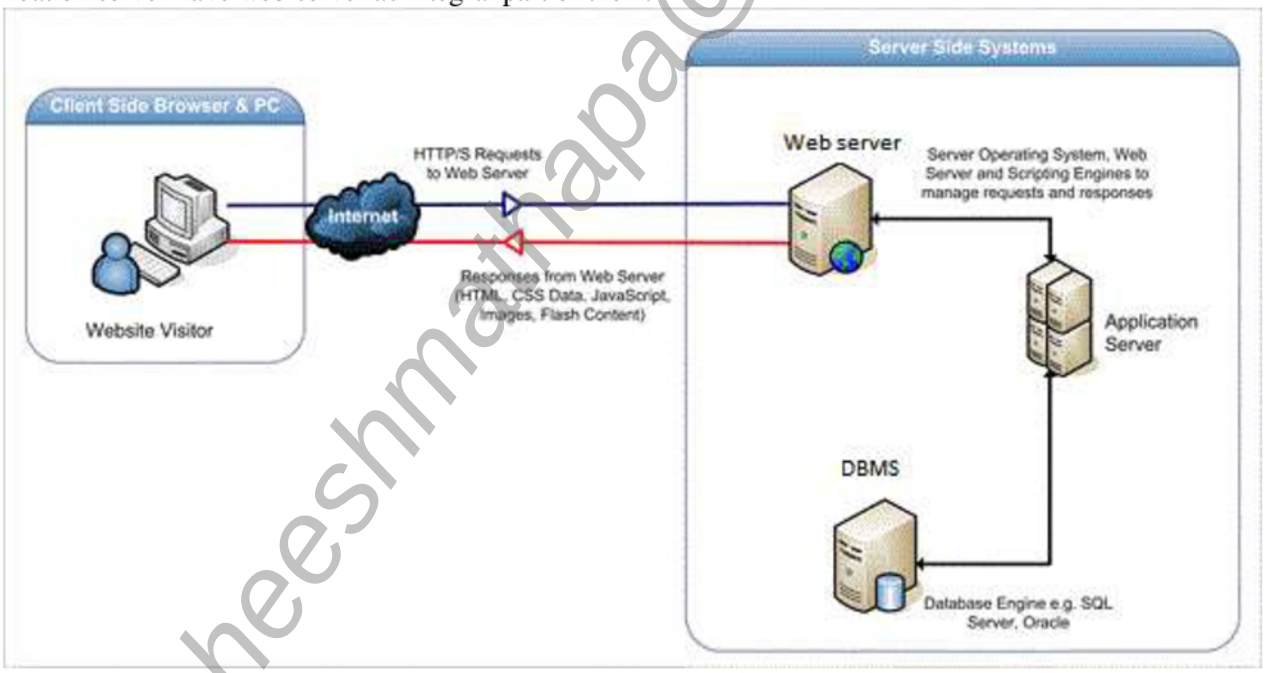
Unit 9
Hosting and Deploying Asp.Net Core Application

Introduction to Web Server and Application Server

- A **Web Server** (like Internet Information(**IIS**) Server, Apache, Nginx) is a piece of software that enables a website to be viewed using **HTTP**. i.e. It allows resources (web pages, images, etc.) to be requested over the HTTP protocol.



  - Internet Information Services server (IIS server) is a Windows Server-based web application used to deliver website content over the internet to an end user.
  - Apache Web Server is an open source software developed and maintained by the Apache Software Foundation. Apache HTTP Server is not a physical server, but rather a software that runs on a server.
  - Nginx is an **open source HTTP Web server and reverse proxy server**. Pronounced as Engine-Ex, Nginx has emerged as the third most popular Web server behind the Apache Web server and Microsoft s IIS, and it currently powers popular websites like Pinterest, WordPress.com, Netflix, Hulu, CloudFlare, Zappos and Zynga.
- **An application server (Like Glass Fish)** is a program which resides on the server side and it's a server programmer providing business logic behind any application. It facilitates the hosting and delivery of high-end consumer or business applications, which are used by multiple and simultaneously connected local or remote users. Most of the application server have web server as integral part of them.



- In the above figure, the client first makes a request, which goes to the web server. The web server then sends it to the middle tier i.e. the application server which further gets the information from 3rd tier (e.g. database server) and sends it back to the web server. The web server further sends back the required information to the client. Different approaches are being utilized for the processing of requests through the web servers and some of them are approaches like JSP (Java server pages), ASP (Active Server Pages), Java Scripts, Java servlets, etc.

Advantage

- It becomes very easy to install applications in one place.
- Changing to any configuration such as moving of Database server, all can be done centrally from one location.
- Patches and security updates are easy to deploy through them.
- It enables the ability to distribute requests to different servers based on their availability. This is done via Load Balancing.
- It provides security to applications.

- It enables fault tolerance with the ability to recover/failover recovery.
- It saves big time if we are required to install a copy of configurations on each machine individually.
- It supports transaction support.
- When it comes to performance, the application server greatly improves application performance as it is based on the client-server model.
- Provides a mechanism for dealing with all the components and running services like session management, synchronous and asynchronous client notifications.

Hosting model: Asp.Net Application vs Asp.Net Core Application

- The hosting model for the previous ASP.NET framework was a relatively complex one, relying on Windows IIS to provide the web server hosting.

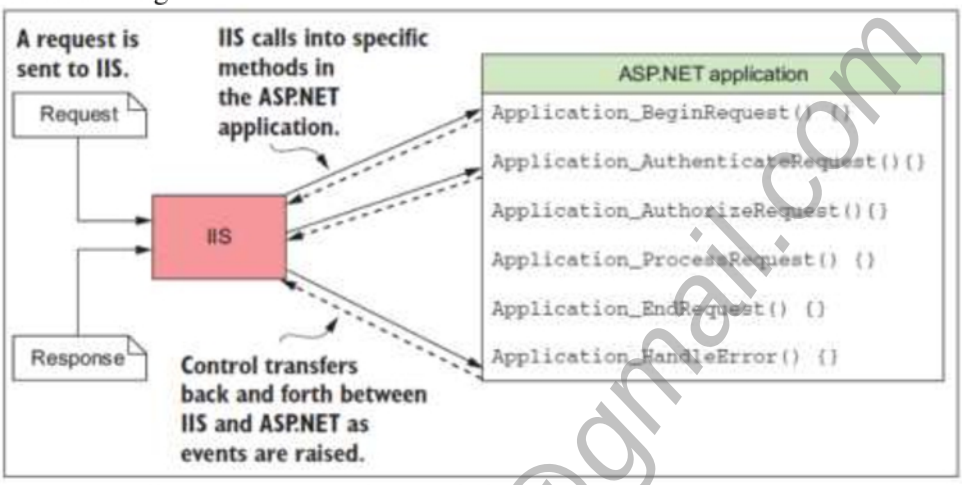


Fig: Hosting Model for ASP.Net (In-Process Hosting)

- With this hosting model, IIS is tightly coupled with the application and everything is hosted inside of an IIS Worker Process (w3wp.exe or iisexpress.exe). Visual Studio uses iisexpress.exe by default to host the application.
- In a cross-platform environment, this kind of interdependent relationship isn't possible, so an alternative hosting model has been adopted, one which separates web applications from the underlying host. This opportunity has led to the development of **Kestrel**: a fast, cross-platform HTTP server on which ASP.NET Core can run.
- Instead of the previous design, whereby IIS calls into specific points of your application, ASP.NET Core applications are console applications that self-host a web server and handle requests directly, as shown in figure below

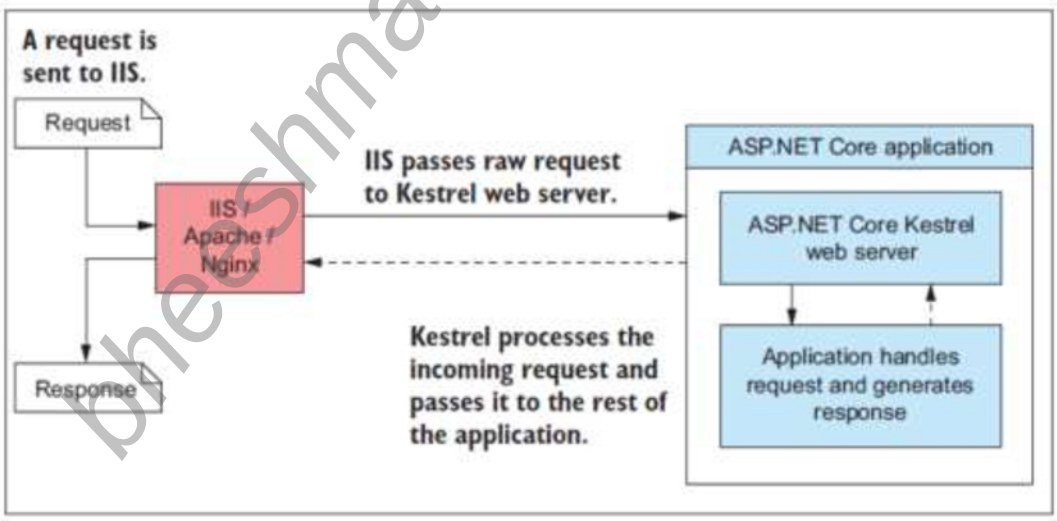


Fig: Hosting Model for ASP.NET Core (Out-of-Process Hosting)

- ASP.NET Core which doesn't **run in-process** to the IIS worker process, but rather runs as a separate, **out of process** Console application that runs its own Web server using the **Kestrel component.**
- With this hosting model, IIS hands off the request to a self-hosted web server (Reverse Proxy Server) in the ASP.NET Core application and receives the response, but has no deeper knowledge of the application.
- A request is received from a browser at the reverse proxy, which passes the request to the ASP.NET Core application, which runs a self-hosted web server.

- The web server processes the request and passes it to the body of the application, which generates a response and returns it to the web server.
- The web server relays this to the reverse proxy, which sends the response to the browser.
- A reverse proxy is software responsible for receiving requests and forwarding them to the appropriate web server. The reverse proxy is exposed directly to the internet, whereas the underlying web server is exposed only to the proxy. This setup has several benefits, primarily security and performance for the web servers.
- A reverse-proxy server captures the request, before passing it to your application. In Windows, the reverse-proxy server will typically be IIS, and on Linux or macOS it might be NGINX or Apache. The same application can run behind various reverse proxies without modification.
- Implementing two web servers i.e. Reverse Proxy/External server and Core/Internal/Kestrel Web Server, while requesting in ASP.NET core application provides various advantage.
  1. The first one is the decoupling of your application from the underlying operating system. The same ASP.NET Core web server/Internal Web Server, Kestrel, can be cross-platform and used behind a variety of proxies without putting any constraints on a particular implementation. Alternatively, if you wrote a new ASP.NET Core web server, you could use that in place of Kestrel without needing to change anything else about your application.
  2. Another benefit of a reverse proxy is that it can be hardened against potential threats from the public internet. They're often responsible for additional aspects, such as restarting a process that has crashed. Kestrel can stay as a simple HTTP server without having to worry about these extra features when it's used behind a reverse proxy. Think of it as a simple separation of concerns: Kestrel is concerned with generating HTTP responses; a reverse proxy is concerned with handling the connection to the internet.
- Using a reverse proxy has a number of benefits:
  1. Security—Reverse proxies are specifically designed to be exposed to malicious internet traffic, so they're typically well-tested and battle-hardened.
  2. Performance—You can configure reverse proxies to provide performance improvements by aggressively caching responses to requests.
  3. Process management—An unfortunate reality is that apps sometimes crash. Some reverse proxies can act as a monitor/scheduler to ensure that if an app crashes, the proxy can automatically restart it.
  4. Support for multiple apps—It's common to have multiple apps running on a single server. Using a reverse proxy makes it easier to support this scenario by using the host name of a request to decide which app should receive the request.

ASP.NET Core Module/Web Server: Kestrel.

- In OutOfProcess hosting model, we can use the kestrel server directly as user request facing server or we can deploy the application into IIS which will act as reverse proxy server and sends request to the internal kestrel server.
- Kestrel is a cross-platform web server for ASP.NET Core. It is supported on all platforms and versions that .NET Core supports.
- Kestrel is the internal web server that's included and enabled **by default in ASP.NET Core project templates.**
- With Out of Process Hosting model, Kestrel can be used in one of the following 2 ways.
  1. Kestrel can also be used by itself as an edge server i.e. Internet-facing web server that can directly process the incoming HTTP requests from the client
     - In this model we are not using an external web server. Only Kestrel is used and it is this server that faces the internet, to directly handle the incoming HTTP requests. When we run the asp.net core application using the .NET core CLI, Kestrel is the only web server that is used to handle and process the incoming HTTP request.

- Fig: Kestrel used as an edge server

2. Kestrel can be used in combination with the reverse proxy server.
    - With Out of Process Hosting, using a reverse proxy server is a good choice as it provides an additional layer of configuration and security. It might integrate better with the existing infrastructure. It can also be used for load balancing.
    - So, with a reverse proxy server in place, it receives incoming HTTP requests from the network and forwards them to the Kestrel server for processing. Upon processing the request, the Kestrel server sends the response to the reverse proxy server which then ultimately sends the response to the requested client over the network.

Fig: Kestrel used with reverse proxy server

- When we run .net core application from CLI, kestrel is the only web server that directly handles the incoming request.
- In Kestrel, the process used to host the app is **dotnet.exe.**

Switching between InProcess and OutofProcess Hosting

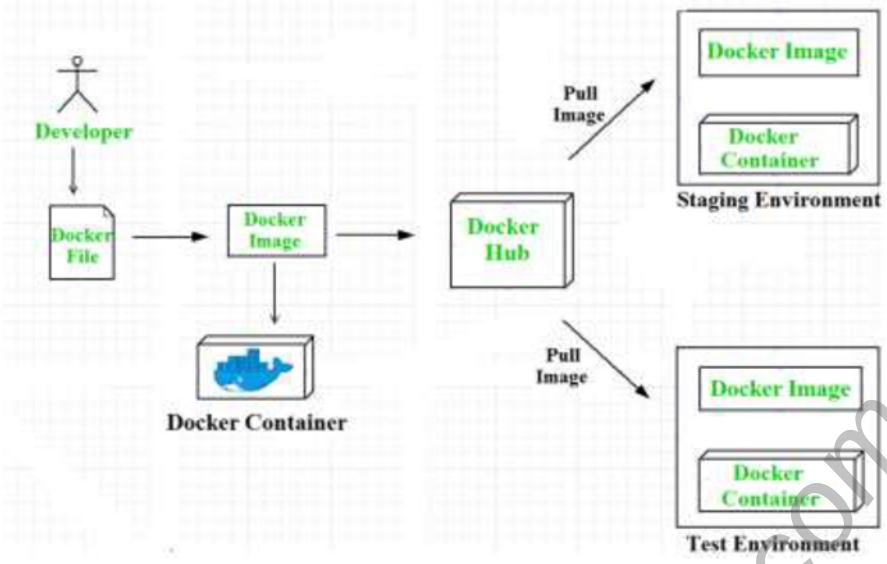
- An ASP.NET core application can be hosted InProcess or OutOfProcess.

```
<PropertyGroup>
  <AspNetCoreHostingModel>InProcess</AspNetCoreHostingModel>
</PropertyGroup>
```

- The relevant project setting (**in .csproj file**) is the AspNetCoreHostingModel which can be InProcess or OutOfProcess as per the hosting model requirement.
- When missing it defaults to the old OutOfProcess mode that uses an external Kestrel server with IIS acting as a proxy.
- This affects how dotnet publish creates your configuration when you publish your project and what it generates into the web.config file when the project is published.

Docker and Containerization

- Containerization is OS-based virtualization which creates multiple virtual units in the user space, known as Containers. Containers share the same host kernel but are isolated from each other through private namespaces and resource control mechanisms at the OS level.
- **Docker** is the containerization platform which is used to package your application and all its dependencies together in the form of containers so to make sure that your application works seamlessly in any environment which can be development or test or production.
- Docker is a tool designed to make it easier to create, deploy, run applications by using containers.



- The main components of Docker include

1. **Docker Client/Server**
   - Docker has a client-server architecture. The Docker Daemon/Server consists of all containers. The Docker Daemon/Server receives the request from the Docker client through CLI or REST APIs and thus processes the request accordingly. Docker client and Daemon can be present on the same host or different host.
2. **Docker File:**
   - Docker file is a text file that contains a series of instructions on how to build your Docker image. This image contains all the project code and its dependencies.
3. **Docker Images**
   - Docker images are used to build Docker containers by using a read-only template.
4. **Docker Containers**
   - Docker Containers are runtime instances of Docker images. Containers contain the whole kit required for an application, so the application can be run in an isolated way.
5. **Docker Registries**
   - Docker Registry is a storage component for Docker images. We can store the images in either public/private repository so that multiple users can collaborate in building the application. Docker Hub is Docker's own cloud repository. Docker Hub is called a public registry where everyone can pull available images and push their own images without creating an image from scratch

Website publishing

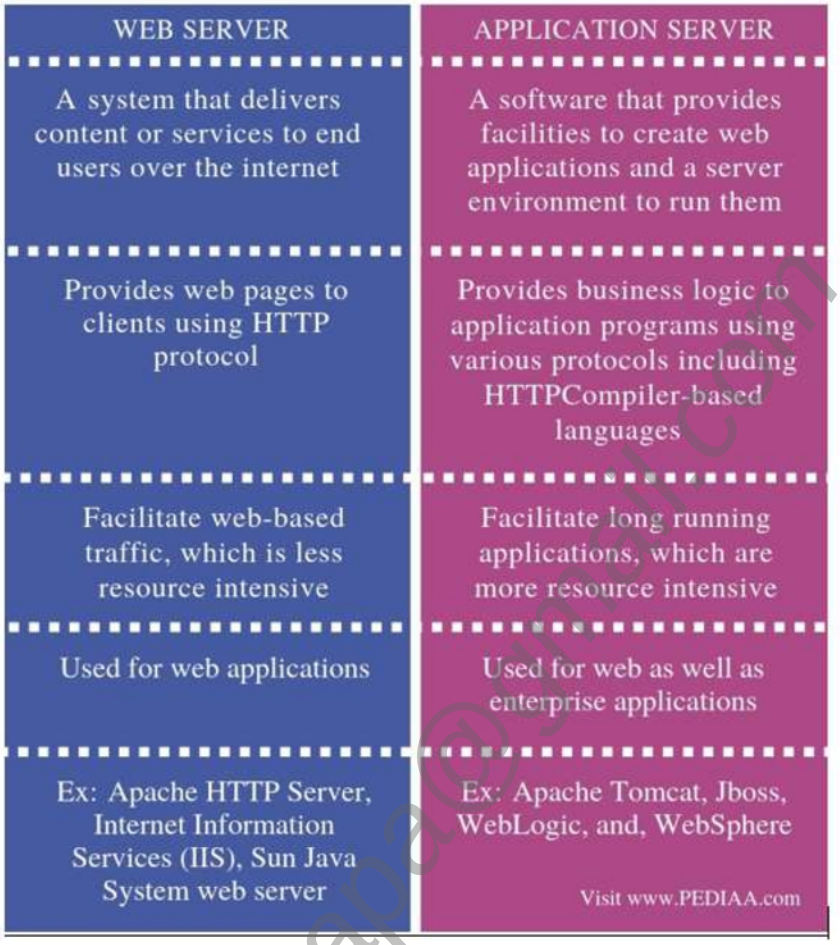
- Web publishing, or "online publishing," is the process of publishing content on the Internet.
- It includes creating and uploading websites, updating webpages, and posting blogs online.
- The published content may include text, images, videos, and other types of media.
- In order to publish content on the web, you need three things:
  1) web development software
  2) an Internet connection, and
  3) a web server
- The software may be a professional web design program like Dreamweaver or a simple web-based interface like WordPress.
- The Internet connection serves as the medium for uploading the content to the web server. Large sites may use a dedicated web host, but many smaller sites often reside on shared servers, which host multiple websites.
- Therefore, anyone with the three requirements above can be a web publisher.
- Additionally, the audience is limitless since content posted on the web can be viewed by anyone in the world with an Internet connection.

Website Deploying/Publishing to IIS and Azure

Will see this in lab

Assignment

1. Application server vs Web server



2. Inprocess Hosting Vs OutOfProcess Hosting
   a. With InProcess hosting, the application is hosted in the IIS worker process (w3wp.exe or iisexpress.exe) but in OutOfProcess hosting, the application is hosted in the process dotnet.exe.
   b. With InProcess hosting, there is only one web server i.e the IIS that hosts the asp.net core application but in OutOfProcess hosting, there are 2 web servers - An internal web server known as Kestrel and an external web server known as revere proxy server like IIS, Ngnix, Apache
   c. From a performance standpoint, InProcess hosting delivers significantly higher request throughput than OutOfProcess hosting