.NET

WEEK 1

Sharad K Singh

AGENDA

Day1 - Introduction

Day 2 - Architecture

Day 3 - Setup & Execution

Day 4 - Middleware & Deployment

DAY 1 OBJECTIVES

Introduction to .NET Core

.NET Core – Overview

.NET Platform Overview

Characteristics of .NET Core

Tooling

Learning Outcomes (Must Achieve)

By end of Day 1, students should:

- Clearly differentiate **.NET Framework vs .NET Core vs modern .NET**
- Understand **why .NET Core exists**
- Run their **first .NET Core console app**
- Understand **CLI vs Visual Studio workflow**

WHY LEARN .NET IN 2026

Enterprise & Cloud Applications

High demand in job market

Strong Microsoft ecosystem

Sharad K Singh

Typical .NET Enterprise Stack

- **Frontend**: Razor Pages, MVC, Blazor, Angular/React (with Web API)
- **Backend**: ASP.NET Core Web API
- **Business Layer**: Services, Domain Models
- **Data Layer**: EF Core, Dapper, SQL Server
- **Security**: JWT, Identity, Policy-based authorization

What Is a Cloud Application (in .NET)?

A **cloud application** is designed to run on cloud infrastructure (primarily **Azure** for .NET) and leverage **cloud-native services**.

Enterprise + Cloud = Modern .NET Applications

.NET is widely used for backend systems in finance, healthcare, manufacturing, government work and large ERP solutions.

Legacy systems still in production need developers for maintenance and upgrade efforts.

.NET isn't limited to one type of workload. Developers get opportunities in:

- **Web development** (ASP.NET Core)
- **Cloud / infrastructure** (Azure, AWS .NET workloads)
- **Cross-platform mobile** (MAUI / Xamarin)
- **Desktop** (Windows desktop, WPF, WinForms)
- **Game development** (Unity uses C# / .NET)

EVOLUTION OF .NET

.NET Framework – Windows only

.NET Core ()– Cross-platform

.NET Core ended with version 3.1

.NET 5 to 8 – Unified platform

There is **no .NET Core 4, 5, 6, 7, or 8 …**

From .NET 5 onward, the product name changed to **“.NET”**

Before .NET Core

- .NET Framework → Windows only
- Different runtimes for different platforms

With .NET Core

- Same code can run on:
  - Windows
  - Linux
  - macOS
- Major achievement:
  - Microsoft entered **Linux & cloud world**
  - Enabled Docker, Kubernetes, Azure Linux VMs

Key Meaning

"Cross-platform" means **same app can run on multiple OS**.

Reality in .NET Core era

- Multiple platforms existed separately:
  - .NET Framework (Windows)
  - .NET Core
  - Xamarin (Mobile)
  - Mono

So even though apps were cross-platform, **.NET itself was fragmented**.

What Microsoft did in .NET 5

They **merged everything** into one platform:

- .NET Core
- Xamarin
- Mono
- .NET Framework concepts

What "Unified" Means

- One runtime
- One BCL
- One SDK
- One CLI (dotnet)
- One way to build:
  - Web
  - API
  - Desktop
  - Mobile
  - Cloud
  - Microservices

Key Meaning

"Unified platform" means **one .NET for everything**.

Transition to Modern .NET (Unified Platform)

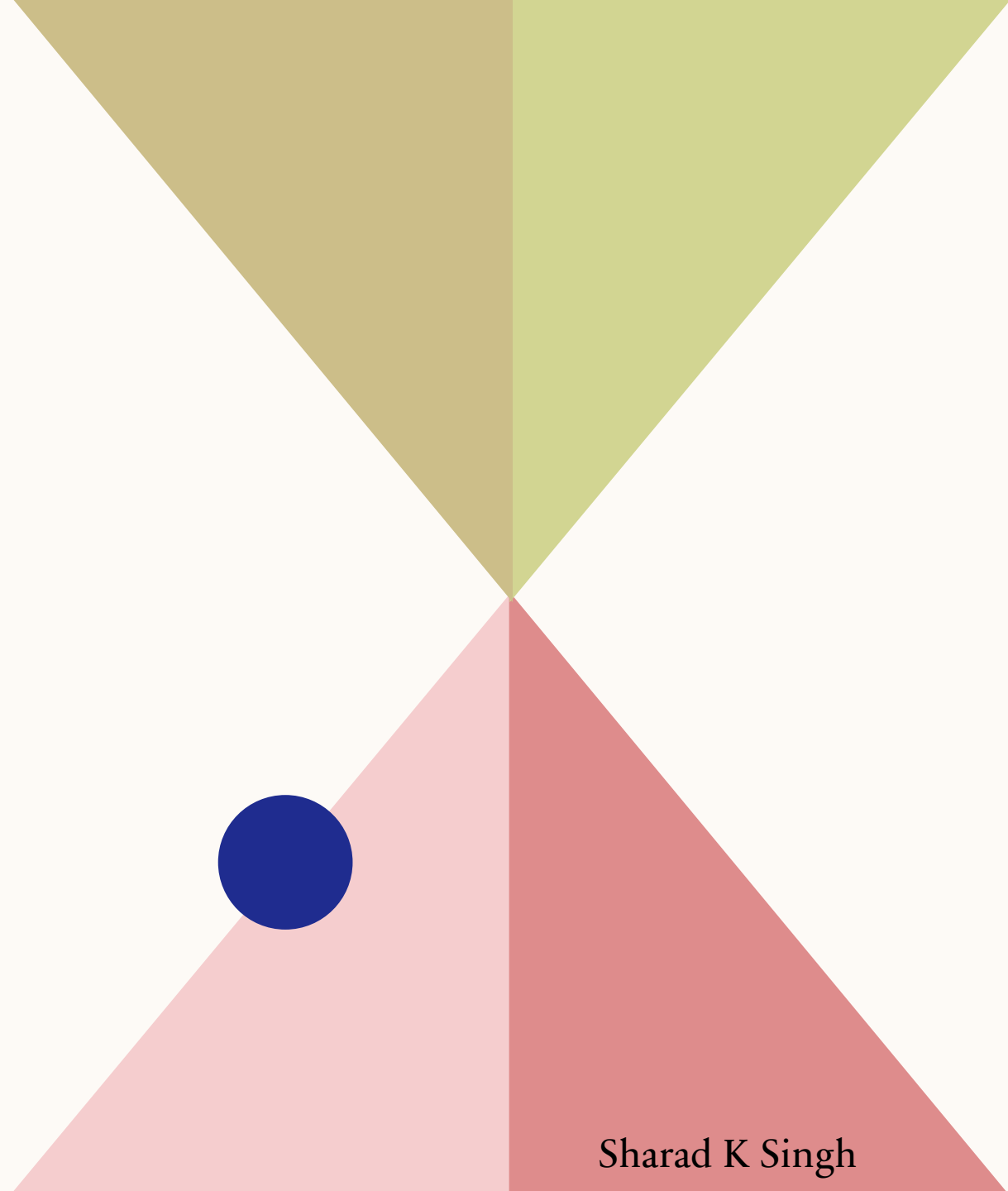
Version	Release Year	Support	Remarks
.NET 5	2020	❌ Ended	First unified .NET
.NET 6	2021	✅ LTS (ended 2024)	Enterprise standard
.NET 7	2022	❌ Ended	STS
.NET 8	2023	✅ **Current LTS**	**Recommended today**
.NET 9	2024	❌ STS	Short-term
.NET 10	2025	✅ LTS (expected)	Future enterprise baseline

WHAT IS .NET CORE

Open source platform

Build Console, Web, API apps

Runs on Windows, Linux, macOS

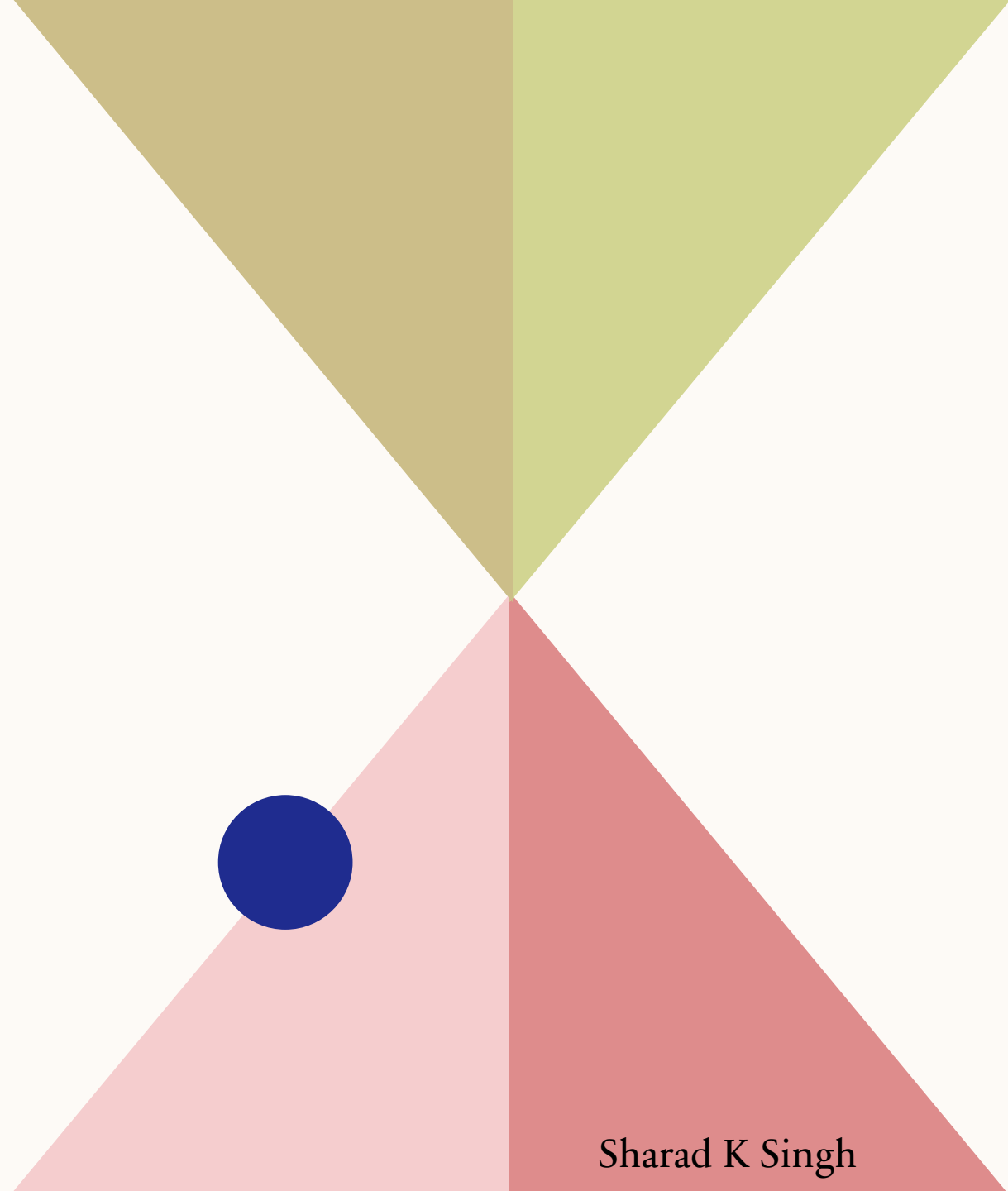


.NET PLATFORM OVERVIEW

CLR – Runtime

BCL – Libraries

SDK – Build tools



Sharad K Singh

1. .NET BCL (Base Class Library)

What it is

- A **collection of reusable classes, interfaces, and types**
- Provides **core functionality** needed by every .NET application

What BCL Contains

- Data types: int, string, DateTime
- Collections: List<T>, Dictionary<TKey,TValue>
- File & IO: File, Stream
- Networking: HttpClient
- Threading: Task, Thread
- LINQ, Reflection, Security

Key Point

BCL is what your application USES at runtime.

2. .NET SDK (Software Development Kit)

What it is

- A **set of tools** used to **build, run, test, and publish** .NET applications

What SDK Contains

- .NET CLI (dotnet)
- Compilers (C# compiler)
- MSBuild
- Templates (console, web, api)
- NuGet tools
- Runtime packs
- BCL (included indirectly)

Key Point

SDK is what developers USE to create applications.

Developer

↓

.NET SDK

↓ builds

Application

↓ runs on

.NET Runtime

↓ uses

.NET BCL

Programming	Real Life
BCL	Engine + parts of a car
SDK	Factory tools to build the car
Runtime	Driver + fuel

.NET BCL vs .NET SDK

- BCL provides core classes and APIs used by applications
- SDK provides tools to build, run, and publish applications
- BCL is used at runtime
- SDK is used during development

runtime/src/libraries/System.Console/src/System/Console.cs at main · dotnet/runtime

HOW .NET CODE EXECUTES

C# → IL → CLR → Machine Code

```
Source Code → IL → CLR → Machine Code
```

Managed execution

Optimized performance

Sharad K Singh

CHARACTERISTICS OF .NET CORE

Cross-platform

High performance

Modular & cloud-ready

Sharad K Singh

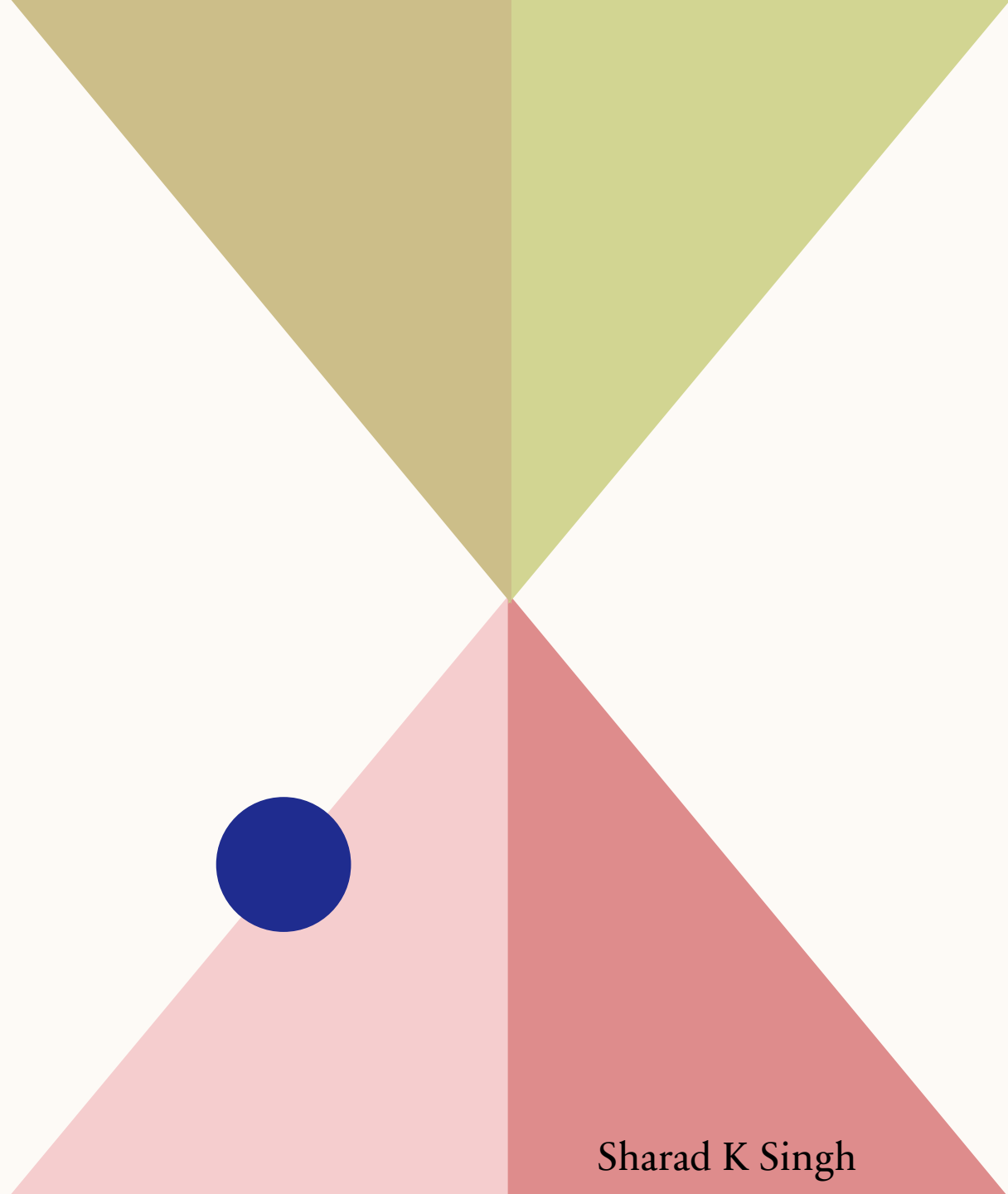
TOOLING

Visual Studio 2022

VS Code

.NET SDK

CLI



DAY 1 PRACTICE

Install .NET 8 SDK

Create Console App

Run using CLI

Sharad K Singh

1. During Development (Compile Time)

```
Console.WriteLine("Hello");
```

What happens:

- The **SDK compiler** checks:
  - Does `Console` exist?
  - Are method signatures valid?
- It does this by **referencing BCL assemblies**
- **BCL is NOT executing** here
- It is only **referenced for metadata**

Step-by-step instructions to use Visual Studio Code with the .NET CLI

1. Prerequisites (One-Time Setup)

Step 1: Install .NET SDK (NOT runtime)

- Download .NET 8 SDK (LTS) from Microsoft official site
- Ensure SDK is installed (not just runtime)

dotnet --version

Step 2: Install Visual Studio Code

- Install VS Code (lightweight editor)
- Recommended extensions:
    - C# Dev Kit
    - C#
    - .NET Install Tool

Step 3: Create a .NET Project Using CLI (Correct Way)

```
mkdir DotNetTraining
cd DotNetTraining

code .
```

Step 4: Create a New .NET Project

```
dotnet new console -n HelloApp
```

Step 5: Run the Application Using CLI

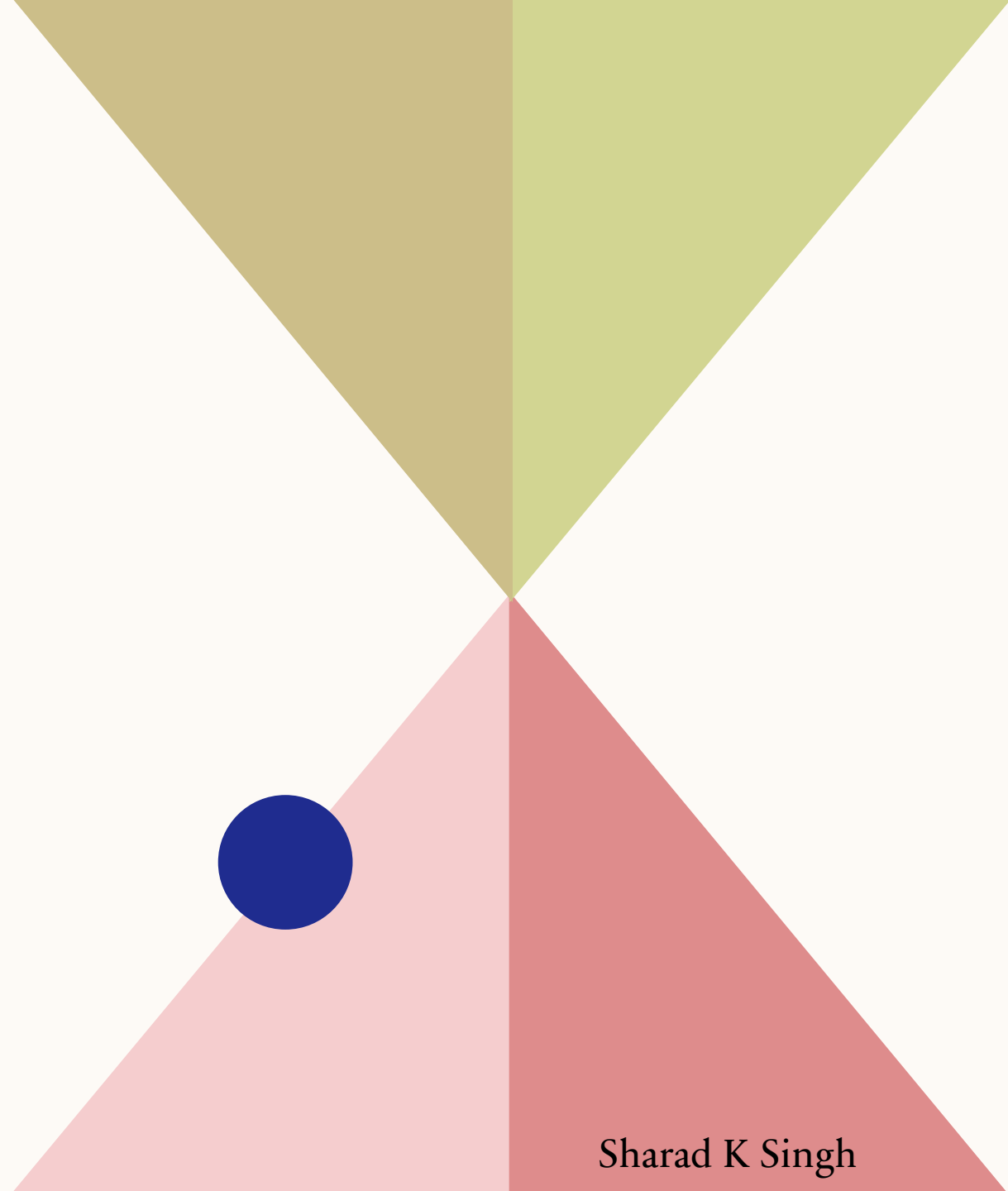
```
dotnet restore
dotnet build
dotnet run
```

DAY 2 OBJECTIVES

Understand Platform Architecture

Build Cross-platform Applications

.NET 8 features



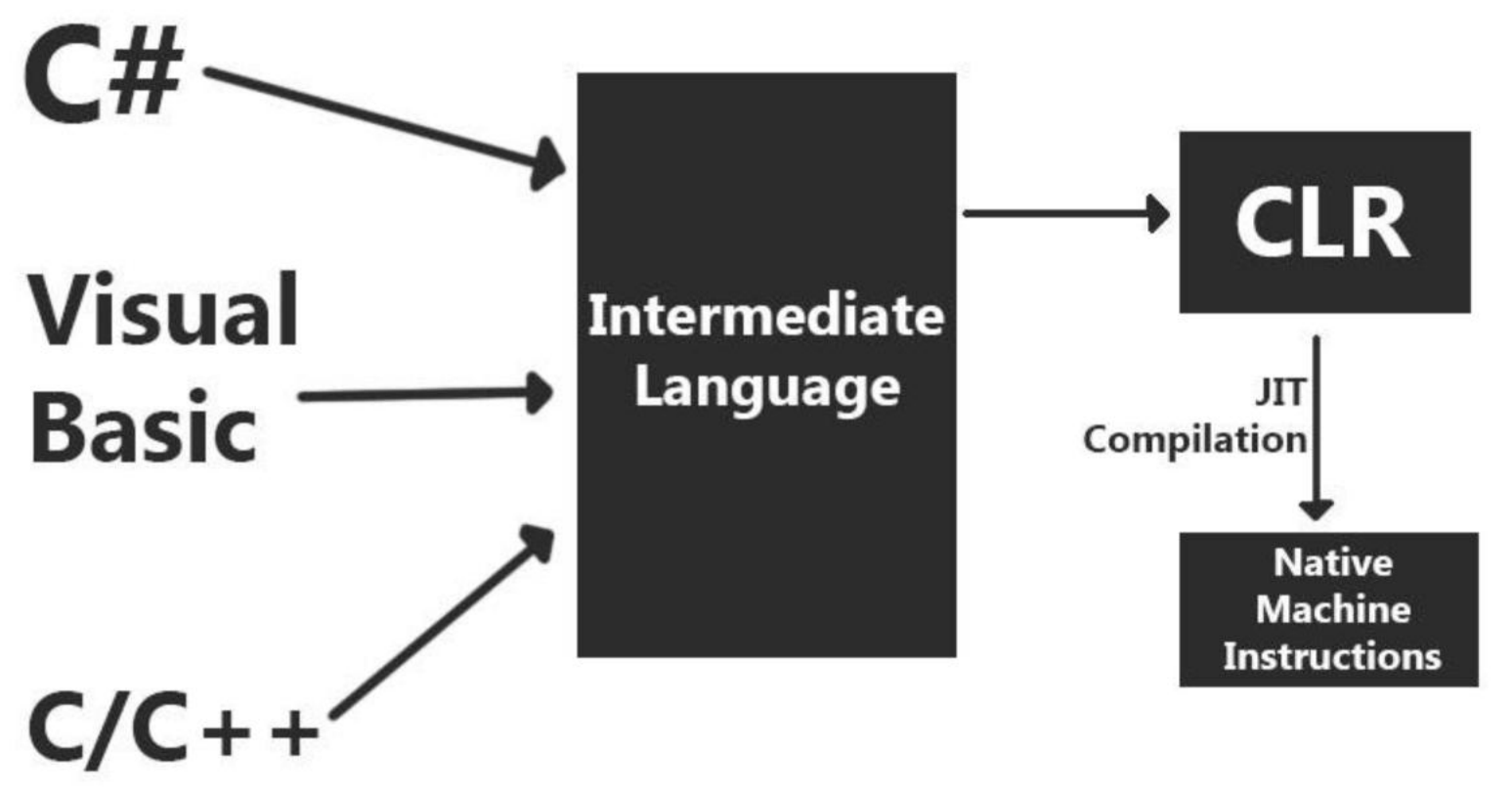
Sharad K Singh

.NET CORE ARCHITECTURE

.NET Runtime Architecture (How Code Executes)

```
C# / F# / VB Source Code
        ↓
Compiler (Roslyn)
        ↓
Intermediate Language (IL)
        ↓
CLR + JIT Compiler
        ↓
Native Machine Code
        ↓
CPU Execution
```

Sharad K Singh


C#
Visual Basic
C/C++
Intermediate Language
CLR
JIT Compilation
Native Machine Instructions

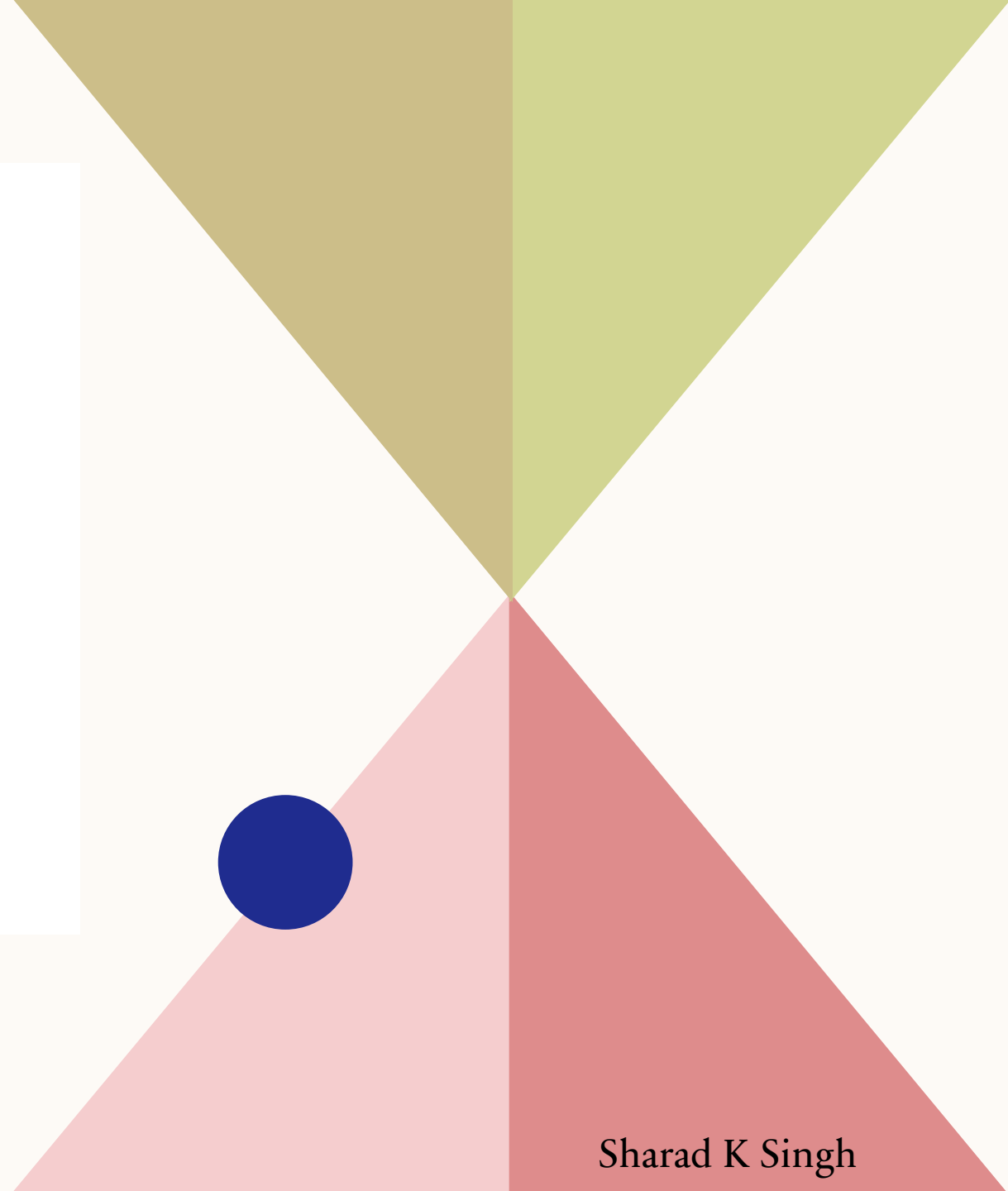
Key Runtime Components

1. Common Language Runtime (CLR)

The **execution engine** of .NET.

Responsibilities:

- Memory management (Garbage Collection)
- JIT compilation
- Exception handling
- Thread management
- Security enforcement



2. Intermediate Language (IL)

- CPU-independent bytecode
- Enables cross-platform execution
- Same IL runs on Windows, Linux, macOS

---

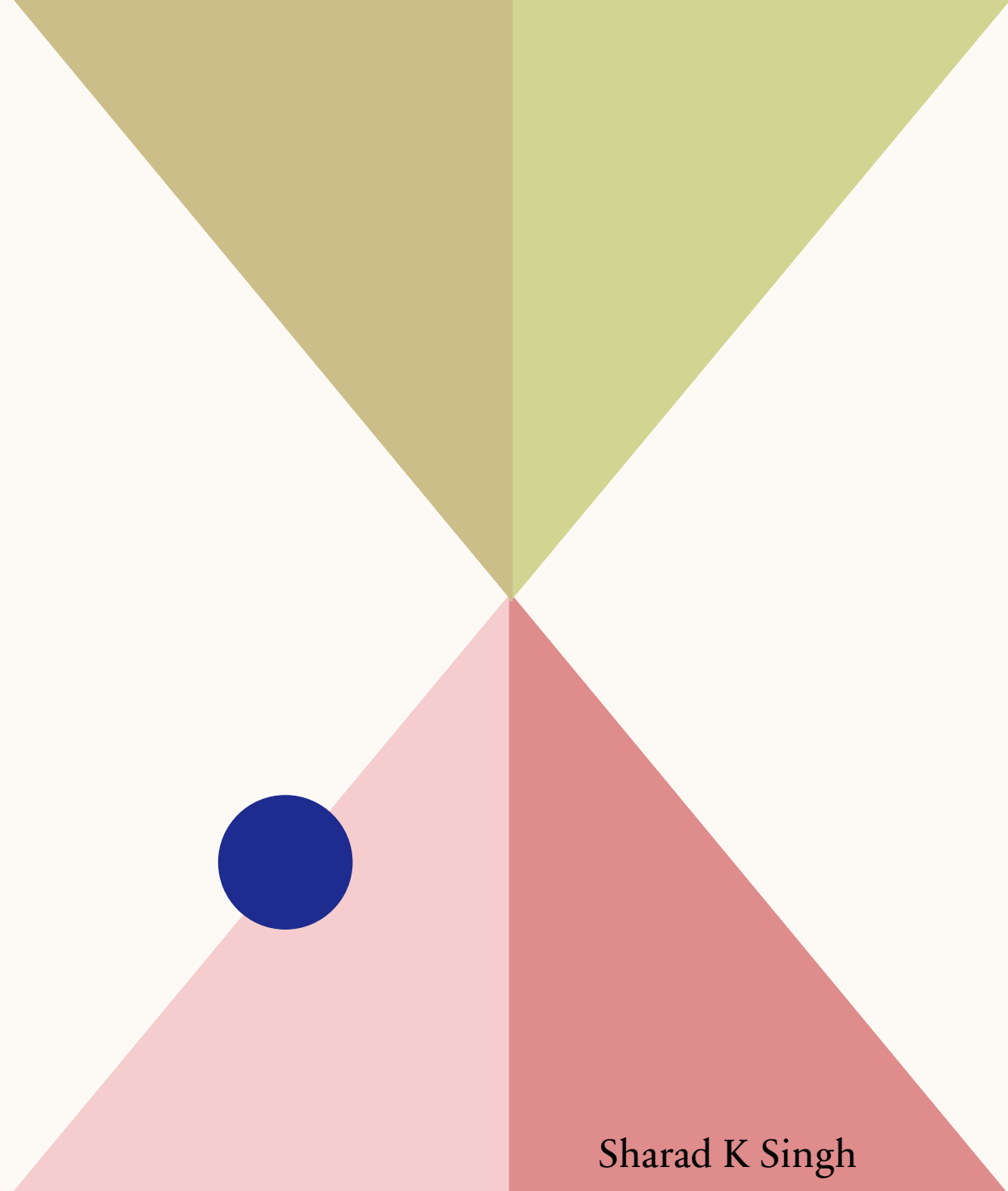
3. JIT Compiler

- Converts IL → native code at runtime
- Optimizes based on hardware and OS
- Improves execution performance

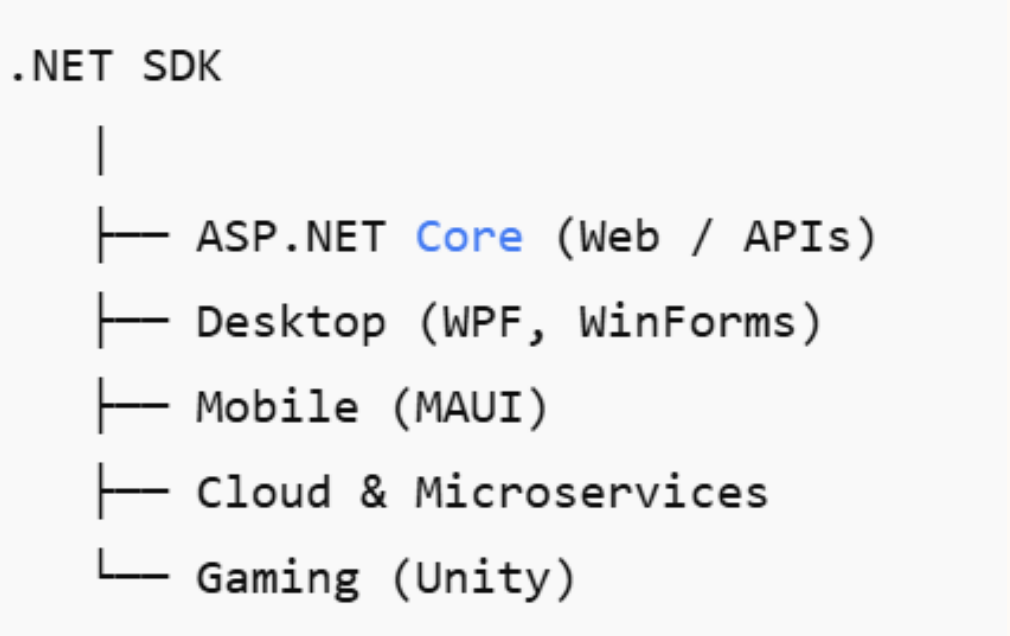
4. Base Class Library (BCL)

A rich library providing:

- Collections
- File I/O
- Networking
- LINQ
- Security
- JSON / XML handling



Unified .NET Platform



ADVANTAGES OF .NET CORE

Performance

Scalability

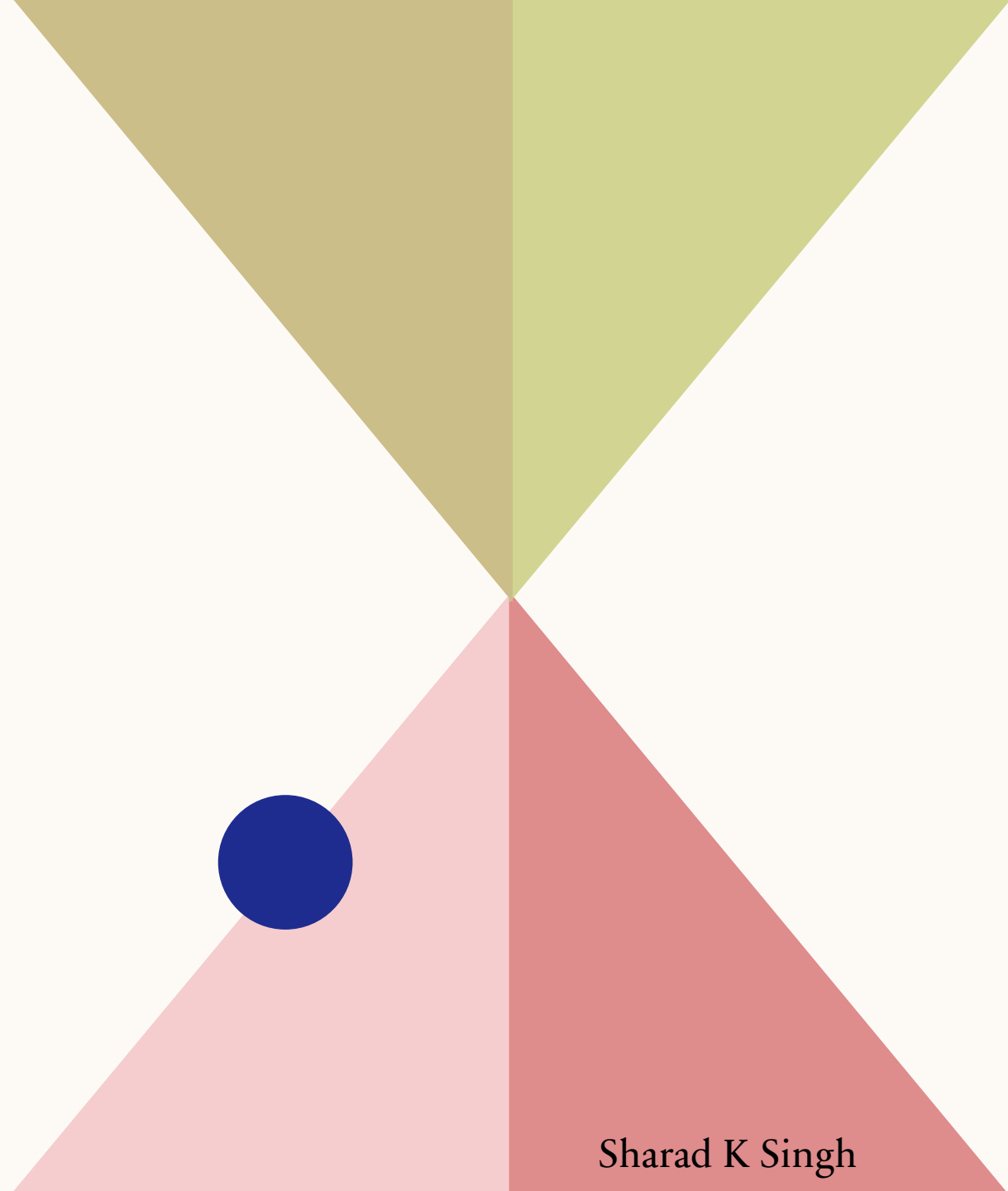
Microservices ready

Sharad K Singh

CROSS-PLATFORM DEVELOPMENT

Single codebase

Multiple OS



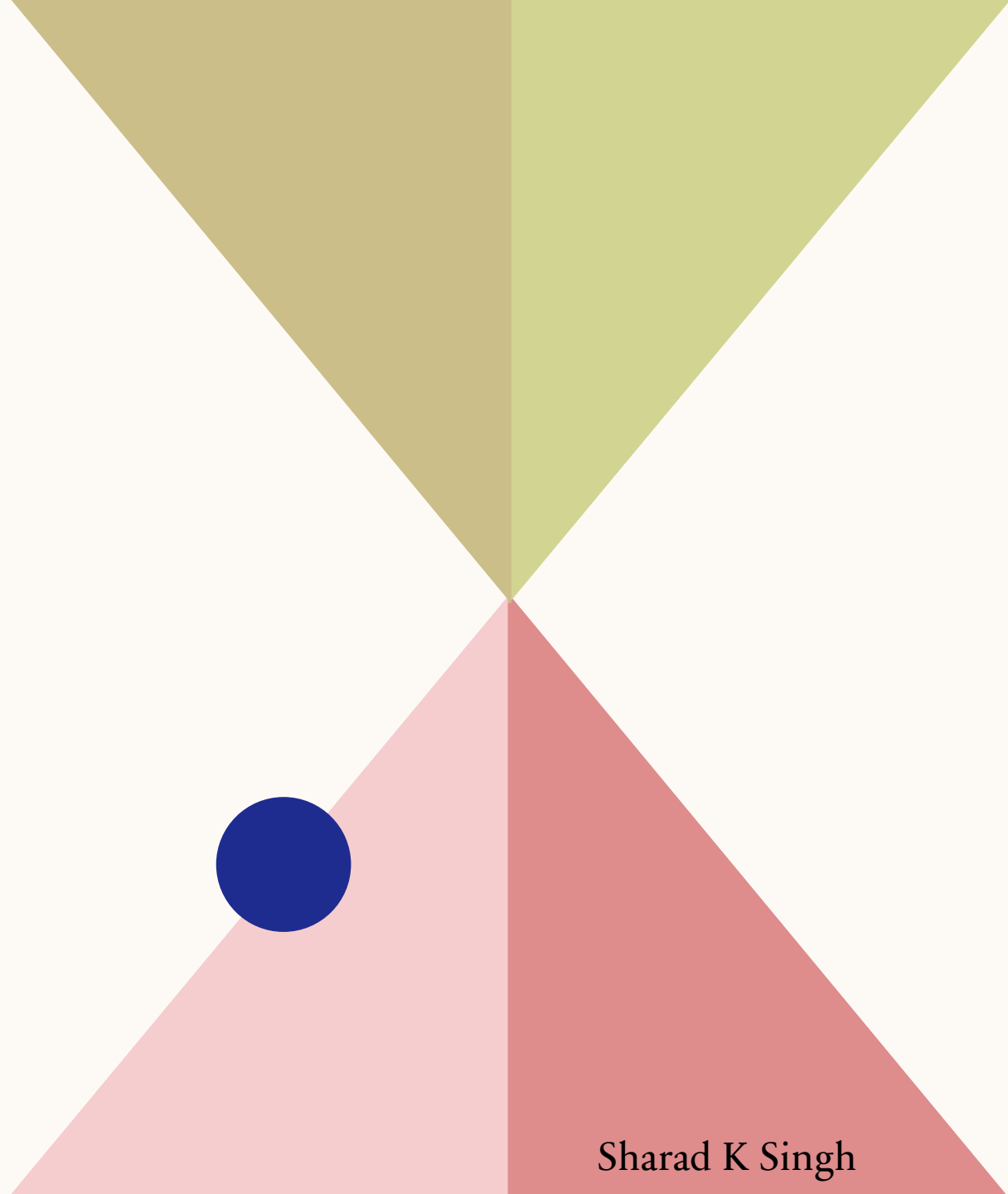
Sharad K Singh

BUILD & PUBLISH

dotnet build

dotnet run

dotnet publish



Demo 1 – Cross-Platform Console App (Best Starter)

Step 1: Create Project

1. Open **Visual Studio 2026**
2. Click **Create a new project**
3. Select **Console App**
4. Language: **C#**
5. Framework: **.NET 8 / .NET 9 / .NET 10**
6. Project Name: `CrossPlatformDemo`

Step 2: Write Platform-Aware Code

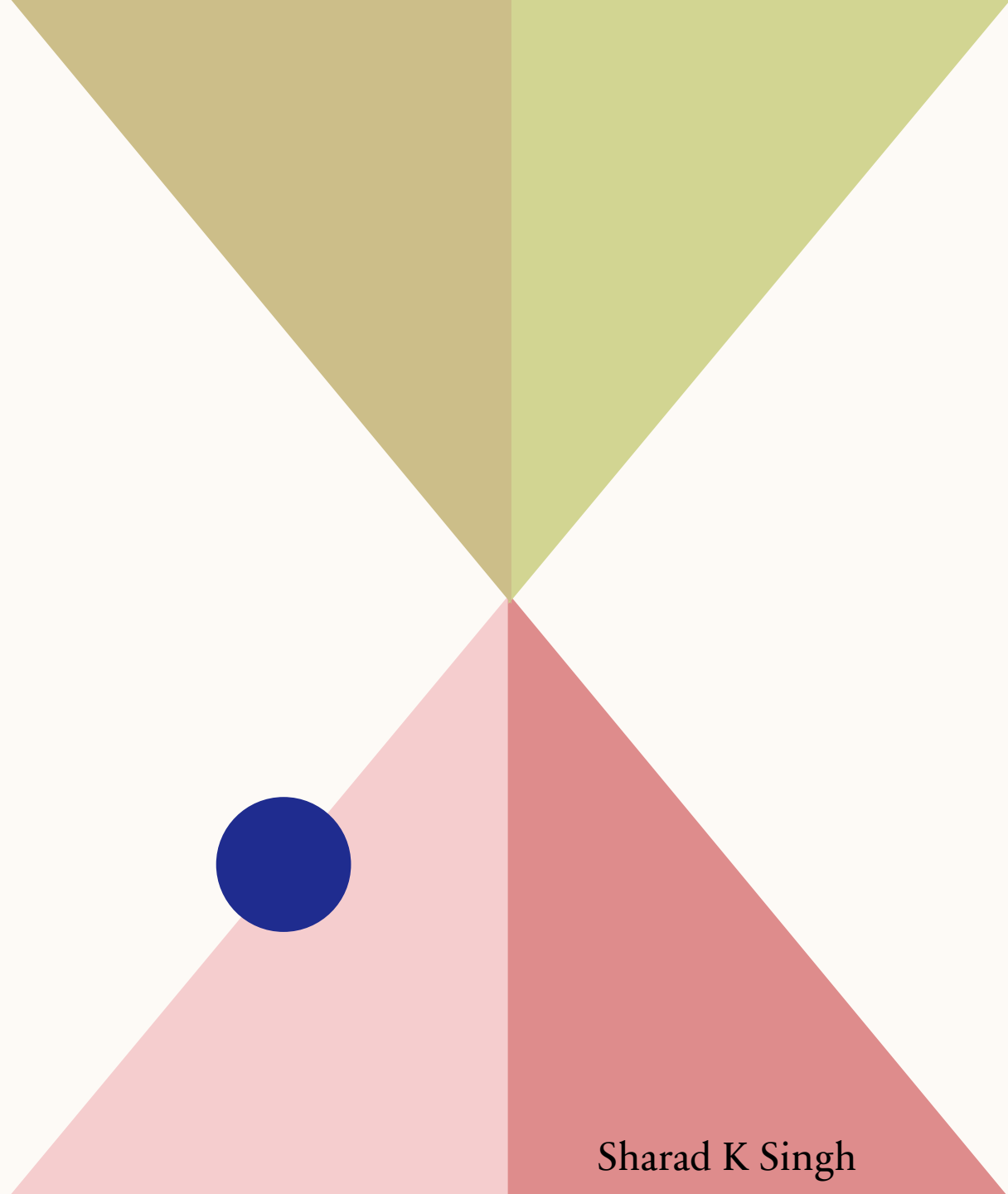
```
using System.Runtime.InteropServices;

Console.WriteLine("Cross-Platform .NET Application");
Console.WriteLine($"OS: {RuntimeInformation.OSDescription}");
Console.WriteLine($"Architecture: {RuntimeInformation.ProcessArchitecture}");
```

Step 3: Run on Windows (Local)

- Press **F5**
- Output window shows:

```
OS: Microsoft Windows 10.0.xxxxx
Architecture: X64
```



Demo 2 – Publish for Multiple Operating Systems

Step 4: Publish for Windows

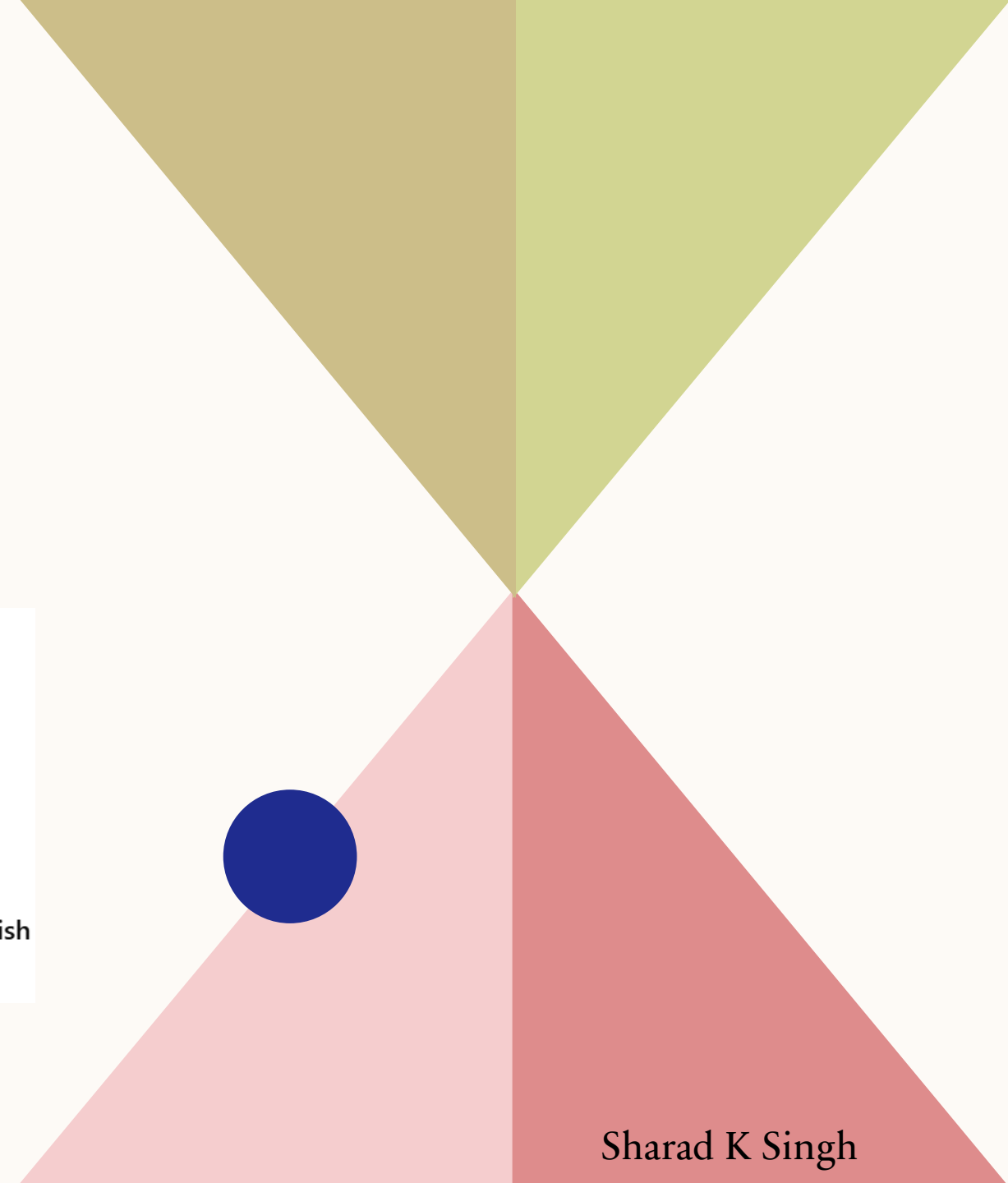
1. Right-click project → **Publish**
2. Target: **Folder**
3. Configuration:
   - Target runtime: `win-x64`
   - Deployment mode: **Self-contained**
4. Click **Publish**

Why You Don't See OS (win-x64 / linux-x64)

Visual Studio initially creates a **framework-dependent, portable publish profile**:

- ❌ No OS selection
- ❌ No self-contained option
- ✅ Runs wherever .NET is installed

To change OS, you must **edit advanced publish settings** or **switch to a custom publish profile**.



Publish

Where are you publishing today?

Target

Azure
Host your application to the Microsoft cloud

ClickOnce
Publish your application with ClickOnce

Docker Container Registry
Publish your application to any supported Container Registry that works with Docker images

Folder
Publish your application to a local folder or file share

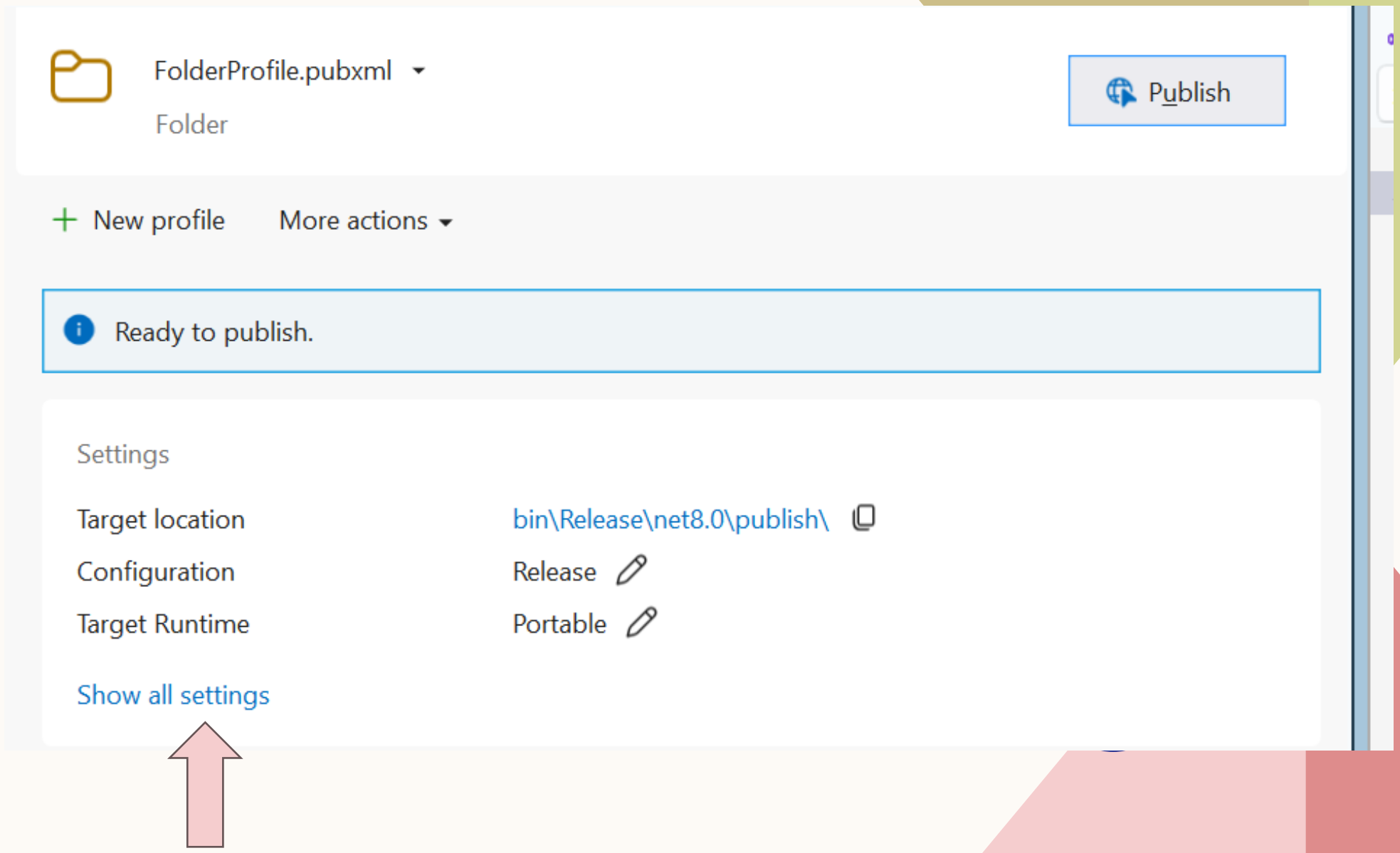
Import Profile
Import your publish settings to deploy your app

Back | Next | Finish | Cancel

Step 2: Open Publish Profile Settings

After profile is created:

1. You will see a **Publish summary page**
2. Click **Edit** (✏️ icon) or **Settings**
    - Usually on the right side or top bar


FolderProfile.pubxml
Folder
Publish
New profile
More actions
Ready to publish.
Settings
Target location
bin\Release\net8.0\publish\
Configuration
Release
Target Runtime
Portable
Show all settings

Profile settings

Profile name FolderProfile

Configuration: Release | Any CPU

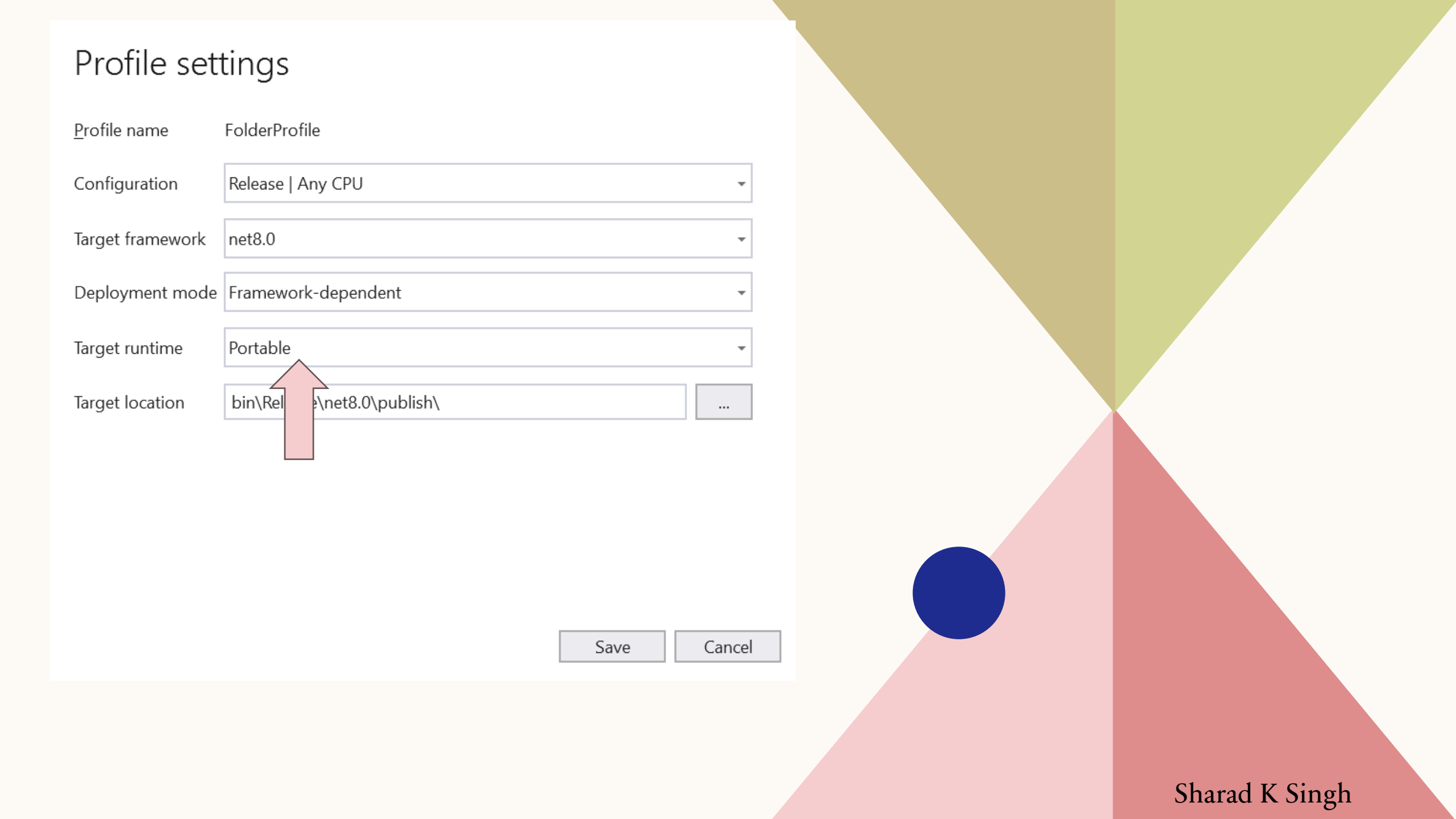
Target framework: net8.0

Deployment mode: Framework-dependent

Target runtime: Portable

Target location: bin\Rel e\net8.0\publish\ ...

Save Cancel



Portable Deployment (Framework-Dependent)

> **Portable deployment produces a single build that runs on any OS where the matching .NET runtime is already installed.**

- No OS specified
- No runtime bundled
- Smaller output
- Depends on installed .NET

OS-Targeted Deployment (Self-Contained)

> **OS-targeted deployment produces a build for a specific operating system and CPU architecture, bundling the .NET runtime with the app.**

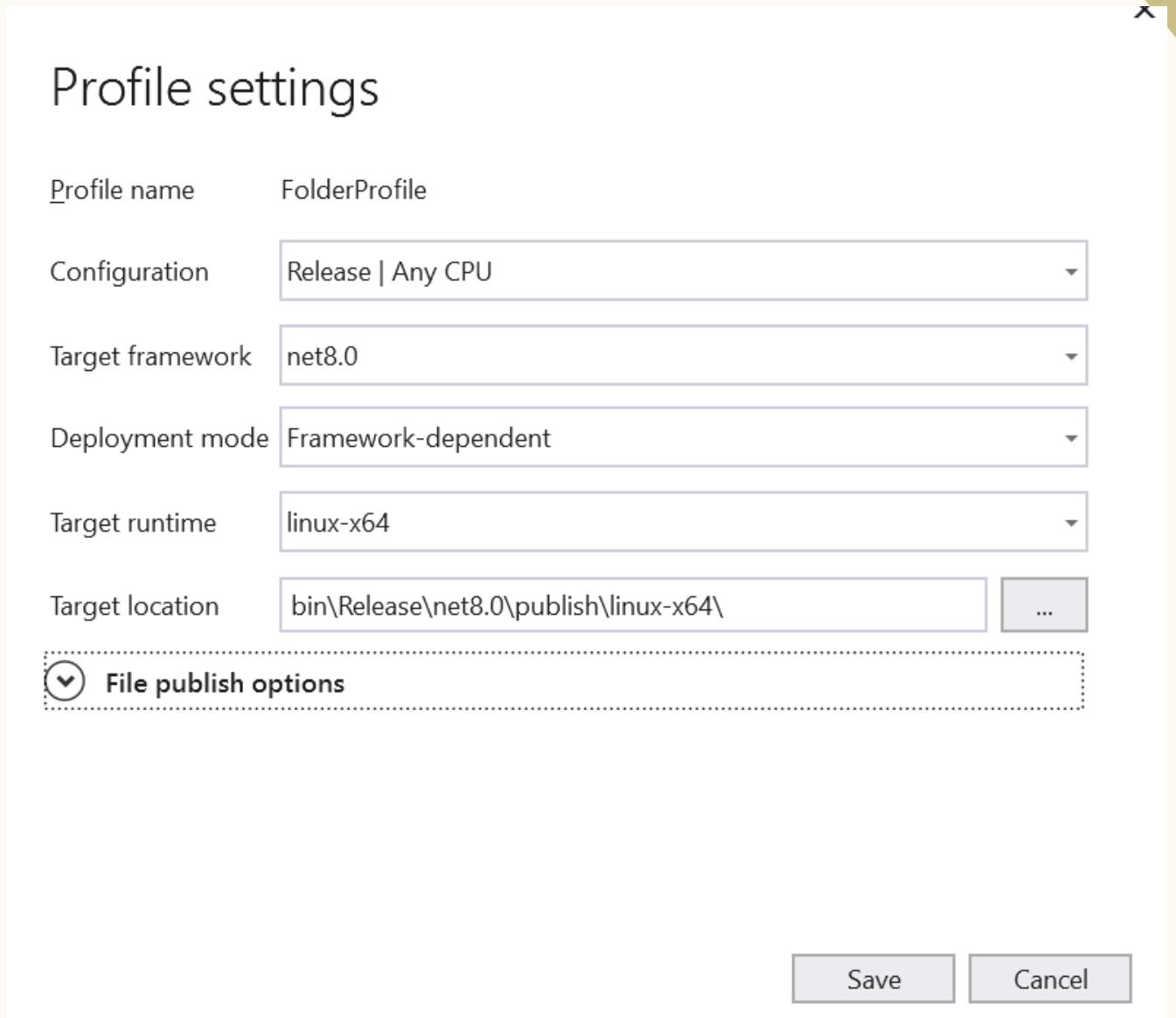
- OS explicitly specified
- Runtime included
- Larger output
- Runs without .NET preinstalled

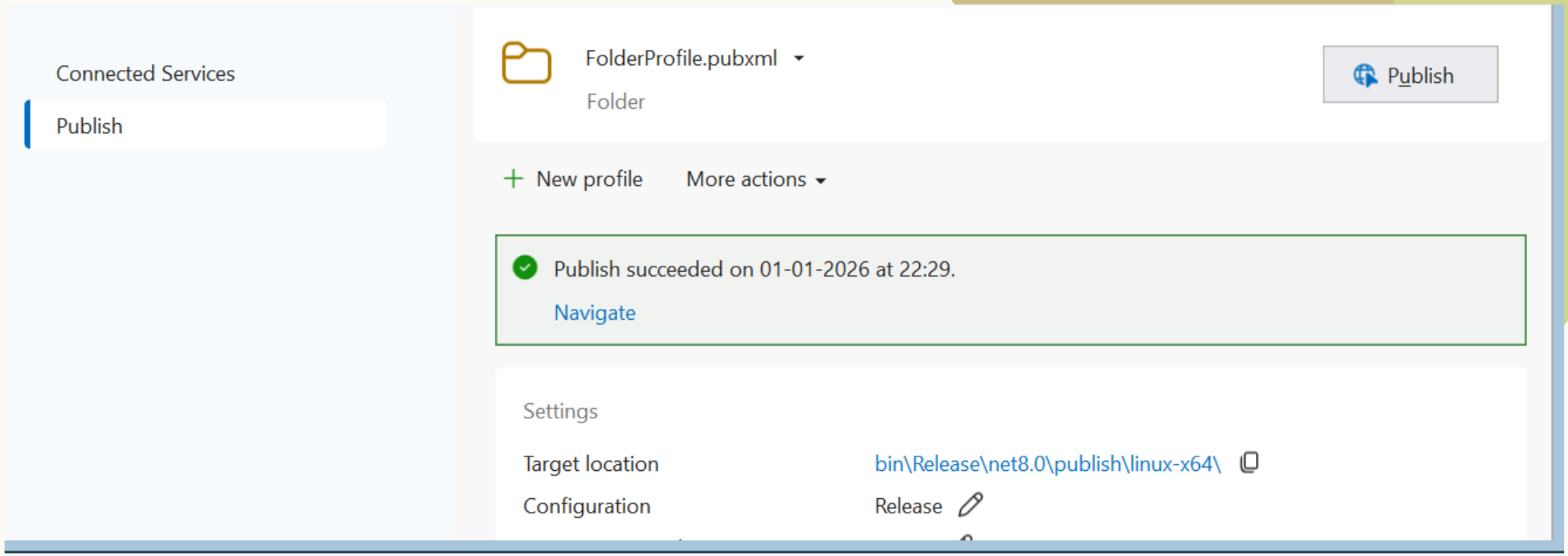
SELF-CONTAINED VS FRAMEWORK-DEPENDENT

Self-contained – includes runtime

runtime-dependent – smaller

Sharad K Singh


Profile settings
Profile name
FolderProfile
Configuration
Release | Any CPU
Target framework
net8.0
Deployment mode
Framework-dependent
Target runtime
linux-x64
Target location
bin\Release\net8.0\publish\linux-x64\
...
File publish options
Save
Cancel



Output

how output from: Build

```
1>------ Build started: Project: ConsoleApp3, Configuration: Release Any CPU ------
1>  ConsoleApp3 -> C:\Users\Sharad\source\repos\ConsoleApp3\ConsoleApp3\bin\Release\net8.0\ConsoleApp3.dll
2>------ Publish started: Project: ConsoleApp3, Configuration: Release Any CPU ------
2>Determining projects to restore...
2>Restored C:\Users\Sharad\source\repos\ConsoleApp3\ConsoleApp3\ConsoleApp3.csproj (in 7.29 sec).
2>C:\Program Files\dotnet\sdk\10.0.101\Roslyn\binfx\..\bincore\csc.exe /noconfig /sdkpath:C:\Windows\Microsoft.NET\Framework64\v4.0.30
2>ConsoleApp3 -> C:\Users\Sharad\source\repos\ConsoleApp3\ConsoleApp3\bin\Release\net8.0\linux-x64\ConsoleApp3.dll
2>ConsoleApp3 -> C:\Users\Sharad\source\repos\ConsoleApp3\ConsoleApp3\bin\Release\net8.0\publish\linux-x64\
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 22:29 and took 12.847 seconds ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
========== Publish completed at 22:29 and took 12.847 seconds ==========
```

Portable (framework-dependent) deployment produces a single cross-platform build that relies on a preinstalled .NET runtime, while OS-targeted (self-contained) deployment produces separate builds per operating system, bundling the runtime for standalone execution.

❌ “Portable means Windows only”

❌ “OS-targeted is faster”

✅ Portable = runtime-dependent

✅ OS-targeted = runtime included

.NET FEATURES

Garbage Collection (GC)

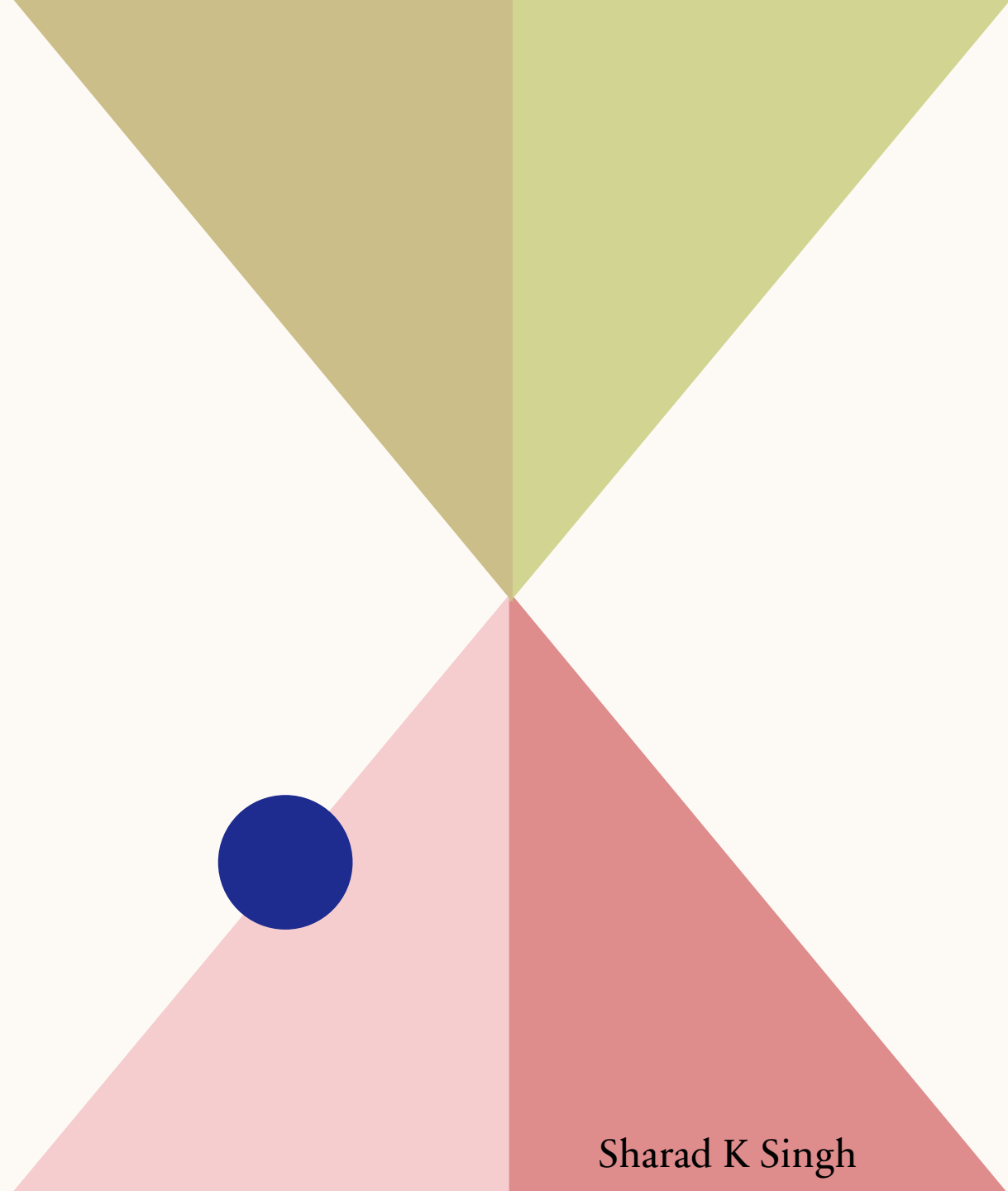
Cross-Platform Development

Language Interoperability

Managed Runtime (CLR)

Rich Base Class Library (BCL)

Strong Type Safety



.NET 8 FEATURES

Performance boosts - **Faster apps:** .NET 8 runs programs quicker than before

Ahead-of-Time (AOT) Compilation: You can turn your code into a program that the computer runs directly.

C# 12 shipped with the .NET 8 SDK.

DAY 2 PRACTICE

Add NuGet package

Publish app

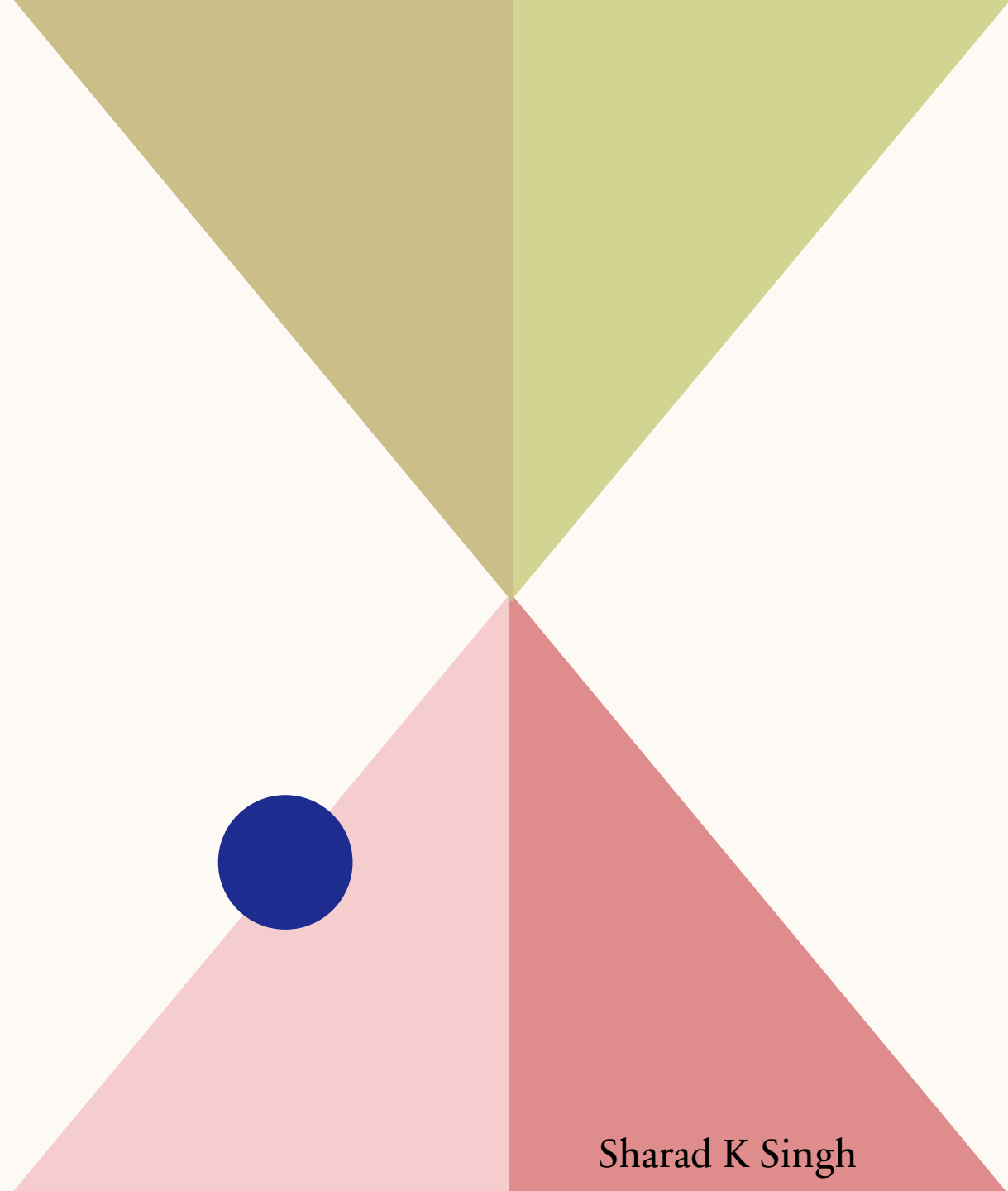
Sharad K Singh

DAY 3 OBJECTIVES

SDK vs Runtime

Execution flow

Project structure

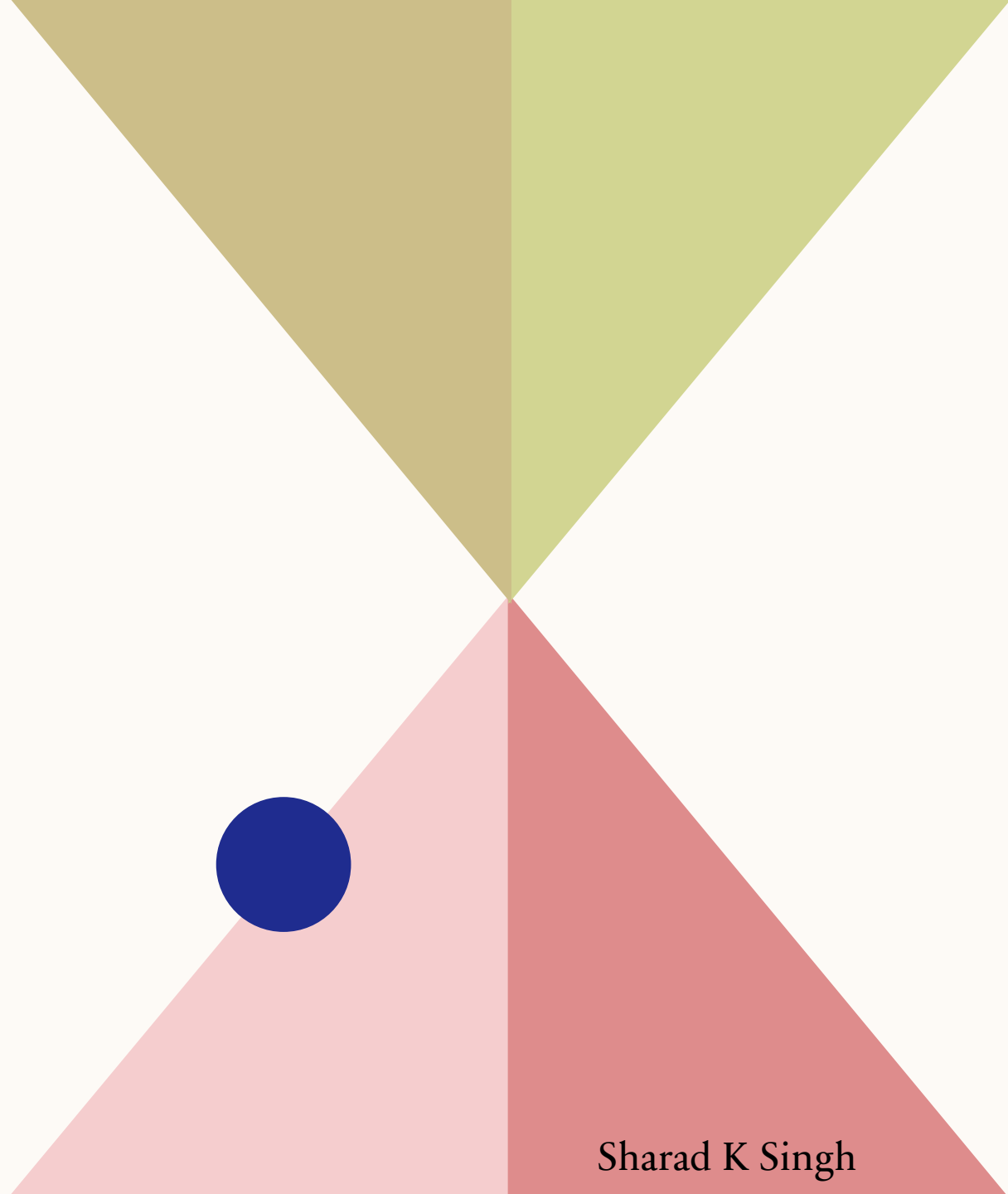


Sharad K Singh

SDK VS RUNTIME

SDK builds apps

Runtime runs apps

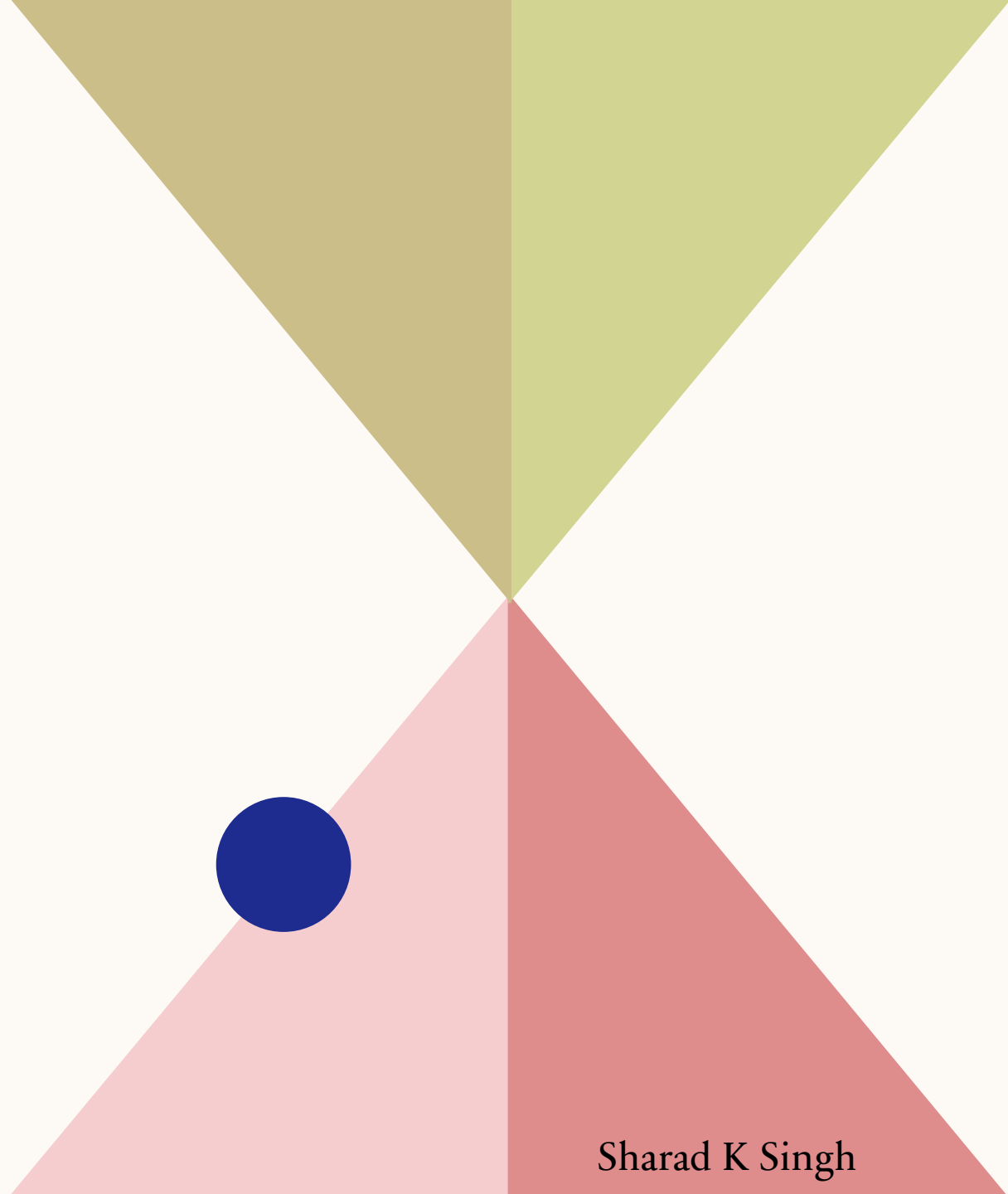


Sharad K Singh

GLOBAL.JSON

Lock SDK version

Team consistency



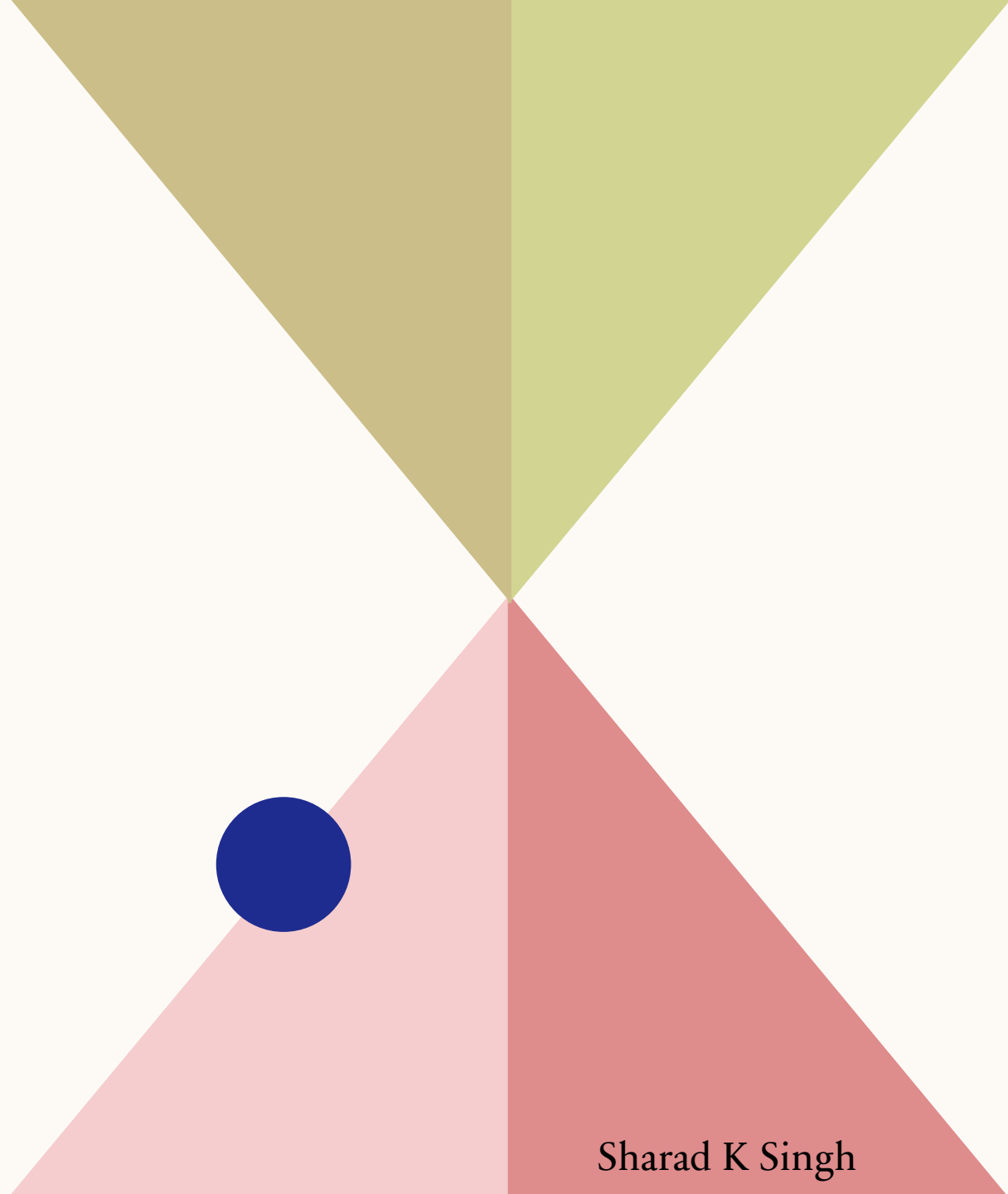
Sharad K Singh

EXECUTION FLOW

Entry point

Top-level statements

Build pipeline



DOTNET CLI COMMANDS

new

build

run

clean

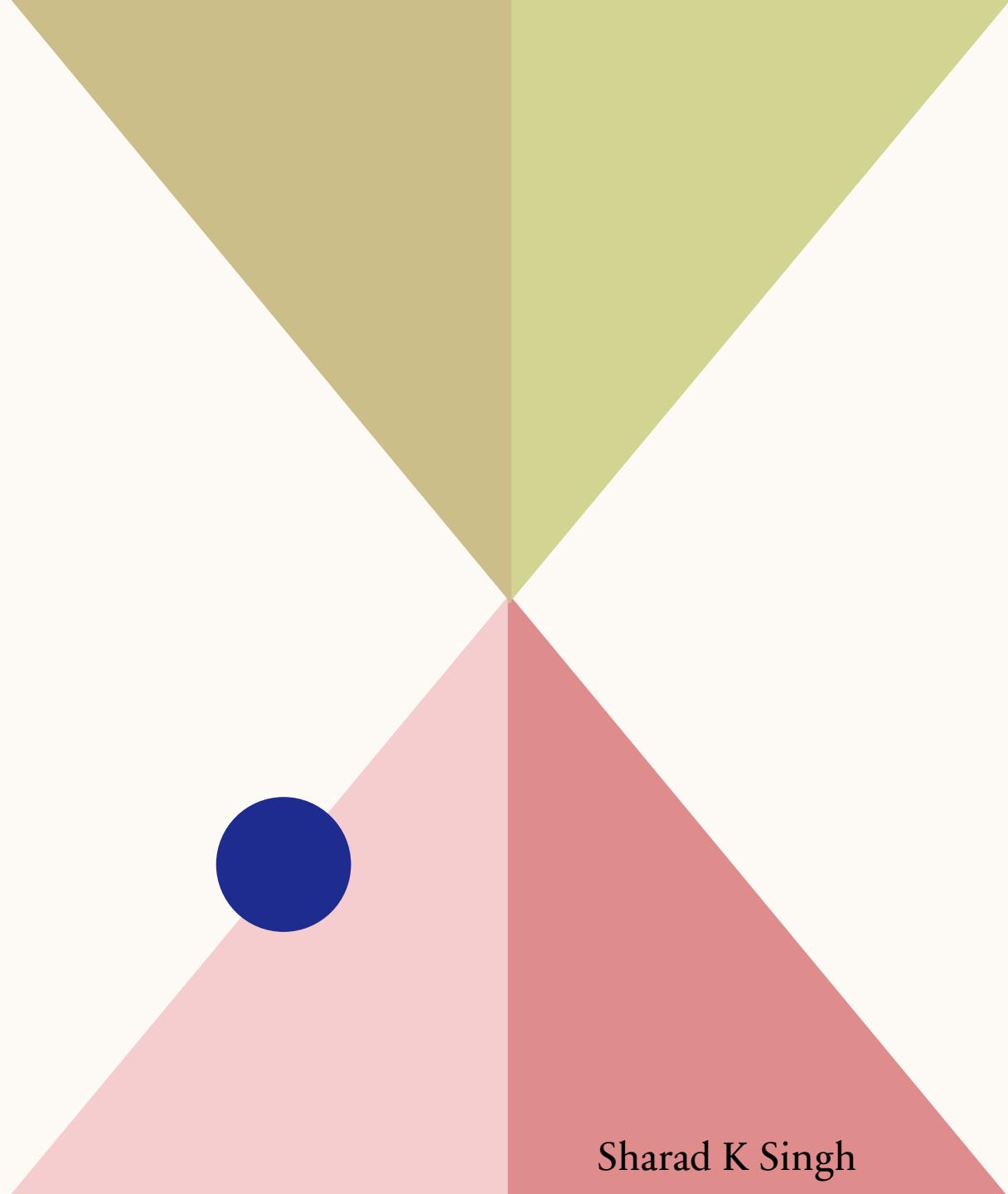
Sharad K Singh

MODULARITY

Class Libraries

Separation of concerns

Reusable code



Sharad K Singh

PROJECT STRUCTURE

Program.cs

bin / obj

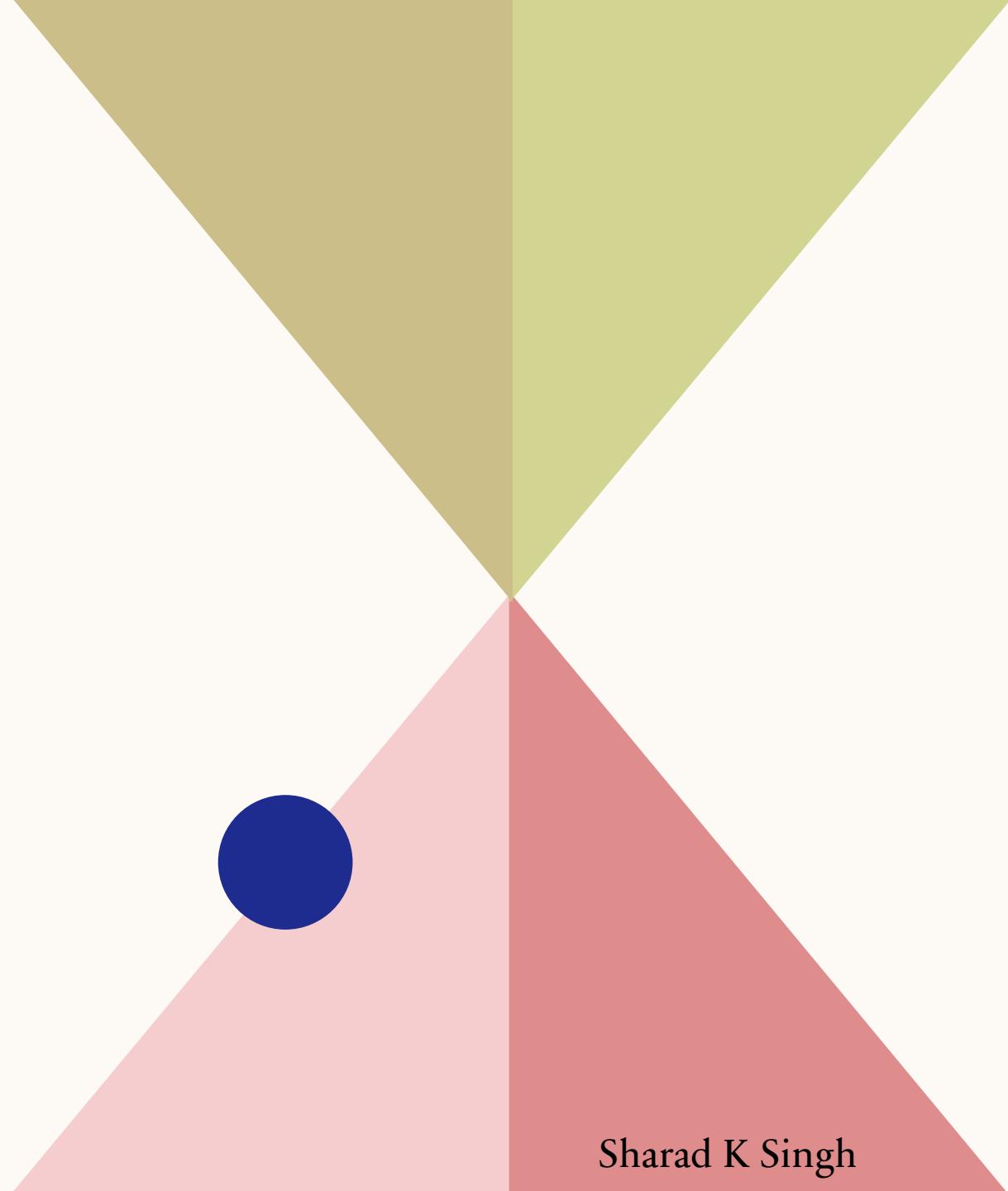
.csproj

.CSPROJ FILE

TargetFramework

Nullable

ImplicitUsings



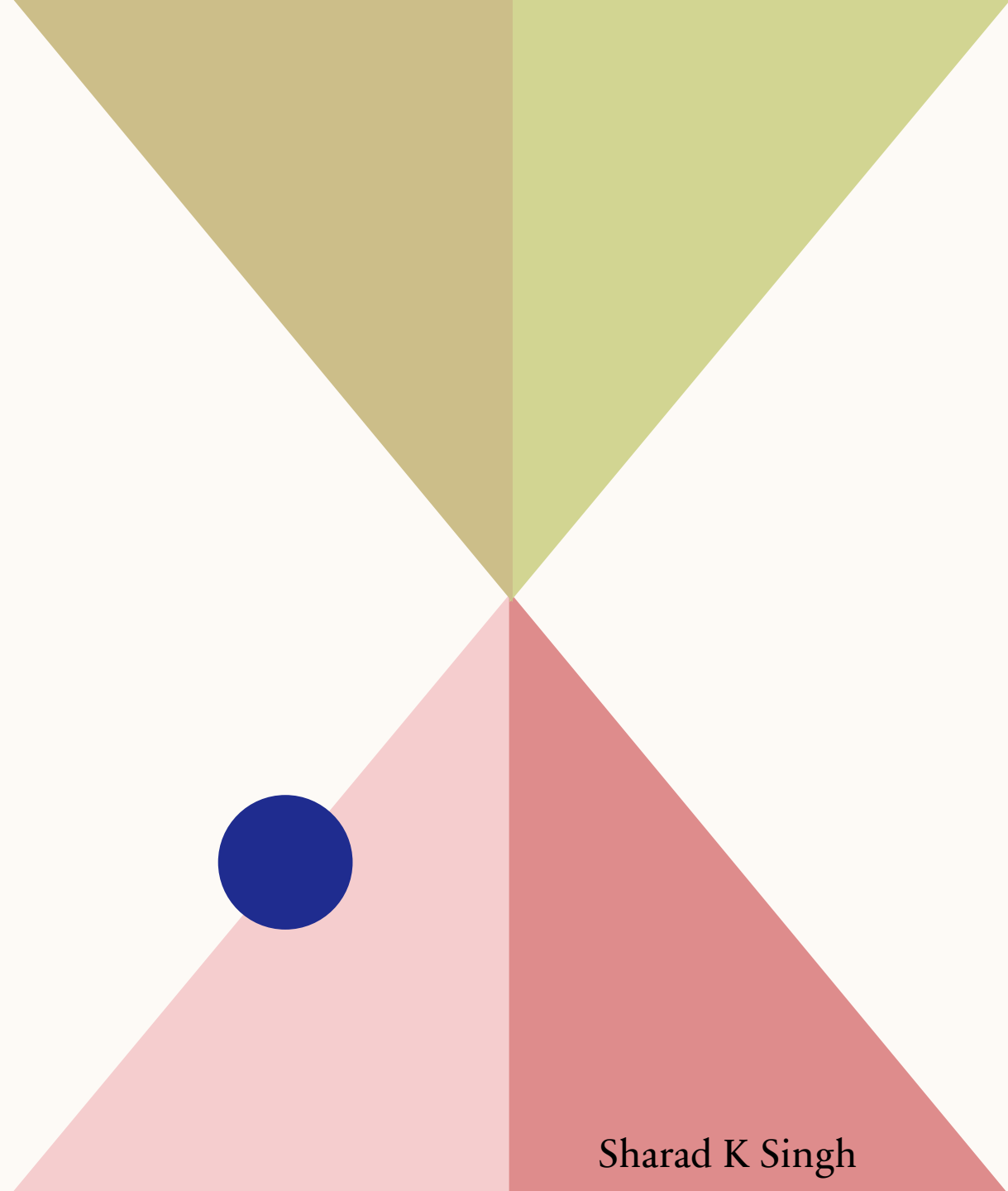
Sharad K Singh

DAY 3 PRACTICE

Create class library

Add reference

Run solution



Sharad K Singh

DAY 4 OBJECTIVES

Middleware

IIS hosting

Deployment basics

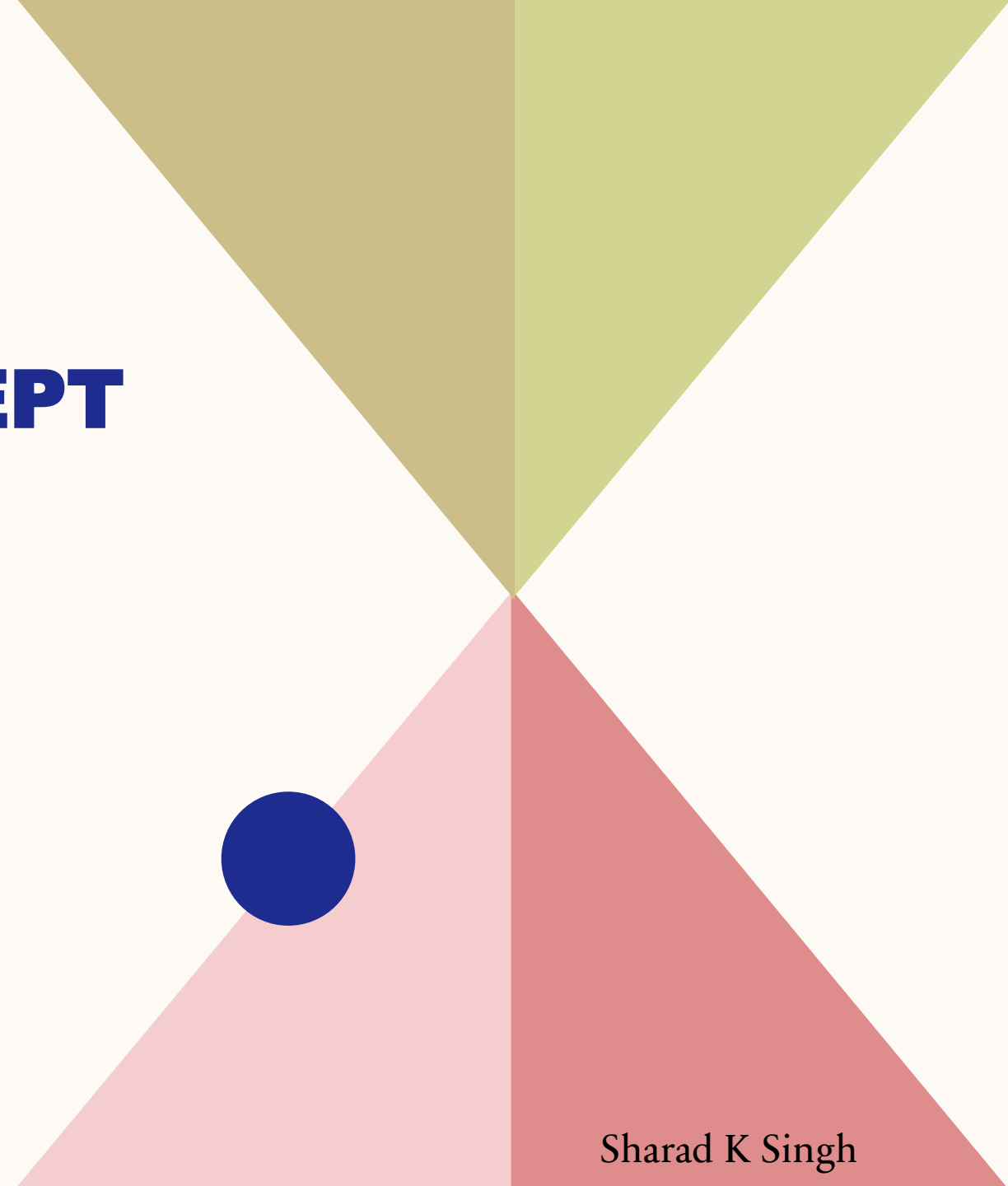
Sharad K Singh

MIDDLEWARE CONCEPT

Request pipeline

Order matters

Use / Run / Map



Sharad K Singh

MIDDLEWARE FLOW

Request → Middleware → Response

Sharad K Singh

CUSTOM MIDDLEWARE

Before logic

Next delegate

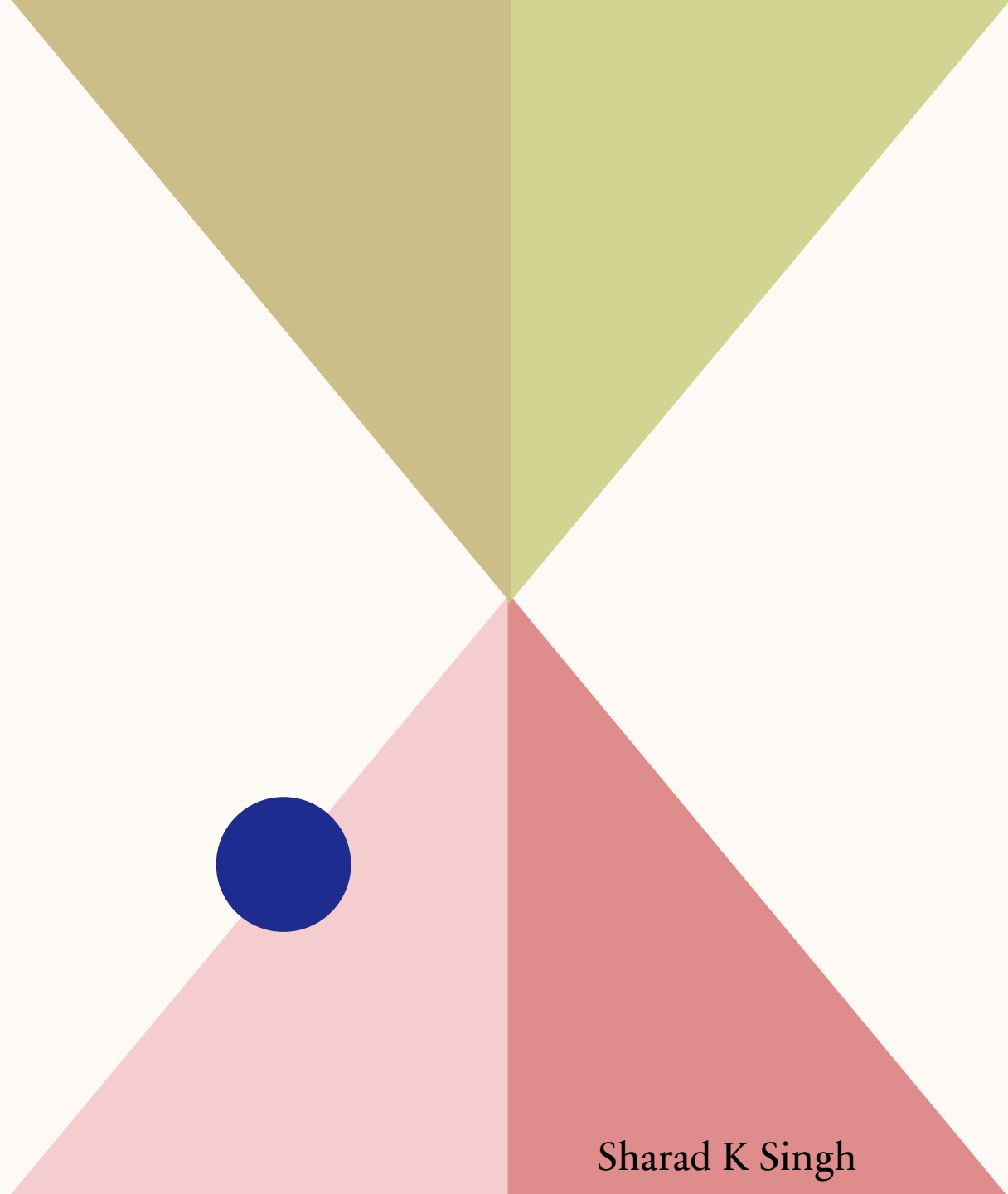
After logic

HOSTING OPTIONS

Kestrel

IIS

Reverse Proxy



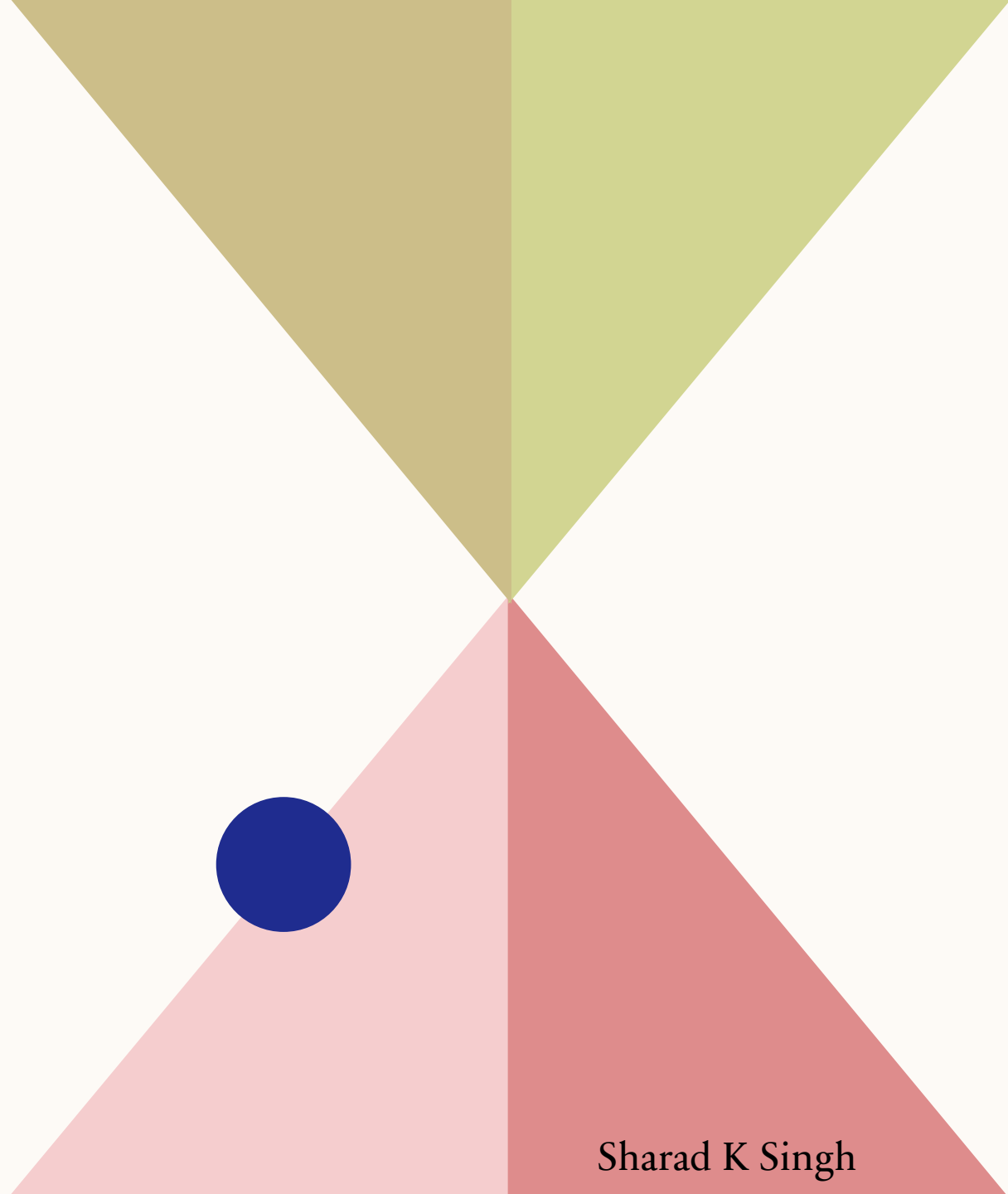
Sharad K Singh

IIS PUBLISHING

Hosting bundle

Publish profile

In-process vs Out



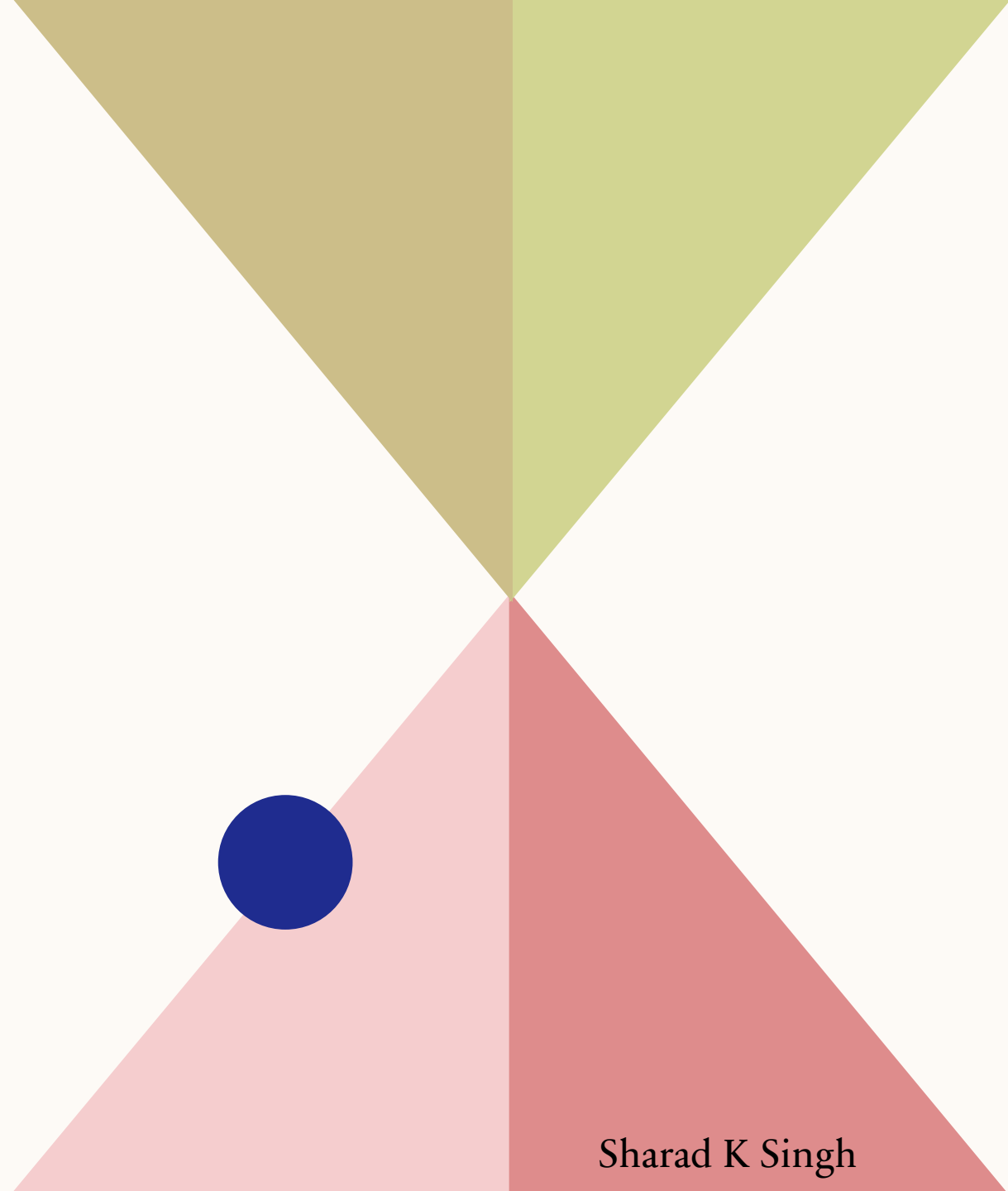
Sharad K Singh

CROSS-PLATFORM DEPLOYMENT

Windows

Linux

Containers



Sharad K Singh

RUNTIME & SDK ROLE

Runtime executes

SDK builds & publishes

Sharad K Singh

DAY 4 PRACTICE

Create web app

Add middleware

Publish to IIS

Sharad K Singh

INTRODUCTION

THANK YOU

Sharad K Singh