#Coding
VentiVenticinque

.NET Aspire, Clean Code e Clean Architecture

Costruisci applicazioni moderne e scalabili

Francesco Del Re
Lead Execution Manager

Microsoft

AGIC

CONSOFT INFORMATICA

KEY2

LOBRA

LIMEWARE

Agenda

- Obiettivi del Workshop
  - Setup e configurazione ambiente di lavoro
    - Principi del Clean Code & Clean Architecture
    - Introduzione a .NET Aspire
  - Hands-on Coding
- Conclusione e Q&A

Obiettivi del Workshop

☑ Applicare i principi del Clean Code

- Scrivere codice **leggibile**, **manutenibile** e **testabile**.
- Evitare duplicazioni e complessità inutili.

☑ Progettare software modulare e scalabile

- Utilizzo della **Clean Architecture**.
- Separazione dei livelli (Domain, Application, Infrastructure, UI).
- Riduzione delle dipendenze e miglioramento della scalabilità.

☑ Esplorare e comprendere .NET Aspire

- Introduzione e vantaggi per sviluppare applicazioni moderne.
- Architettura e componenti chiave.

☑ Riconoscere il valore della qualità del software

- Un codice pulito e ben organizzato riduce il tempo e i costi di manutenzione.
- Un codice chiaro migliora la collaborazione tra sviluppatori e riduce il rischio di errori.
- Investire nella qualità previene problemi futuri e garantisce un ciclo di sviluppo più efficiente.

Repository ufficiale del Workshop:
https://github.com/sharpcoding-conference/SharpCodingWorkshop-2025

SCAN ME!

Software e configurazioni

1. Clonare il repository

Clona il repository del workshop sul tuo computer utilizzando il seguente comando:

```
git clone https://github.com/sharpcoding-conference/SharpCodingWorkshop-2025.git
```

2. Visual Studio 2022

Visual Studio è l'IDE che utilizzeremo per lo sviluppo durante il workshop.

- Vai su https://visualstudio.microsoft.com/it/downloads/
- Seleziona la versione Community o una delle altre versioni a pagamento.

3. Podman (+ Podman Desktop)

Podman è un tool open-source per la gestione di container e pod compatibile con Docker, ma senza necessità di un demone centralizzato.

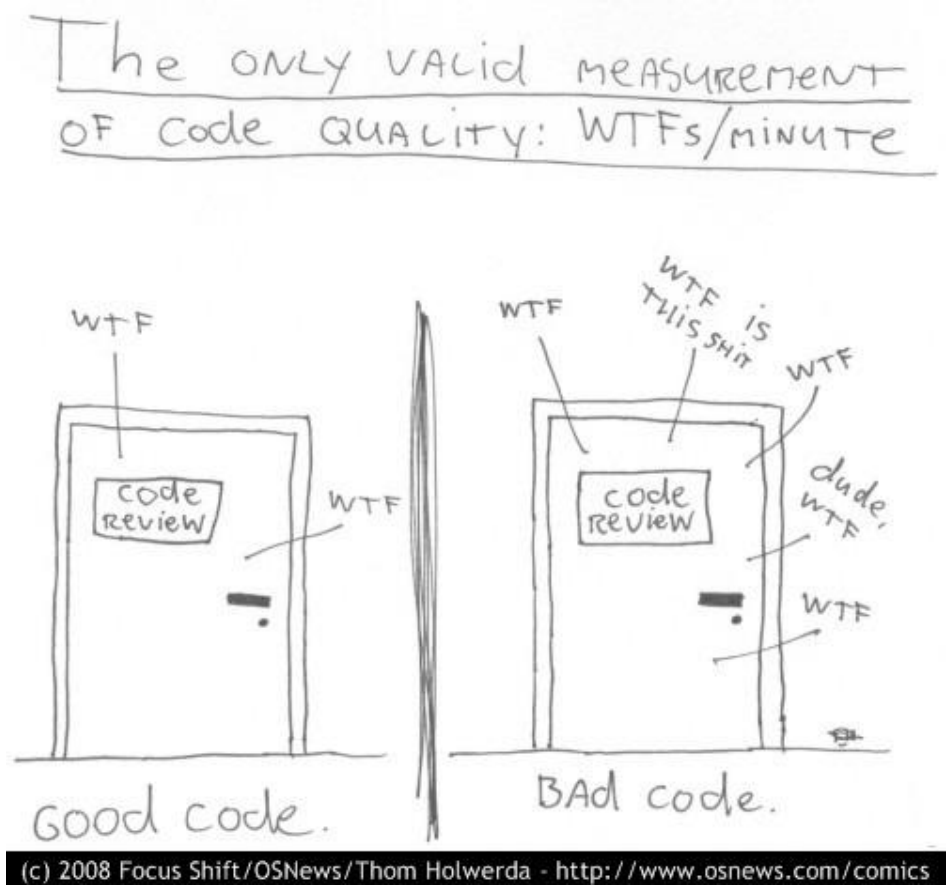
- Vai su https://podman.io/
- Seleziona la versione compatibile per il sistema operativo di riferimento.
- Assicuratevi che sia abilitata la virtualizzazione sul vostro OS.

4. Docker (in alternativa a Podman)

Docker è una piattaforma che consente di creare, distribuire ed eseguire applicazioni in container.

- Vai su https://www.docker.com/products/docker-desktop/
- Seleziona la versione compatibile per il sistema operativo di riferimento.

Cosa significa Clean Code?



Principi Fondamentali del Clean Code

"Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.“ – **Martin Fowler**

READABLE	Il codice deve essere chiaro e comprensibile
SIMPLE	Evita complessità inutili, punta a soluzioni dirette e intuitive
CLEAN	Mantieni il codice ordinato e privo di elementi ridondanti
EFFICIENT	Scrivi codice ottimizzato per prestazioni e risorse senza sacrificare la chiarezza
EXTENDABLE	Progetta soluzioni modulari e flessibili, facili da adattare ed espandere in futuro

https://engineering87.github.io/2025/02/13/clean-code.html

Meaningful Naming

Good naming conventions improve code readability and maintainability.

Bad Example:

```
var d = DateTime.Now;
var p = new Person();
```

Good Example:

```
var currentDate = DateTime.Now;
var customer = new Person();
```

Avoiding Magic Numbers and Strings

Hardcoded values make code difficult to understand and maintain.

Bad Example:

```
if (status == 1)
{
    Console.WriteLine("Active");
}
```

Good Example:

```
if (status == Status.Active)
{
    Console.WriteLine("Active");
}
```

Single Responsibility Principle

A class should have only one reason to change.

Bad Example:

```
public class ReportService
{
    public void GenerateReport()
    {
        Console.WriteLine("Generating report...");
    }

    public void SaveReportToDatabase()
    {
        Console.WriteLine("Saving report...");
    }
}
```

Good Example:

```
public class ReportGenerator
{
    public string Generate()
    {
        return "Report generated.";
    }
}

public class ReportRepository
{
    public void Save(string report)
    {
        Console.WriteLine($"Saving report: {report}");
    }
}
```

Proper Exception Handling

Avoid generic exceptions and provide meaningful error messages.

Bad Example:

```
try
{
    var result = 10 / divisor;
}
catch (Exception ex)
{
    Console.WriteLine("Something went wrong.");
}
```

Good Example:

```
if (divisor == 0)
{
    Console.WriteLine("Cannot divide by zero.");
}
else
{
    try
    {
        var result = 10 / divisor;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred: {ex.Message}");
    }
}
```

Dependency Injection

Dependency Injection is a software design pattern that promotes loose coupling, making code more maintainable. Instead of a class creating its own dependencies, they are injected from the outside, often using an IoC.

Bad Example:

```
public class UserService
{
    private Database _database = new Database();

    public void SaveUser(User user)
    {
        _database.Save(user);
    }
}
```

Good Example:

```
public class UserService
{
    private readonly IDatabase _database;

    public UserService(IDatabase database)
    {
        _database = database;
    }

    public void SaveUser(User user)
    {
        _database.Save(user);
    }
}
```

Using LINQ for Cleaner Code

LINQ simplifies collection operations, improving readability and reducing boilerplate code when filtering, sorting, and transforming data.

Bad Example:

```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6 };
List<int> evenNumbers = new List<int>();

foreach (var number in numbers)
{
    if (number % 2 == 0)
    {
        evenNumbers.Add(number);
    }
}
```

Good Example:

```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6 };
List<int> evenNumbers = numbers
                        .Where(n => n % 2 == 0)
                        .ToList();
```

Early Termination

Using early termination improves code readability by reducing nested conditions and making the intent clear.

Bad Example:

```
public void ProcessOrder(Order order)
{
    if (order != null)
    {
        if (order.IsPaid)
        {
            if (!order.IsShipped)
            {
                ShipOrder(order);
            }
        }
    }
}
```

Good Example:

```
public void ProcessOrder(Order order)
{
    if (order == null) return;
    if (!order.IsPaid) return;
    if (order.IsShipped) return;

    ShipOrder(order);
}
```

User request and Response Suffixes for DTOs

When designing APIs and web services, using suffixes like Request and Response for Data Transfer Objects (DTOs) is a best practice that enhances code clarity and maintainability.

Bad Example:

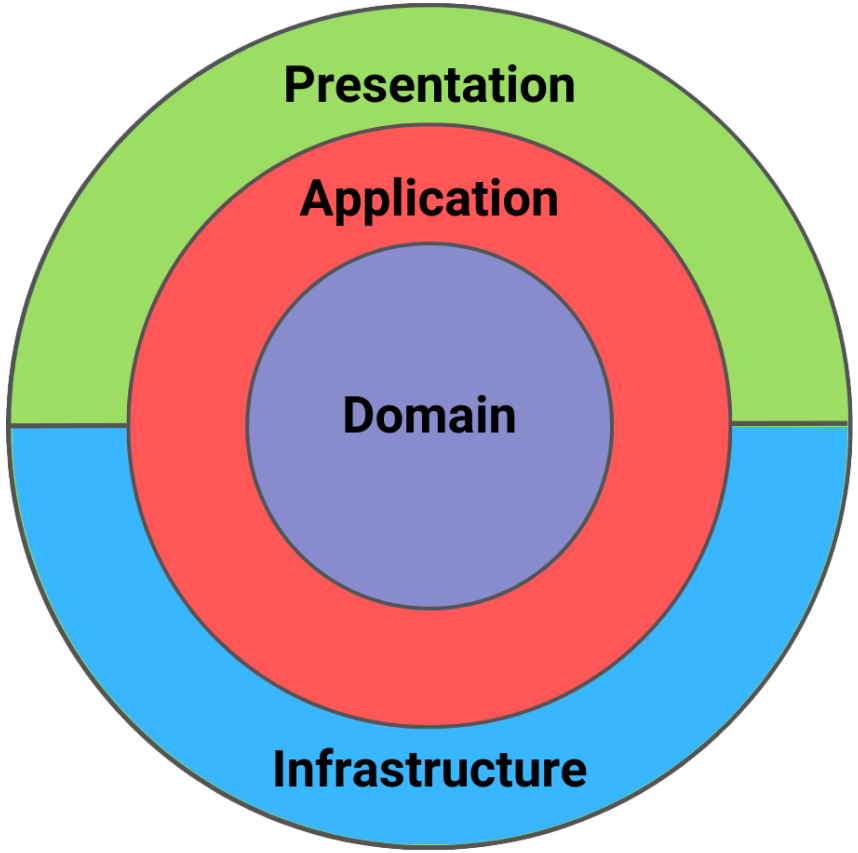
```
public class UserDTO
{
    public string UserName { get; set; }
    public string Email { get; set; }
}
```

Good Example:

```
public class CreateUserRequest
{
    public string UserName { get; set; }
    public string Email { get; set; }
}

public class CreateUserResponse
{
    public int UserId { get; set; }
    public string UserName { get; set; }
}
```

Cos'è la Clean Architecture?



DOMAIN (Core) — Contiene la logica di business

- ENTITIES
- VALUE OBJECTS
- ENUMS
- REPOSITORIES

APPLICATION — Contiene gli Use Cases

- USE CASES
- QUERIES
- COMMANDS

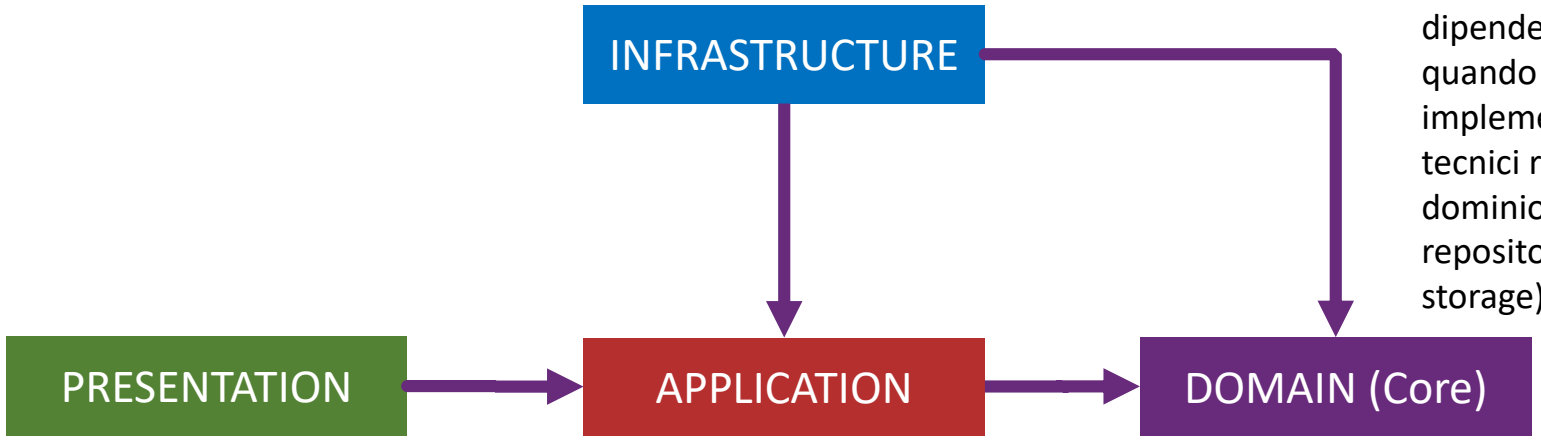
PRESENTATION — Rappresenta il punto di ingresso

- API
- SERVICES
- ASP.NET
- MIDDLEWARE

INFRASTRUCTURE — Contiene qualsiasi risorsa esterna

- DATABASES
- STORAGE
- MESSAGE BROKER

Dipendenze nella Clean Architecture

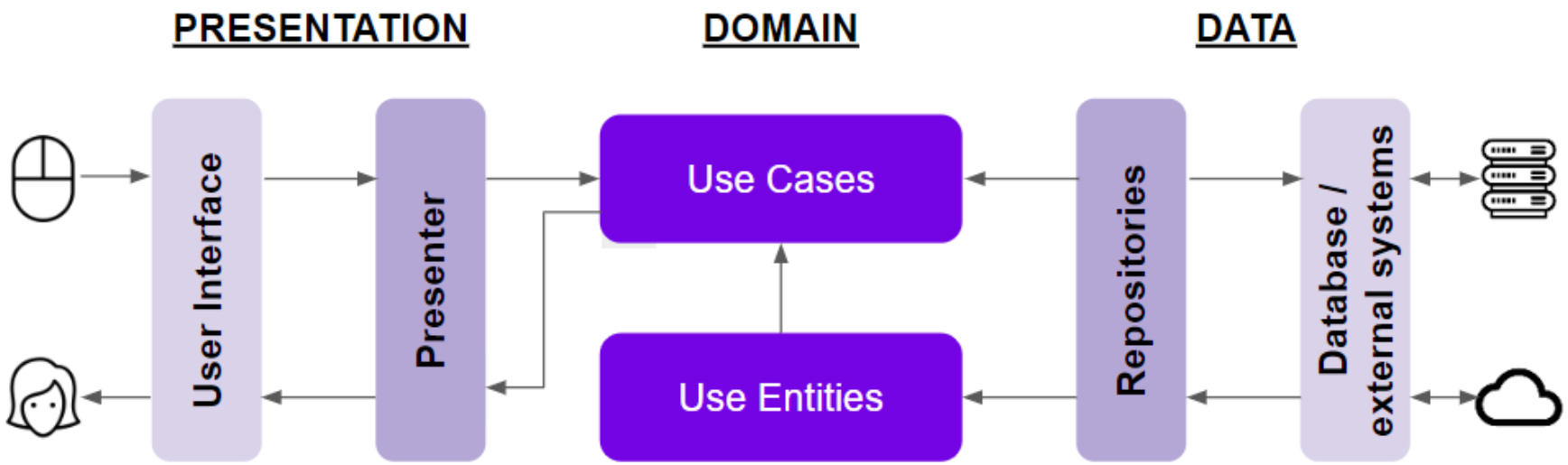


Infrastructure può dipendere dal **Domain** quando deve implementare dettagli tecnici relativi ai dati del dominio (es. persistenza, repository, mapping, storage).

✓ **Regola chiave**: le dipendenze devono sempre puntare verso l'interno.

- **Indipendenza dal framework:** Il sistema non è vincolato a un database o tecnologia specifica.
- **Testabilità:** Il core dell'applicazione può essere testato senza dipendenze esterne.
- **Manutenibilità:** Le tecnologie possono essere sostituite senza impattare la logica di business.

DDD e Clean Architecture: Sinergia per un software robusto



Hands-On Coding: Creazione di una Clean Architecture

In questa sessione pratica, svilupperemo un'applicazione seguendo il modello di **Clean Architecture** basato sul repository di Jason Taylor.

Obiettivo:
Imparare a creare rapidamente un progetto **ready to use** seguendo la Clean Architecture, utilizzando il template di Jason Taylor. Questo approccio automatizza la configurazione iniziale, consentendo di partire con una base solida e scalabile, pronta per essere personalizzata e sviluppata secondo le best practices del Clean Code.

Specifiche dell'Applicazione Distribuita - Gestione Webinar Online

Durante questo workshop, svilupperemo un'applicazione web con **.NET Aspire** per la gestione dei **webinar**, degli **utenti** e delle **prenotazioni**.

Principi e Struttura:

- **Clean Code**

Scrivere codice leggibile, manutenibile ed efficiente seguendo le best practices di sviluppo.

- **Clean Architecture**

Utilizzo della Clean Architecture per garantire separazione delle logiche e modularità del sistema.

Caratteristiche Principali:

- **Applicazione Web in Blazor WebAssembly**

Interfaccia utente dinamica e interattiva per la gestione dei webinar e delle prenotazioni.

- **Microservizio REST per le API**

Servizio separato e scalabile che espone le API per gestire i webinar, gli utenti e le prenotazioni.

- **Persistenza su PostgreSQL**

Database relazionale per memorizzare i dettagli dei webinar, degli utenti e delle prenotazioni.

- **Cache con Redis**

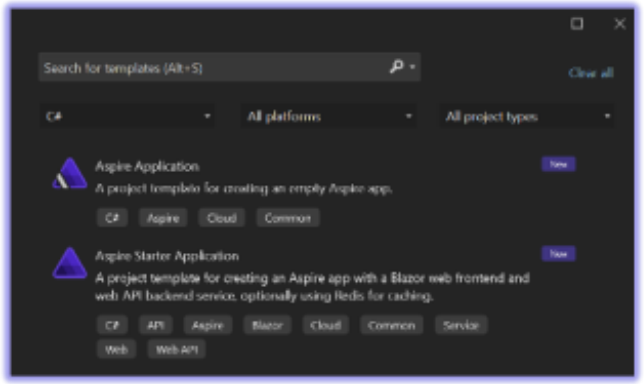
Ottimizzazione delle performance per operazioni frequenti e accesso rapido ai dati.

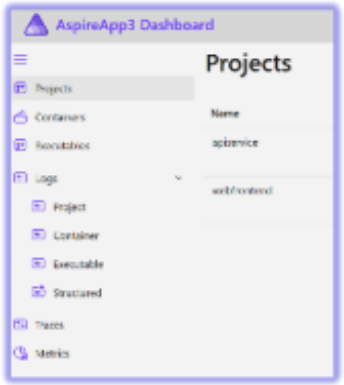
ONE DOES NOT SIMPLY
DEVELOPING DISTRIBUTED APPLICATIONS
imgflip.com

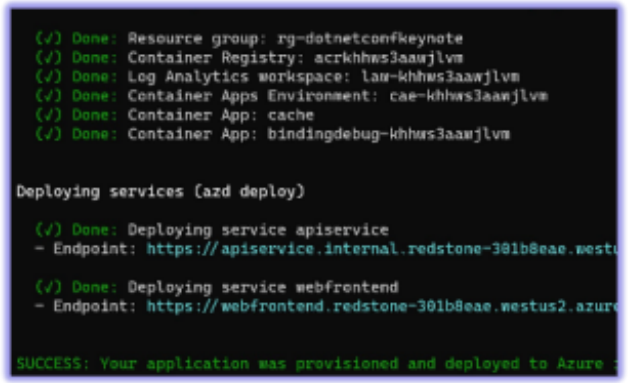
.NET Aspire

A cloud ready stack for building observable, production ready, distributed applications

- Developer Dashboard
- Orchestration
- Components
- Service Discovery
- Deployment







Easy to get started

- SDK Workload
- Templates
- Components

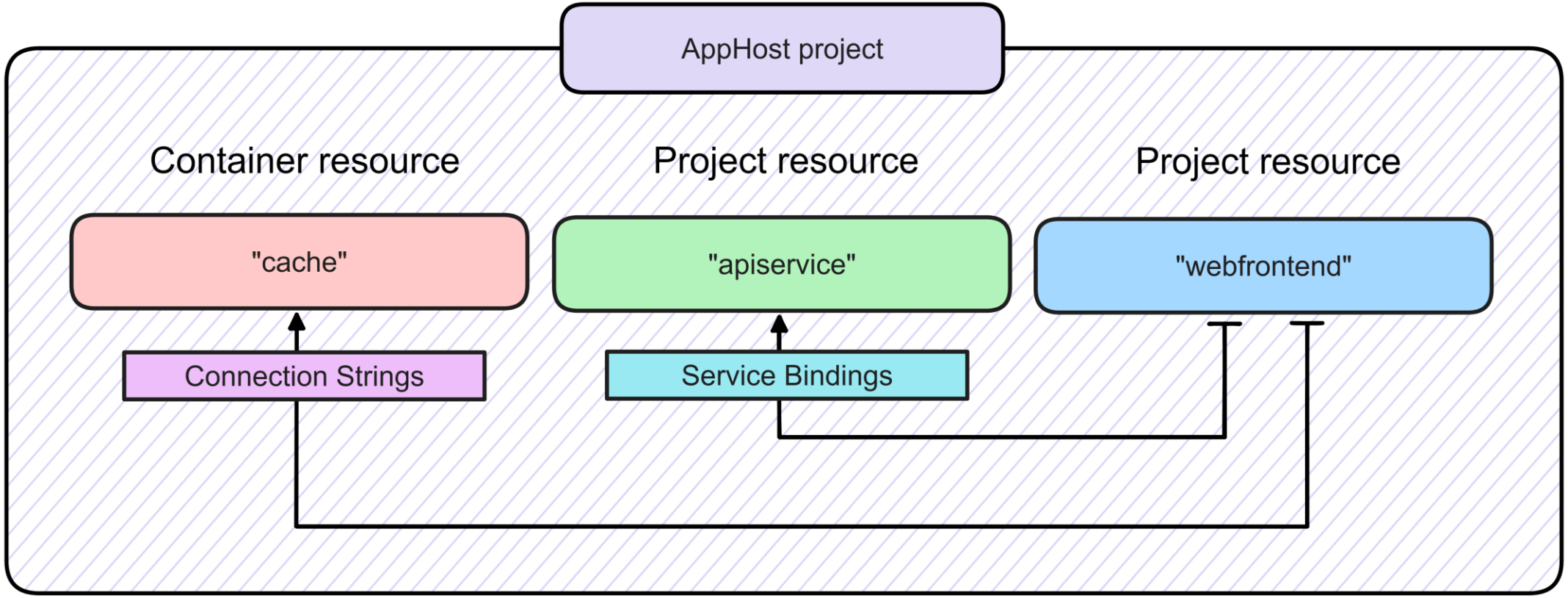
Easy to build

- Service Discovery
- Developer Dashboard
- Logs, Metrics, Distributed Traces

Easy to deploy

- Single command run
- App topology in C#
- Cloud Deployment

Orchestrazione in fase di sviluppo



Questions & Answers

```
{
  “Informations": {
    "name": "Francesco Del Re",
    "role": “Lead Execution Manager“,
    "education": "Engineer in Computer Science"
},
  “Social": {
    "mail": "francesco.delre@protonmail.com",
    "linkedin": "https://www.linkedin.com/in/francesco-delre",
    "github": "https://github.com/engineering87 ",
   }
}
```