# Stop Building Boilerplate. Start Building Your App.

A Production-Ready .NET Template for Modern Web Applications

.NET 10.0+

ASP.NET Core 10.0+

Entity Framework Core 10.0+

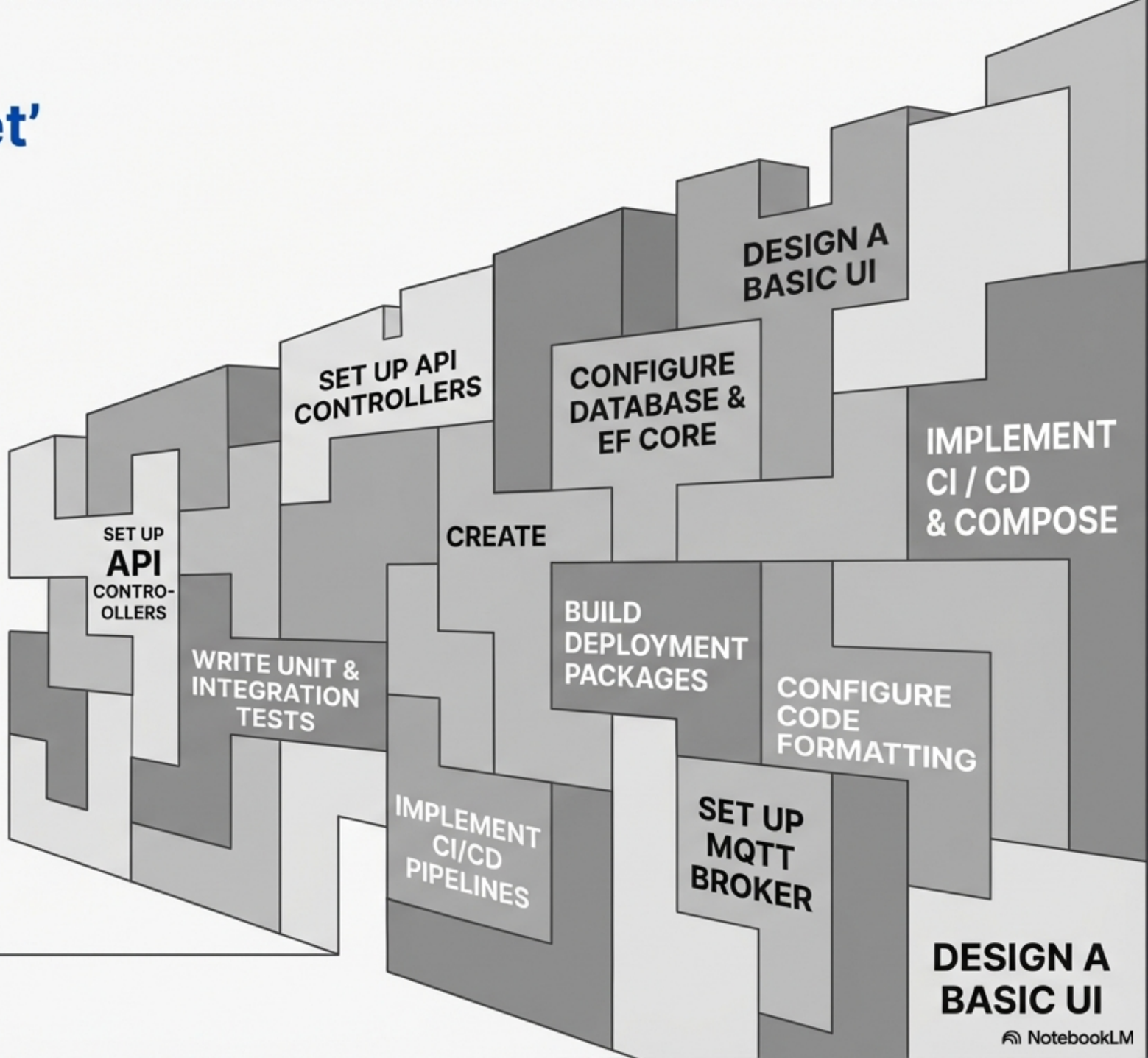
MQTTnet 5.0+

Bootstrap 5.3+

License: MIT

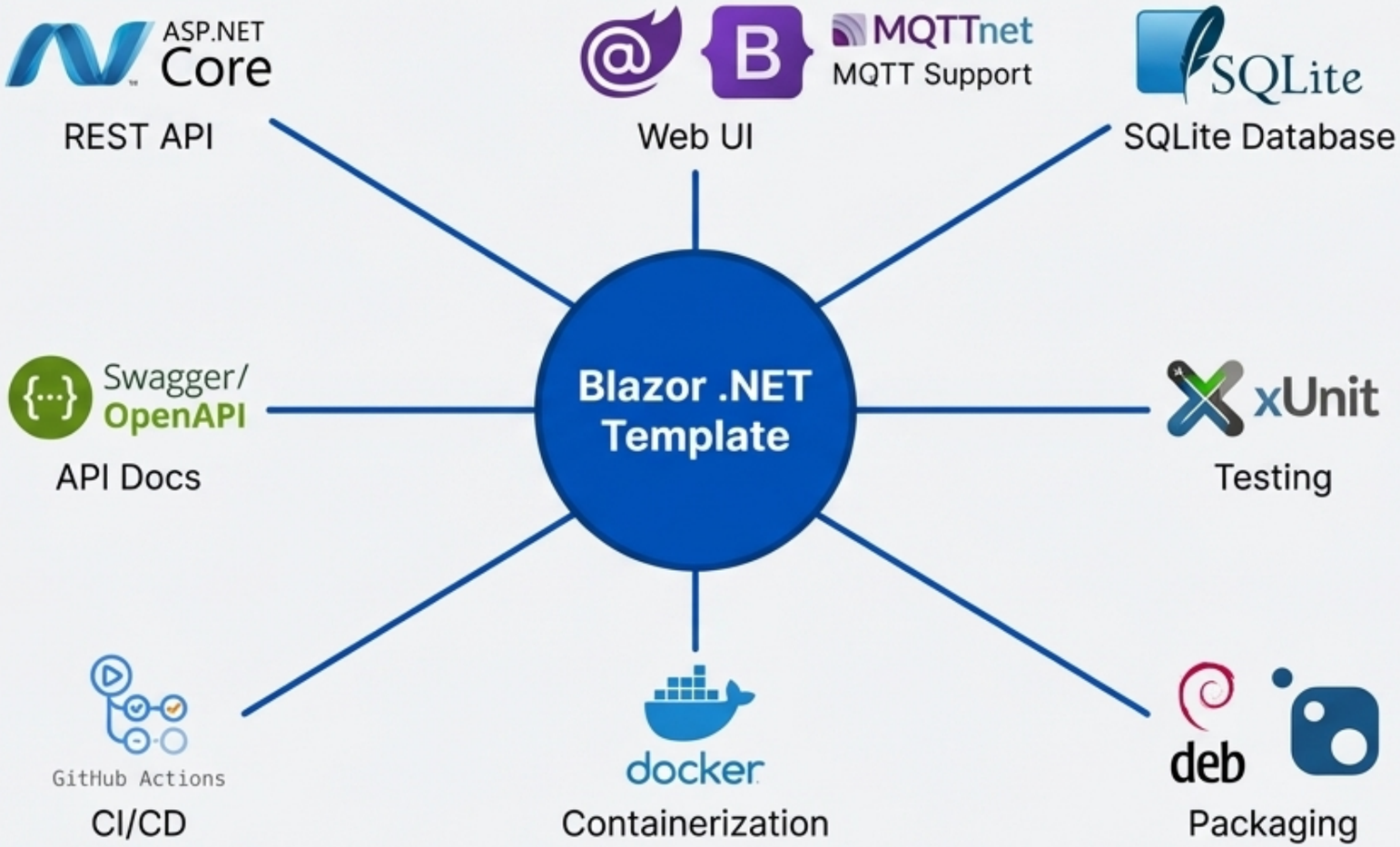
# The ‘Boilerplate Gauntlet’ of a New Project

Starting a new .NET application means days, not hours, of setup. Before you write a single line of business logic, you’re wrestling with configuration.



# Your Production-Ready Foundation, Solved.

The Blazor .NET Template integrates the entire stack into a cohesive, ready-to-run foundation. It's not just code; it's a complete development and deployment ecosystem.



# A Comprehensive Toolkit for the Entire Application Lifecycle

## Core Features

- **REST API**: Full CRUD operations via ASP.NET Core Web API.
- **MQTT Support**: Pub/sub messaging with MQTTnet.
- **SQLite Database**: Pre-configured with Entity Framework Core & migrations.
- **OpenAPI/Swagger**: Interactive API documentation out-of-the-box.
- **Web Interface**: Blazor Server UI built with Bootstrap 5.
- **Comprehensive Tests**: Includes xUnit unit tests and WebApplicationFactory integration tests.

## Developer Experience

- **VS Code Ready**: Includes launch and task configurations.
- **Hot Reload**: Use `dotnet watch` for rapid development.
- **Git Hooks**: Pre-commit hook for automatic code formatting.
- **Setup Scripts**: Automated dependency installation.
- **Code Formatting**: Enforced via `.editorconfig`.

## Production Deployment

- **Docker Images**: Multi-architecture images for `amd64` & `arm64`.
- **Docker Compose**: For local development with an included MQTT broker.
- **DEB Packages**: For easy installation on Ubuntu/Debian.
- **NuGet Packages**: Deploy as a .NET Global Tool on Windows.
- **Systemd Service**: Production-ready service configuration.

# The 5-Minute Challenge: From Clone to Running App

Go from a git clone to a fully functional application with a database, API, and UI in minutes. Choose the path that fits your workflow.

## Step 1: Clone

```
git clone https://github.com/mlmdevs/blazor-net-template.git
cd blazor-net-template
```

## Path 1: Docker (Recommended)

### The All-in-One Environment

Starts the application AND a pre-configured MQTT broker. Ideal for a clean, consistent dev setup.

```
./docker-dev.sh start
```

## Path 2: .NET CLI

### The Native Experience

Use the .NET SDK directly on your machine.

```
dotnet restore
```

```
dotnet run --project src/BlazorNetApp.Api
```

# A Closer Look: The Docker Development Environment

The `docker-dev.sh` script provides a complete, isolated development environment with a single command.

## The Command

```
# Linux/macOS
./docker-dev.sh start

# Windows (PowerShell)
.\\docker-dev.ps1 -Command start
```

## What It Does

- **Builds** the .NET application Docker image.
- **Starts** the application container.
- **Starts** a Mosquitto MQTT broker container.
- **Creates** persistent volumes for the database and MQTT data.

## Access Your App

- **Web UI**: `http://localhost:5000`
- **Swagger API**: `http://localhost:5000/swagger`
- **MQTT Broker**: `localhost:1883`

---

Also includes: `logs`, `stop`, `clean`, `build`

# Step 1: Make It Your Own with a Single Command

Transition from the ‘blazor-net-app’ template name to your unique project name effortlessly. The automated script handles all the tedious work for you.

### Bash (Linux/macOS)

**Inter Bold**

```
./rename-app.sh my-cool-app
```

### PowerShell (Windows)

**Inter Bold**

```
.\\Rename-App.ps1 -NewAppName my-cool-app
```

### The Automation: What the script updates

- ✓ Renames all directories and files.
- ✓ Updates namespaces in all C# files.
- ✓ Updates all configuration files.
- ✓ Updates all deployment scripts.
- ✓ Updates documentation files.

**Note:** The app name should be in kebab-case (e.g., `my-cool-app`).

# Step 2: An Efficient and Polished Development Loop

The template is engineered to keep you in the flow, with tools that accelerate development and maintain high code quality standards automatically.

## VS Code Integration

Open the project and press `F5` to start debugging immediately. Recommended extensions are suggested on first open.

## Pre-Commit Formatting

A git hook automatically runs `dotnet format` before every commit, ensuring consistent code style across the project. Install with `./.githooks/install.sh`.

## Database Migrations

Manage your database schema with standard EF Core commands.

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

## Comprehensive Testing

Run all unit and integration tests with a single command.

```
dotnet test
```

# Under the Hood: A Fully Functional API and MQTT Interface

## REST API Endpoints

Access via HTTP and explore with Swagger at `/swagger`.

**GET** `/api/todoitems`

**GET** `/api/todoitems/{id}`

**POST** `/api/todoitems`

**PUT** `/api/todoitems/{id}`

**DELETE** `/api/todoitems/{id}`

## MQTT Command & Control

Interact with your application via publish/subscribe messaging.

Topic Structure: `blazor-net-app/command/{action}`

**Example: Create a TODO Item**

Topic: `blazor-net-app/command/create`

```
{
  "title": "New TODO Item",
  "isCompleted": false,
  "correlationId": "request-123"
}
```

Responses are published to `blazor-net-app/response/{correlationId}`.

# Step 3: Deploy to Production with Confidence

Move from development to production seamlessly with multiple, robust deployment options. The template provides automated builds for every major platform and architecture ( `amd64` & `arm64` ).

### Docker (Recommended)

Deploy as a lightweight, multi-architecture container. Images are automatically published to GitHub Container Registry on every release. ( `amd64` & `arm64` .
Ideal for cloud and on-premise hosting.

### DEB Package (Ubuntu/Debian)

Install the application as a self-contained systemd service. Includes the .NET runtime. Perfect for bare-metal or VM deployments on Debian-based systems. Packages are pushed to an APT repository.

### NuGet Package (Windows)

Install and run the application as a .NET Global Tool. The simplest way to run on a Windows machine. Published to GitHub Packages.

# Production Deep Dive: Deploying with Docker

Pull versioned, multi-architecture images directly from the GitHub Container Registry (GHCR) for a standardized production deployment.

## Single Container Deployment

Run the application container.

```
docker pull ghcr.io/mlmdevs/blazor-net-template:latest

docker run -d \
  --name blazor-net-app \
  -p 8080:8080 \
  -v blazor-net-app-data:/app/data \
  -e ASPNETCORE_ENVIRONMENT=Production \
  ghcr.io/mlmdevs/blazor-net-template:latest
```

## Multi-Container with Docker Compose

Use Docker Compose for the app and MQTT broker.

```
version: '3.8'
services:
  blazor-net-app:
    image: ghcr.io/mlmdevs/blazor-net-template:latest
    ports: ["8080:8080"]
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - Mqtt__Broker=mosquitto
    volumes: ["blazor-net-app-data:/app/data"]
    restart: unless-stopped
  mosquitto:
    image: eclipse-mosquitto:2
    ports: ["1883:1883"]
    restart: unless-stopped
```

# Production Deep Dive: Deploying with Packages

Detailed instructions for deploying your application as a **packaged systemd service** or a **global tool**, tailored for specific production environments.

## DEB Package on Ubuntu/Debian

Install as a Systemd Service.

Install the package:

```
sudo dpkg -i blazor-net-app_<version>_<arch>.deb
```

Manage the service:

```
sudo systemctl status blazor-net-app
sudo systemctl restart blazor-net-app
```

```
sudo systemctl restart blazor-net-app
```

View logs:

```
sudo journalctl -u blazor-net-app -f
```

ℹ The database is stored at `/var/lib/blazor-net-app/`.

## NuGet Package on Windows

Install as a .NET Global Tool.

Configure NuGet source (one-time setup):

```
dotnet nuget add source "https://nuget.pkg.github
.com/mlmdevs/index.json" -n GitHub -u <USERNAME>
-p <GITHUB_TOKEN>
```

Install the tool:

```
dotnet tool install --global BlazorNetApp
```

Run the application:

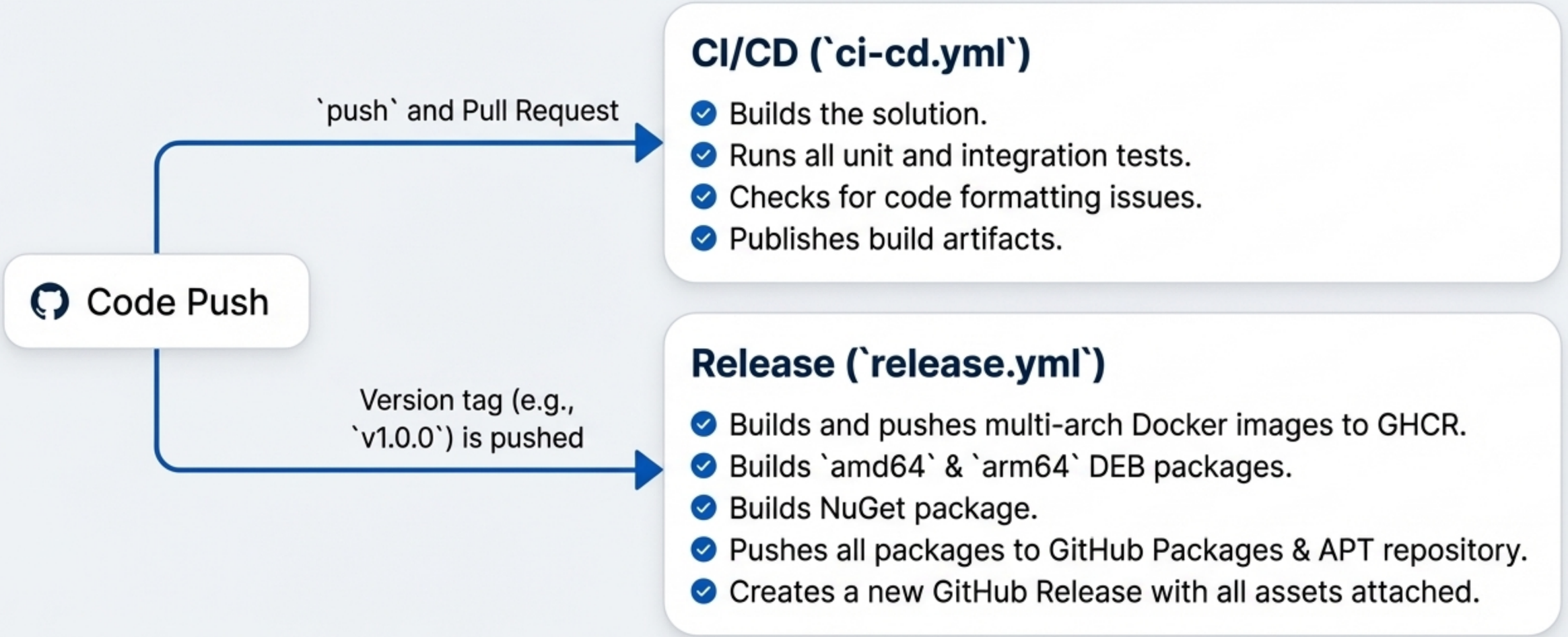
```
blazor-net-app
```

Update the tool:

```
dotnet tool update --global BlazorNetApp
```

# The Automation Engine: Continuous Integration & Delivery

The project is underpinned by two robust GitHub Actions workflows that automate testing, packaging, and releases, ensuring every commit and release is high quality.



# The Blueprint: Project Architecture & Statistics

## Project Structure

```
blazor-net-template/
├── src/            # Main application
├── tests/          # Unit & Integration tests
├── scripts/        # Setup scripts
├── deployment/     # Systemd service, etc.
├── .githooks/      # Pre-commit hooks
├── .github/        # CI/CD workflows
├── Dockerfile
└── docker-compose.yml
```

## Project Statistics

### Feature Checklist

- ✓ REST API
- ✓ MQTT Support
- ✓ Docker Support
- ✓ DEB Packages
- ✓ Rename Scripts

### Test Coverage

**10** total tests passing

- **3** Unit Tests
- **7** Integration Tests

### Lines of Code

- **~4,000** Total
- C# Code: **~1,500**
- Scripts (Shell/PowerShell): **~800**
- Configuration & Docs: **~1,200**

### Dependencies

- Key libraries: EF Core, MQTTnet, Swashbuckle, xUnit.

# Your Accelerator for Building Modern .NET Applications

This template provides the solid, feature-rich foundation you need to move from idea to production faster than ever. Stop wrestling with boilerplate and focus on what truly matters: building your unique application.

## Get the Code

Fork the Repository

github.com/mlmdevs/blazor-net-template

## Read the Docs

Full Documentation

mlmdevs.github.io/blazor-net-template/

## Contribute

Join the Project

Read the `CONTRIBUTING.md` to get started.