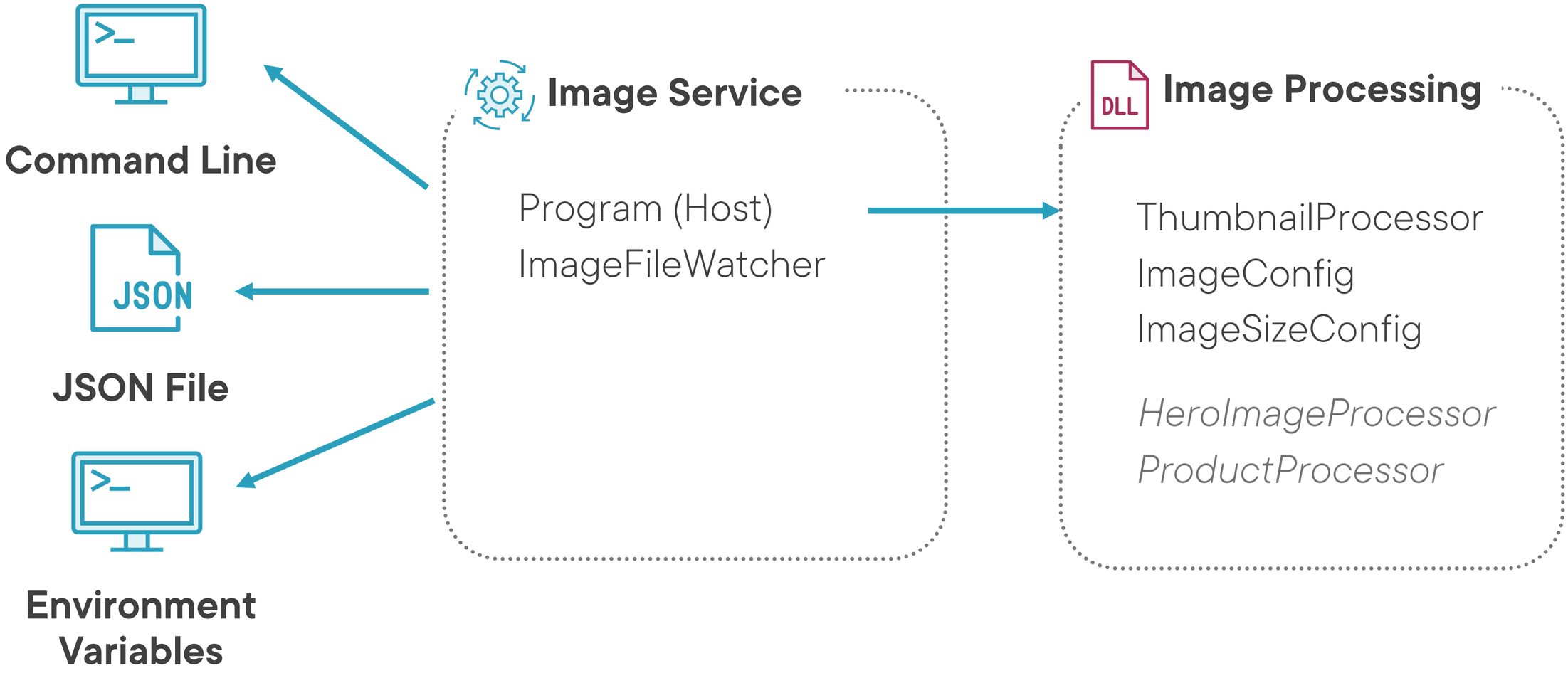
Creating a Configurable Library

Matthew Tester
Technology Consultant

@matttester matthewtester.com

Refactor into a Library



A Configurable Library

Configured by the host

- Independent of specific configuration providers

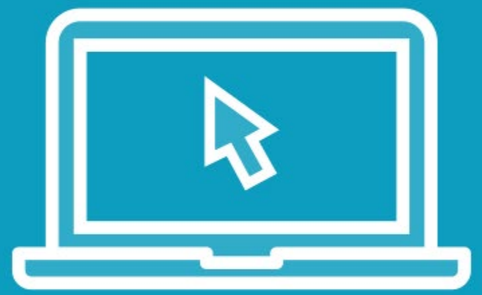
Apply default configuration

- By the host
- By the library

Validate configuration is correct

Registers it's own classes with Dependency Injection container

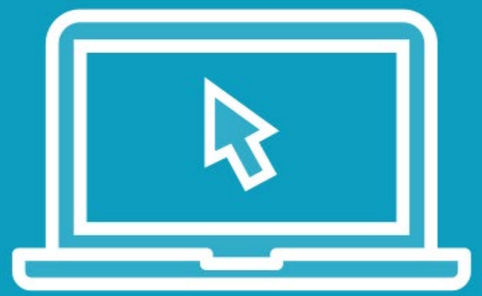
Demo



Making a configurable library

- Carved Rock Image Processing library
- Allow library to register with Dependency Injection container
- .NET Conventions

Demo



Validating configuration

Using OptionsBuilder API

- **Validate method**
- **Data Annotations**
- **Validation service**

OptionsBuilder API Interfaces

```
services
    .AddOptions<ImageConfig>()
    .Configure(imageConfig => {})
    .Bind(configurationSection)
    .Validate(imageConfig => {})
    .PostConfigure(imageConfig => {})
```

IConfigureOptions<ImageConfig>

IConfigureNamedOptions<ImageConfig>

IValidateOptions<ImageConfig>

IPostConfigureOptions<ImageConfig>

Summary

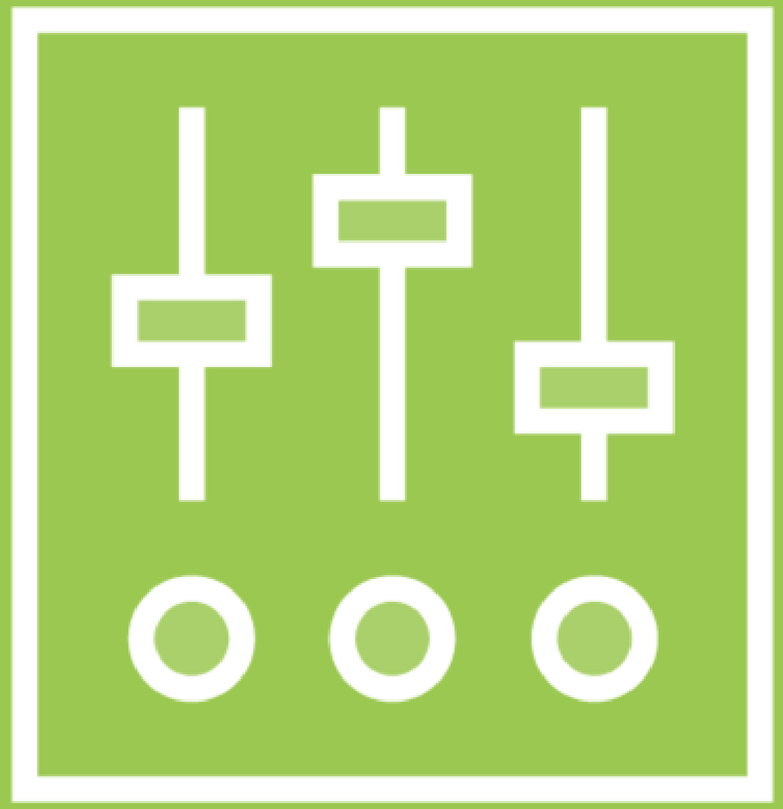
Make a library configurable

- Independent of specific providers
- Patterns and conventions

Allow host and library to set defaults

Validate configuration

- Via Validate on OptionsBuilder API
- Via Data Annotations
- Via IValidateOptions service



Course Complete!

By using configuration, you can build .NET applications that are portable, flexible and more maintainable.

Building Configurable Applications

Console Applications

Configuration API

Use multiple configuration providers

Strongly-typed configuration

Setting defaults

Generic Host

Cross-platform services,
including ASP.NET

Use Dependency Injection

Options Pattern

OptionsBuilder API

Making a library configurable

Secrets in Configuration

Securing Application Secrets in ASP.NET Core

Matthew Tester