

Using the .NET Generic Host

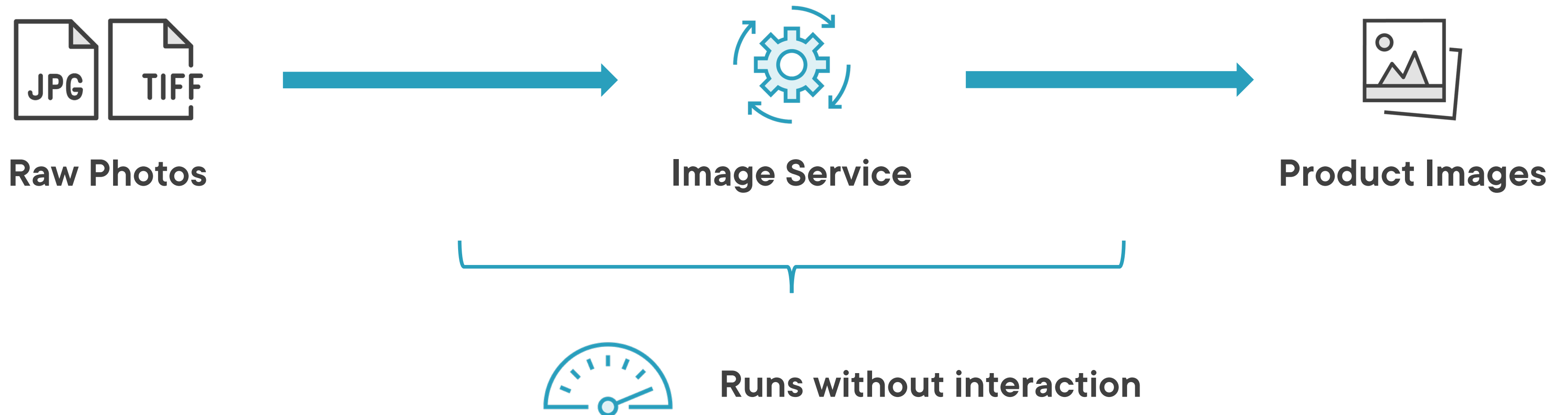


Matthew Tester

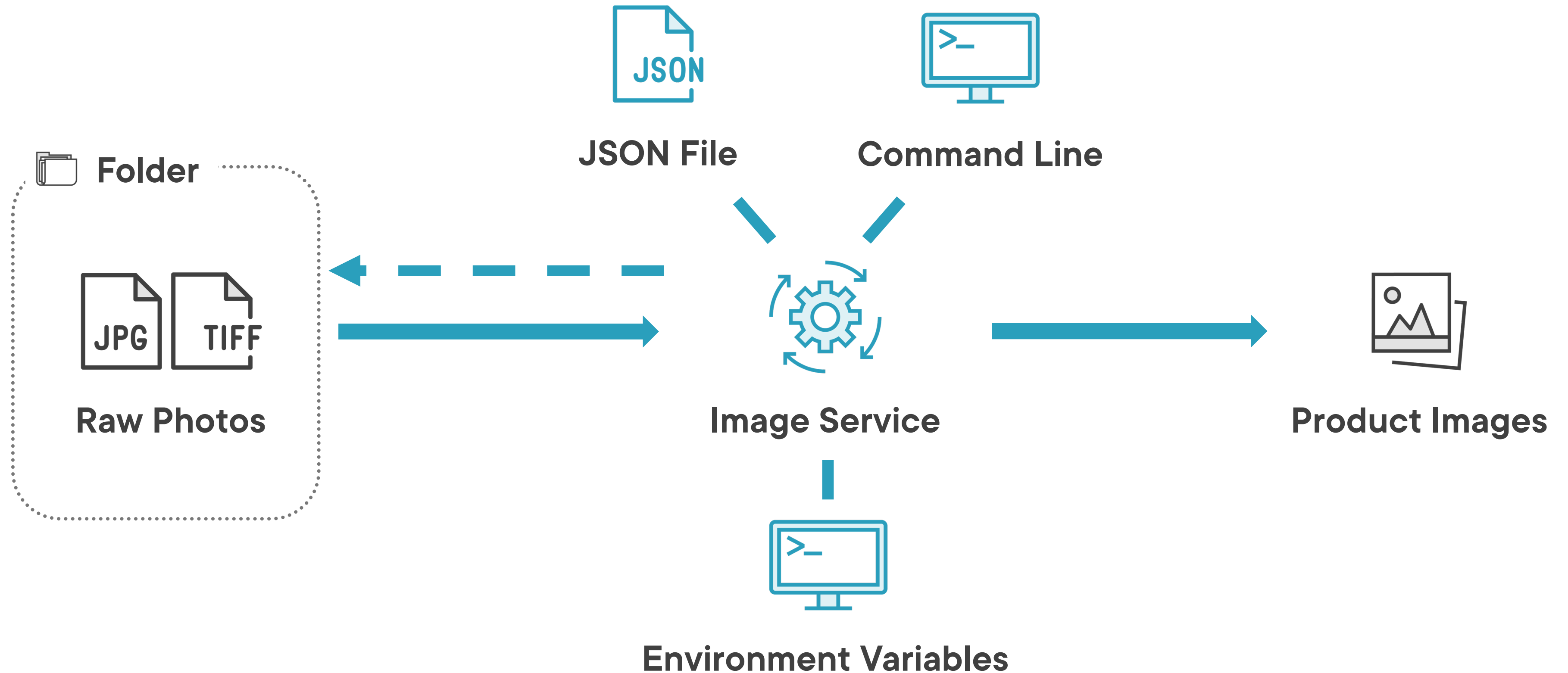
Technology Consultant

@matttester matthewtester.com

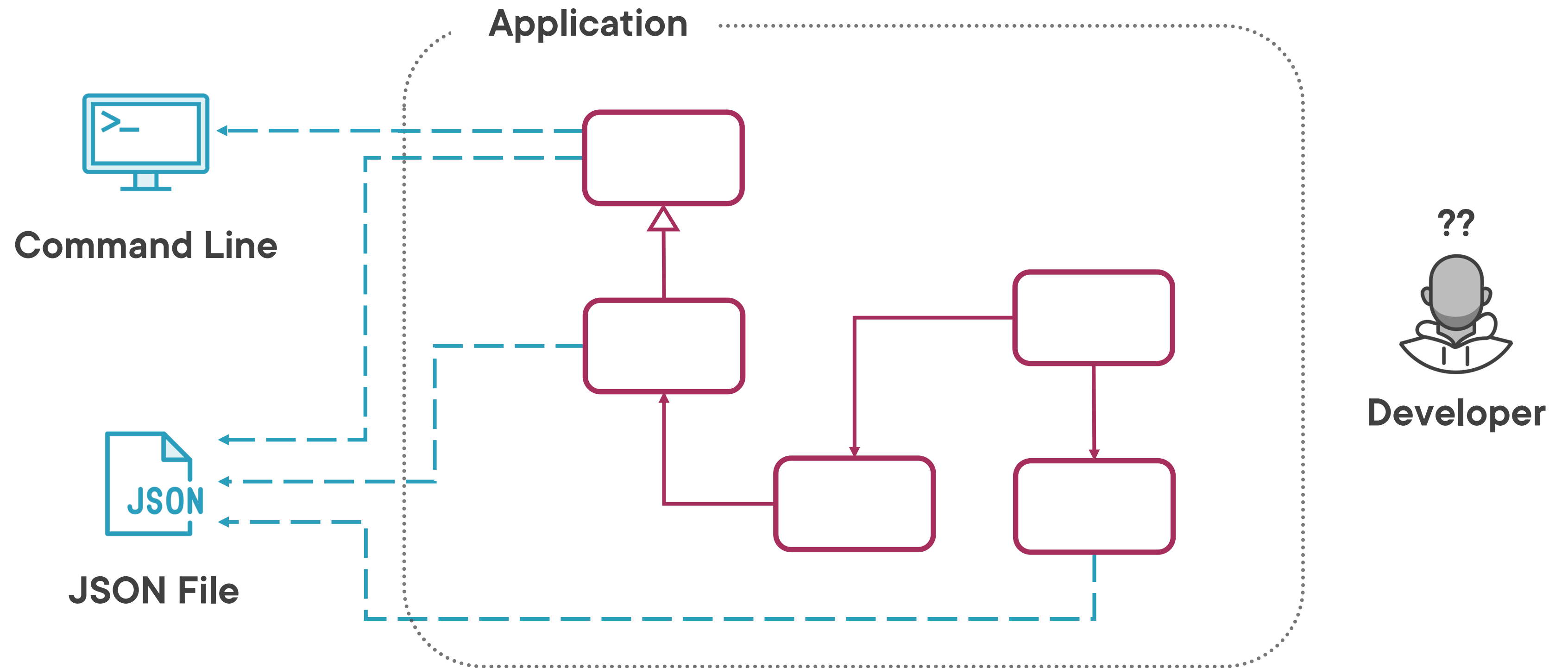
Carved Rock Fitness: Image Process



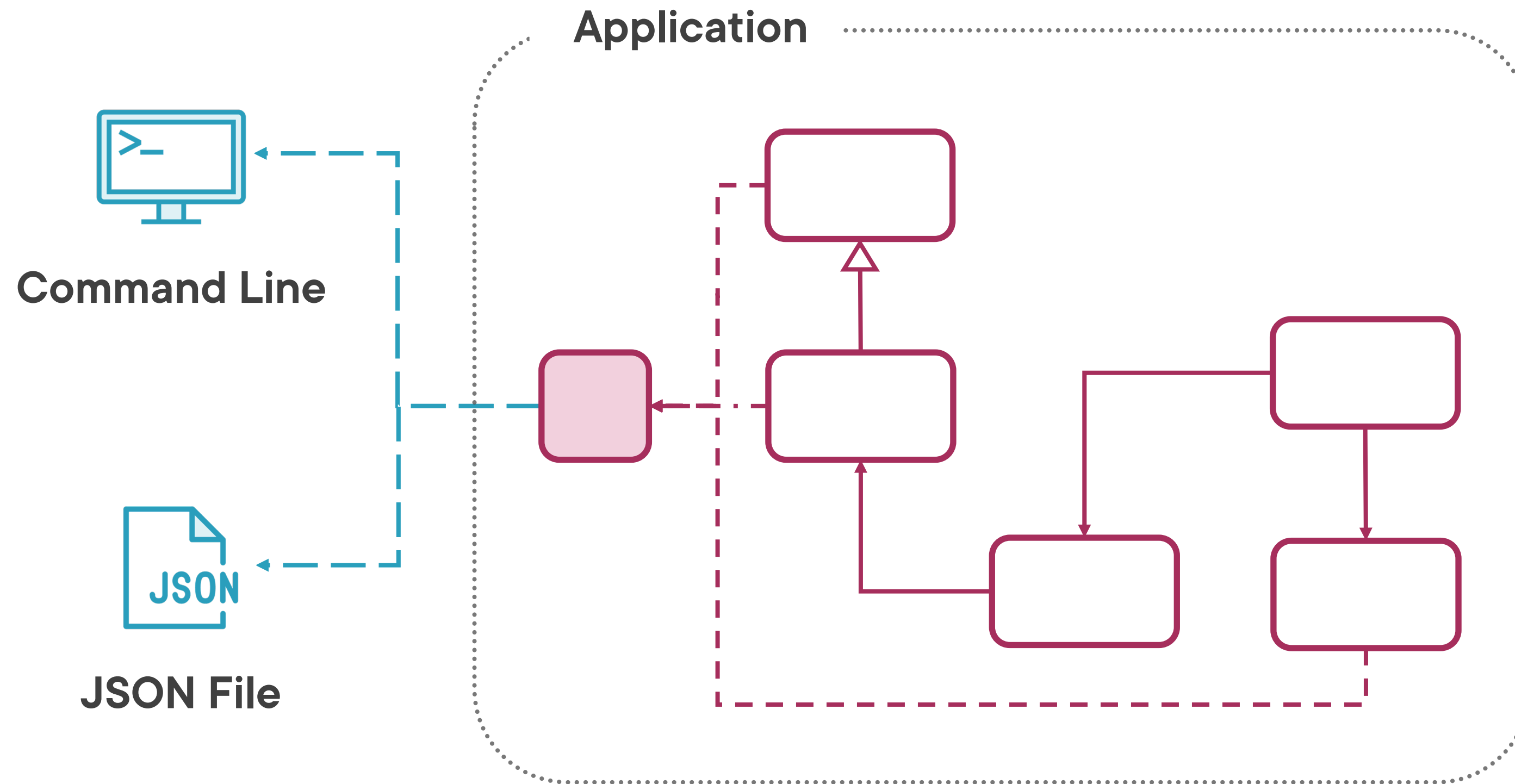
Carved Rock Fitness: Image Process



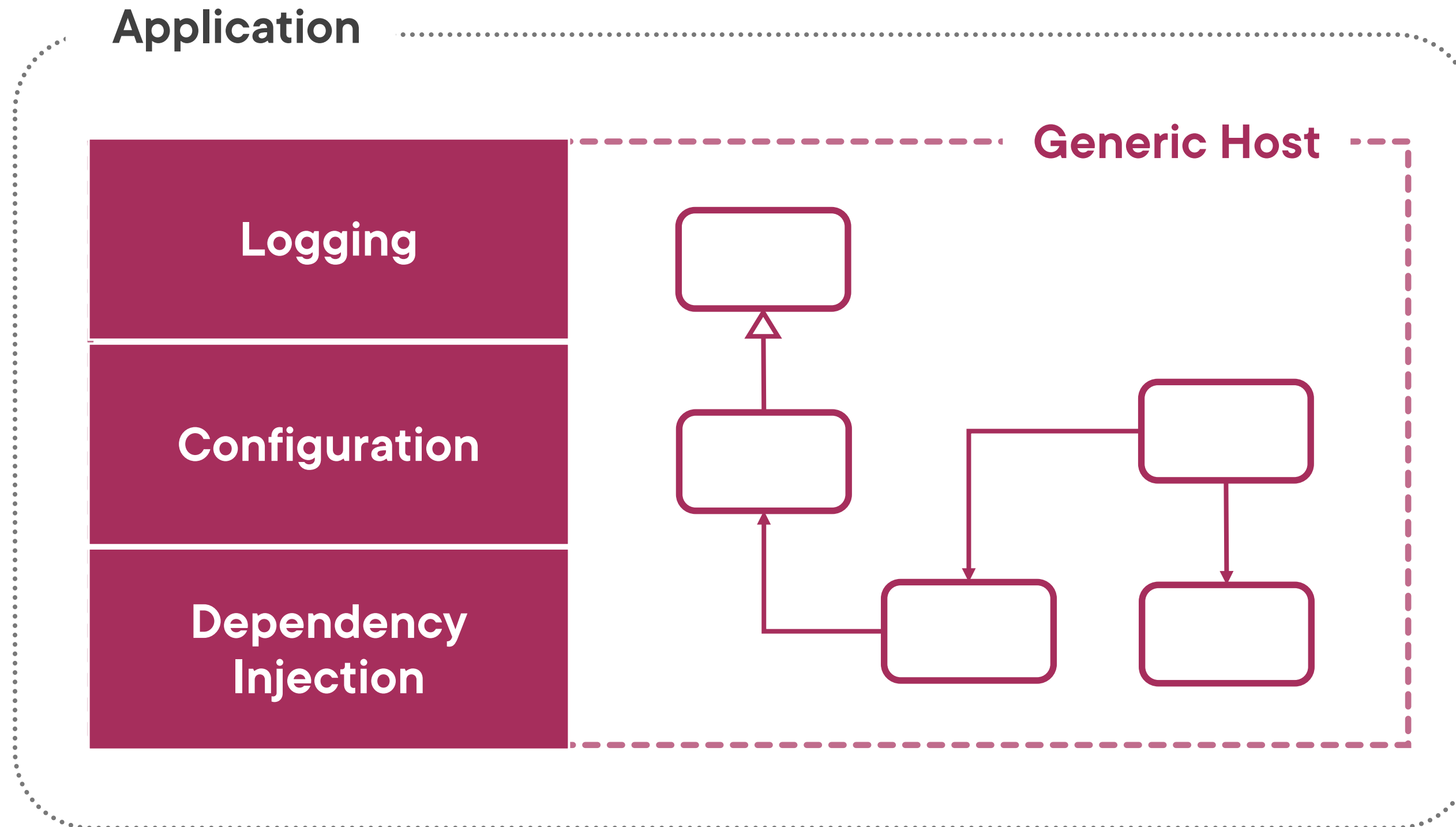
Accessing Configuration



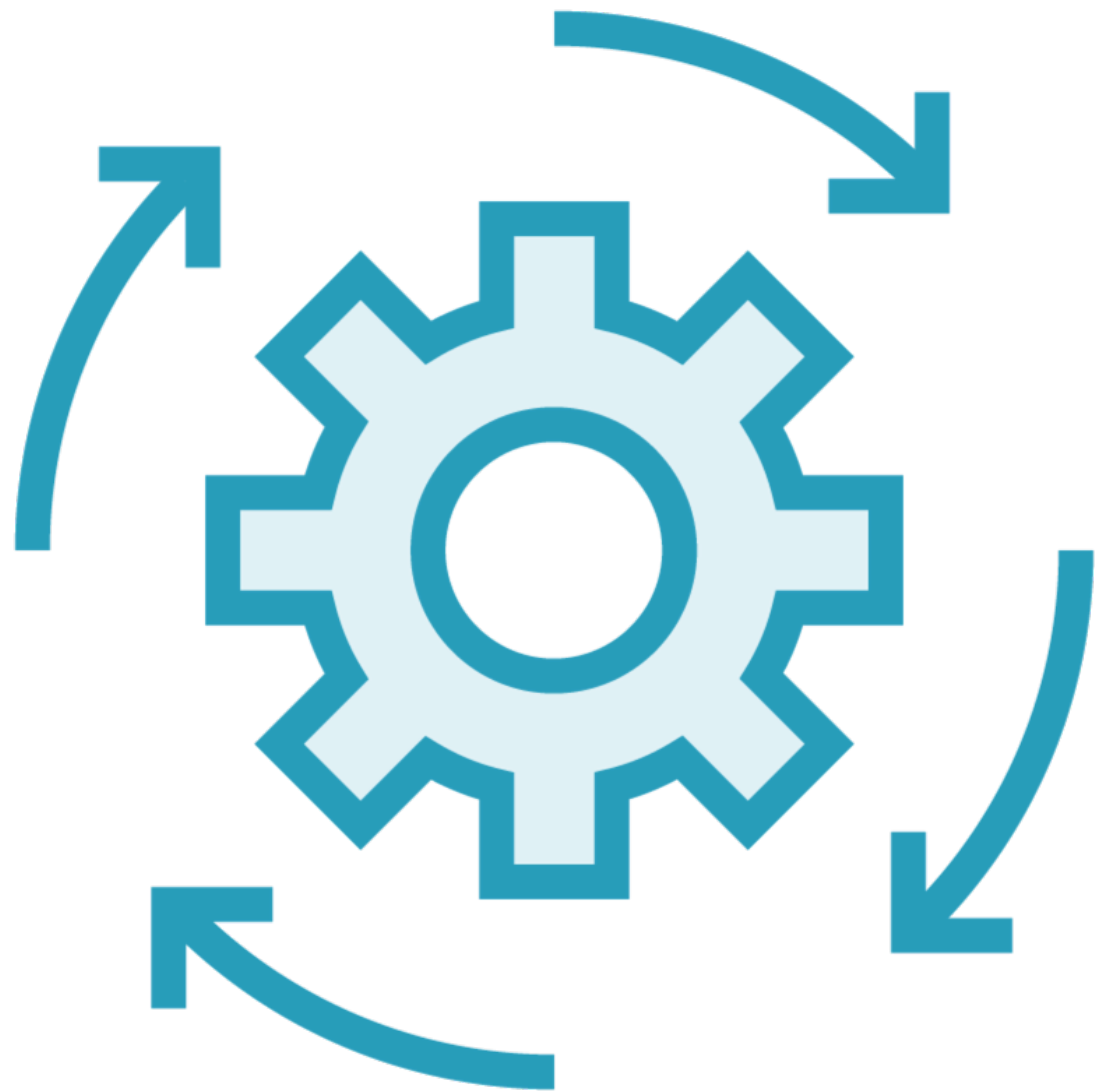
Accessing Configuration



.NET Generic Host



.NET Generic Host



Services

- Windows Services
- Linux Workers
- Long-running console app

Build services using Generic Host

- Graceful start and shutdown
- Cross-platform
- Same model to host ASP.NET applications

Demo



Creating a new service application

- Use the “Worker Service” template
- Explore the classes of the Generic Host

Dependency Injection

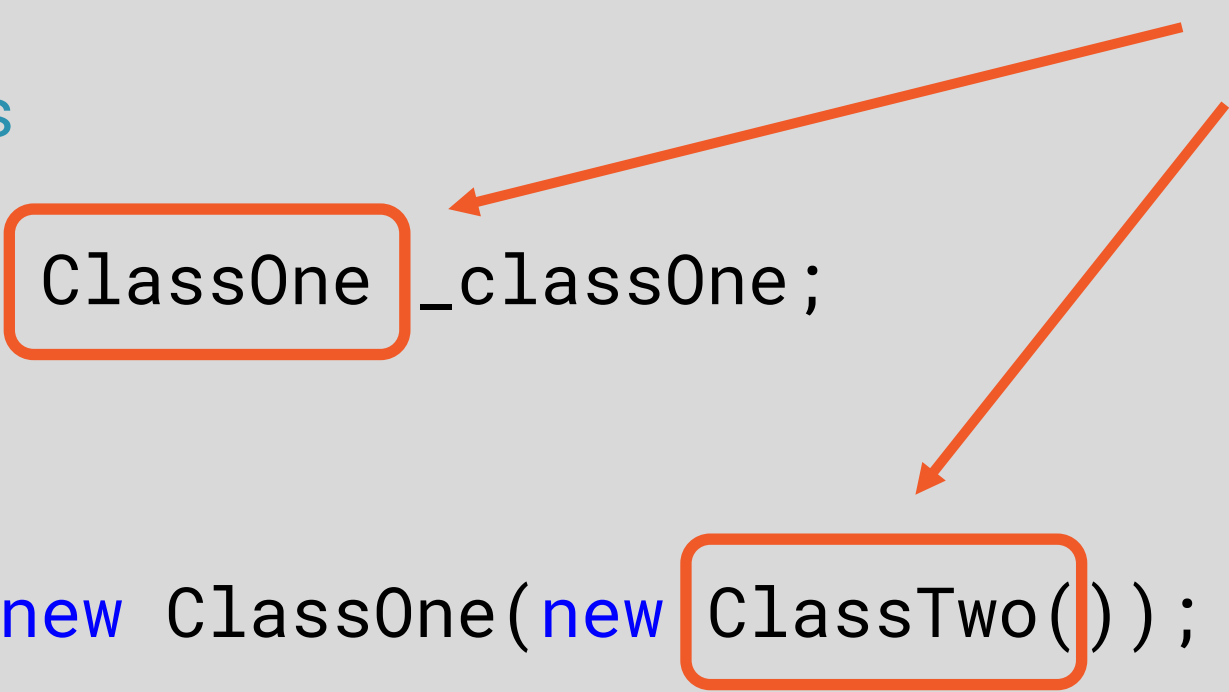
Dependency injection (DI) is a technique in which an object receives other objects that it depends on.

It is one form of the broader technique of inversion of control (IoC).

Dependencies

```
public class MyClass
{
    private readonly ClassOne _classOne;

    public MyClass()
    {
        _classOne = new ClassOne(new ClassTwo());
    }
}
```



The diagram illustrates the dependencies of the `MyClass` class. Two orange arrows originate from the word "Dependencies" and point to the `ClassOne` and `ClassTwo` classes in the code. The `ClassOne` class is highlighted with an orange box in the line `private readonly ClassOne _classOne;`, and the `ClassTwo` class is highlighted with an orange box in the line `_classOne = new ClassOne(new ClassTwo());`.

Tightly Coupled Dependencies

Changes can ripple across the entire application

```
public class MyClass
{
    private readonly ClassOne _classOne;

    public MyClass()
    {
        _classOne = new ClassOne(new ClassTwo("newStringParameter"));
    }
}
```

Tightly Coupled Dependencies

Changes can ripple across the entire application

```
public class MyClass
{
    private readonly ClassOne _classOne;

    public MyClass(ClassOne classOne)
    {
        _classOne = classOne;
    }
}
```

Loosely Coupled Dependencies

Better isolation, reducing risk of cascading changes

```
public class MyClass
{
    private readonly IClassOne _classOne;

    public MyClass(IClassOne classOne)
    {
        _classOne = classOne;
    }
}
```

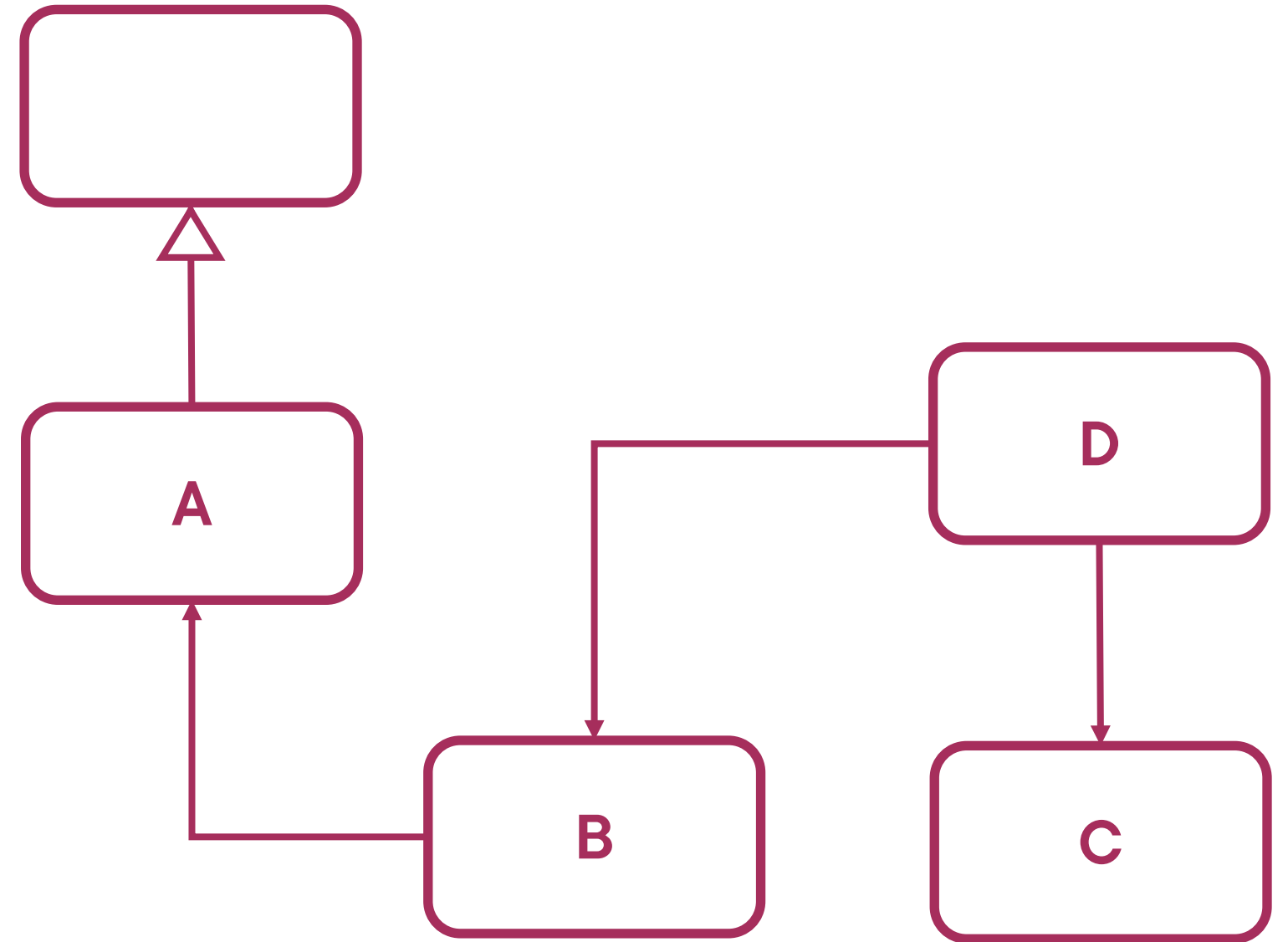
Loosely Coupled Dependencies

Better isolation, reducing risk of cascading changes

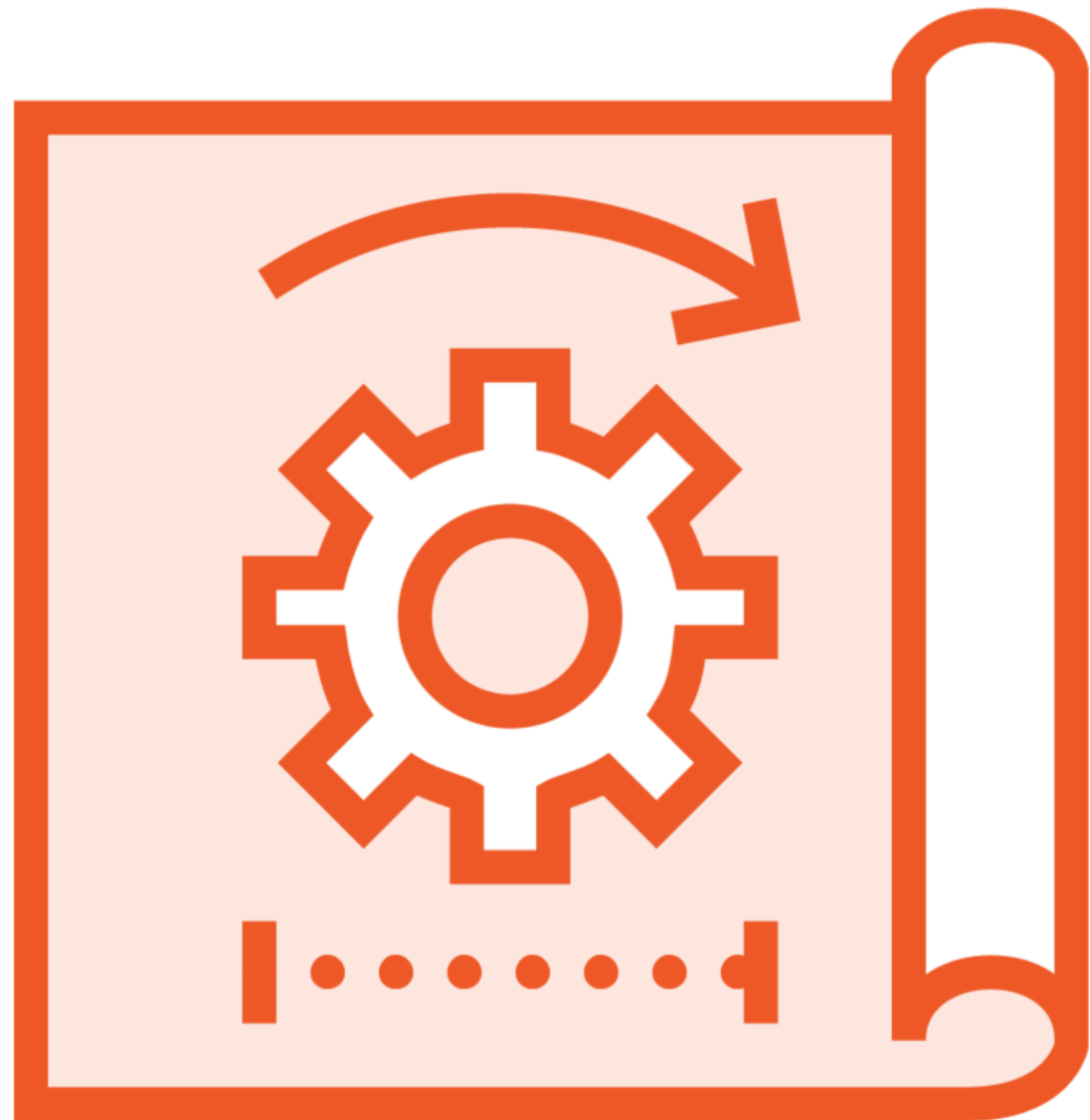
Dependency Injection Container

DI Container

Type
A
B
C
D



Lifetimes



Transient

- New object constructed each time

Singleton

- Constructed once, same instance used

Scoped

- New object constructed once per scope

Dependency Injection Container

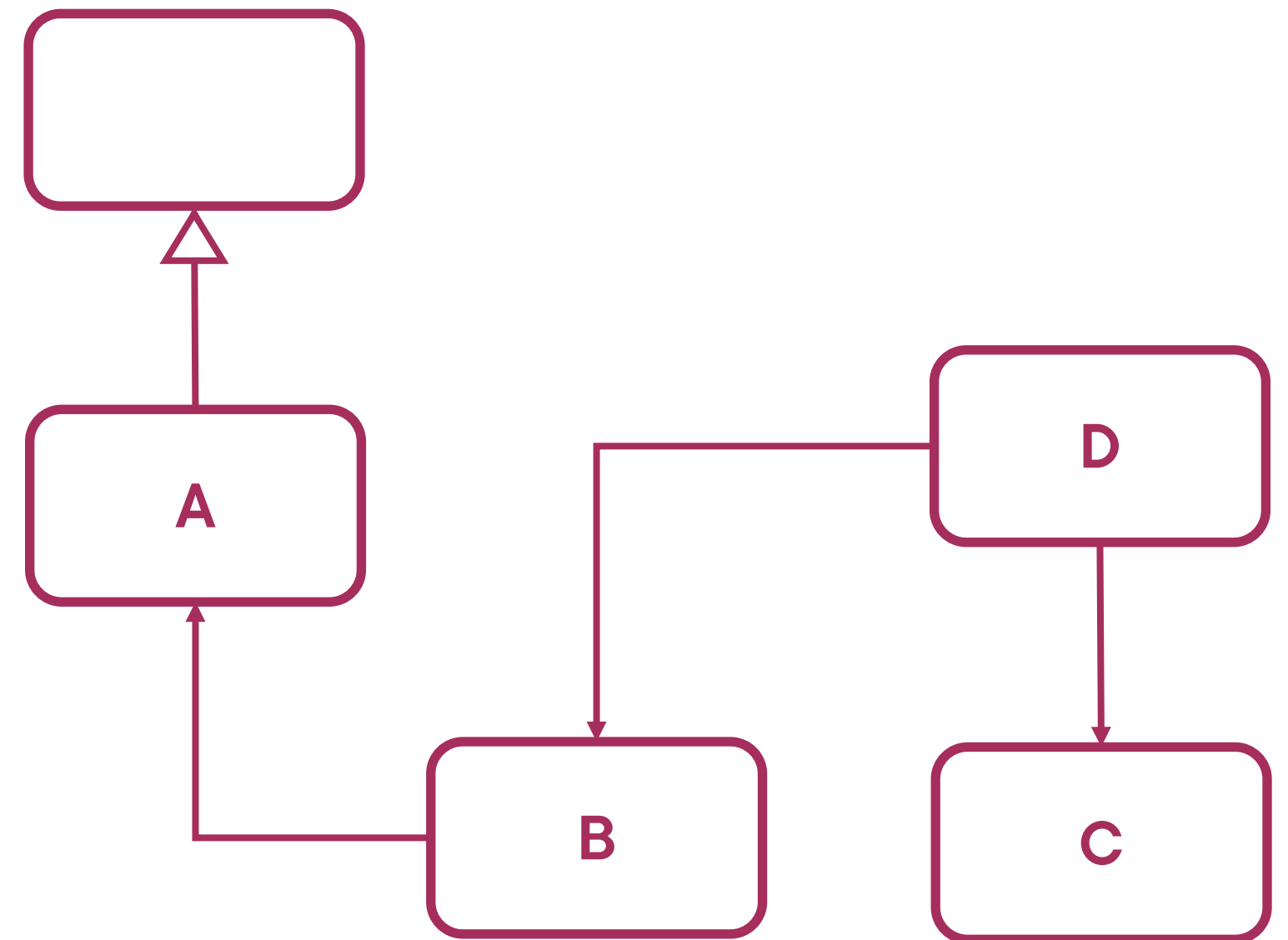
Request for D



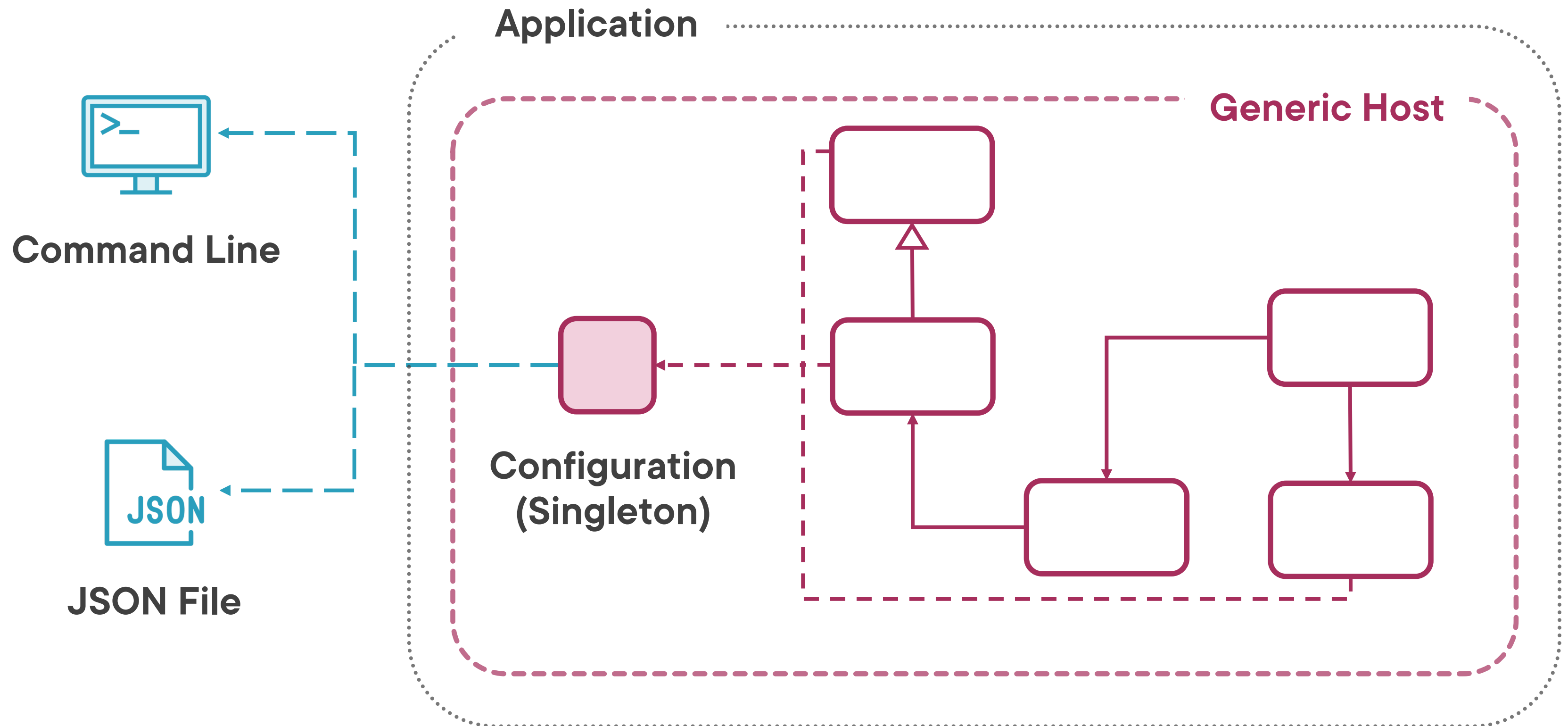
Instance of D

DI Container

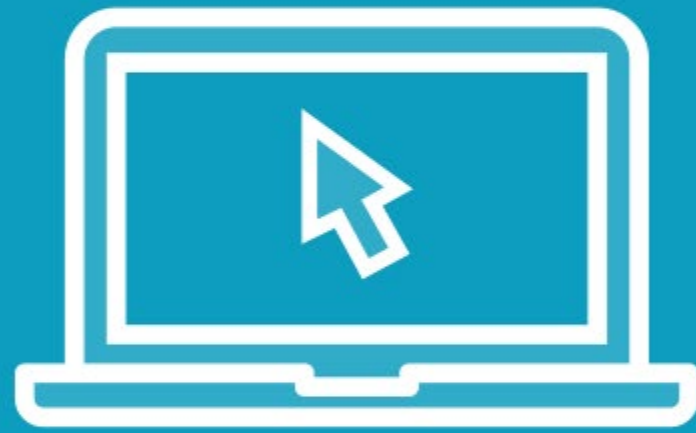
Type	Lifetime
A	Singleton
B	Transient
C	Transient
D	Scoped



Accessing Configuration



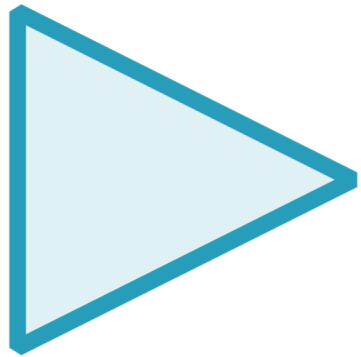
Demo



Add a new image service

- Register service with Host
- Use .NET FileWatcher class

Lifecycle



Starting

Call `StartAsync` on each `IHostedService`



Running

Wait for termination



Shutdown

Call `StopAsync` on each `IHostService`

Summary



.NET Generic Host

- **Cross-platform services**
- **Adds Logging, Configuration and Dependency Injection**

Use Worker Service project template

- **Visual Studio**
- **Or dotnet command line**

Host vs. Application Configuration

Up Next:

Adding Application Configuration to the Host
