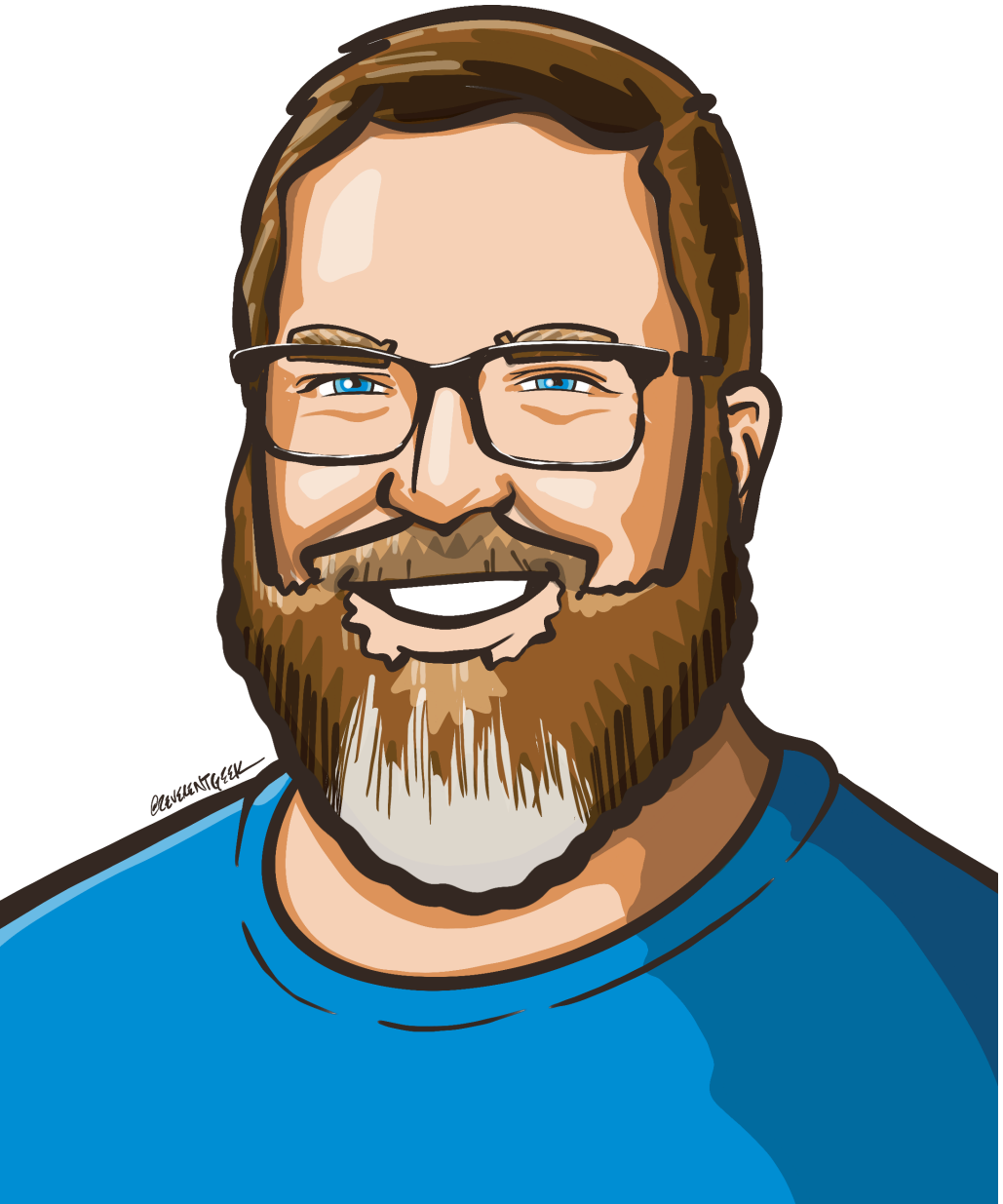
Modern .NET Configuration

Chris Ayers

https://github.com/codebytes/dotnet-configuration-in-depth



Chris Ayers

Principal Software Engineer
Azure EngOps AzRel
Microsoft

- BlueSky: @chris-ayers.com
- LinkedIn: - chris-l-ayers
- Blog: https://chris-ayers.com/
- GitHub: Codebytes
- Mastodon: @Chrisayers@hachyderm.io
- ~~Twitter: @Chris_L_Ayers~~

https://chris-ayers.com

Agenda

- What is configuration?
- How does .NET Framework handle configuration?
- How does .NET and ASP.NET handle configuration?
- Configuration Providers
- Configuration Binding
- The Options Pattern
- Questions?

What is Configuration?

Settings

- Retry Times
- Queue Length

Feature Flags

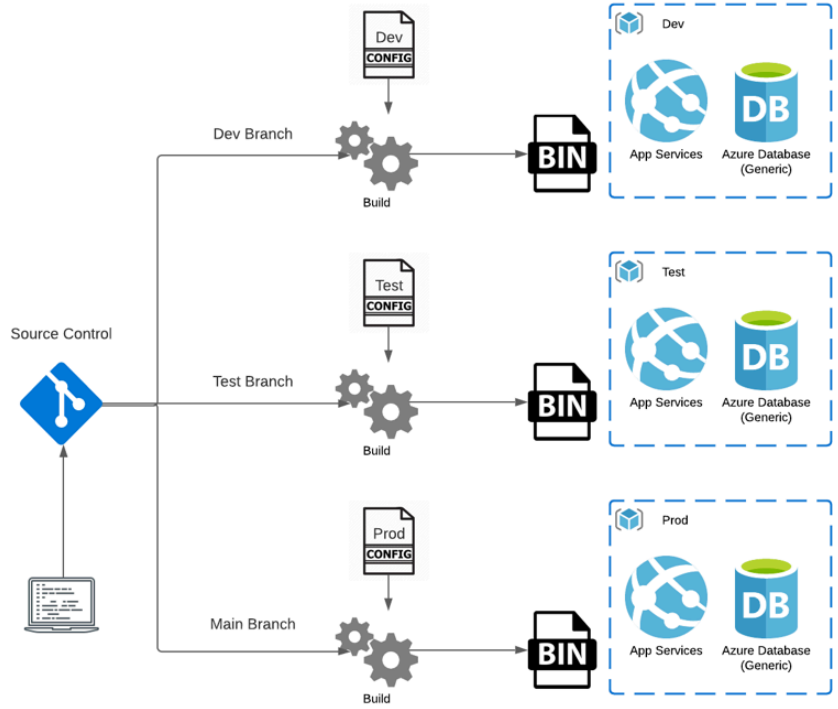
- Per User
- Percentage

Secrets

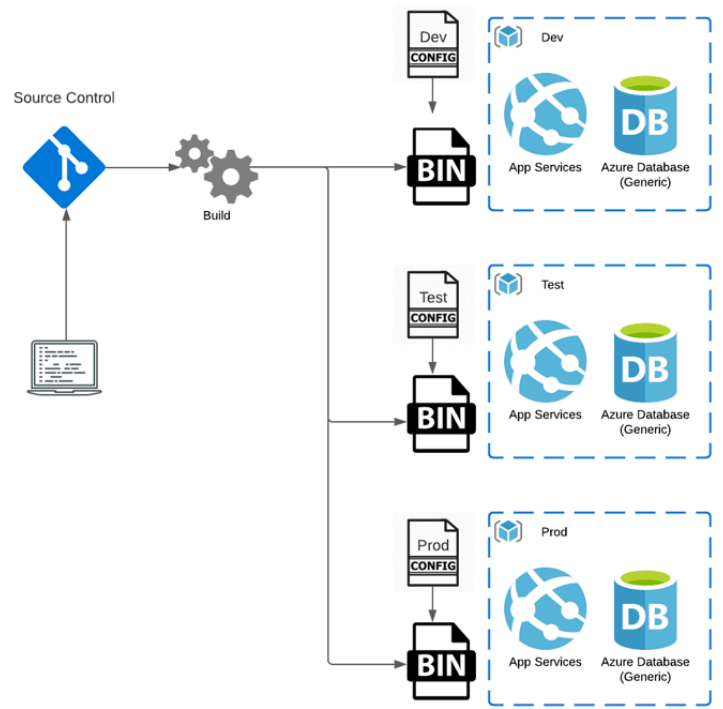
- Connection Strings
- App Registration

When is configuration applied?

</> Compile Time

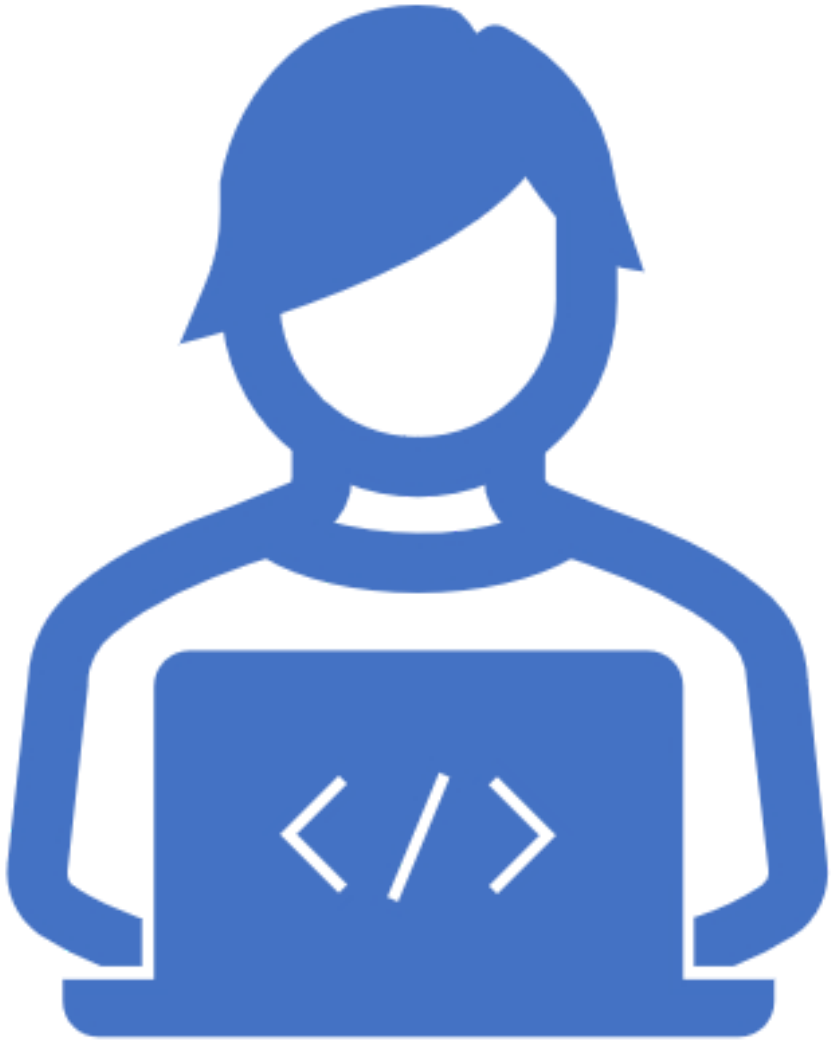


Run Time



https://chris-ayers.com

.NET Framework Configuration



Web.Config

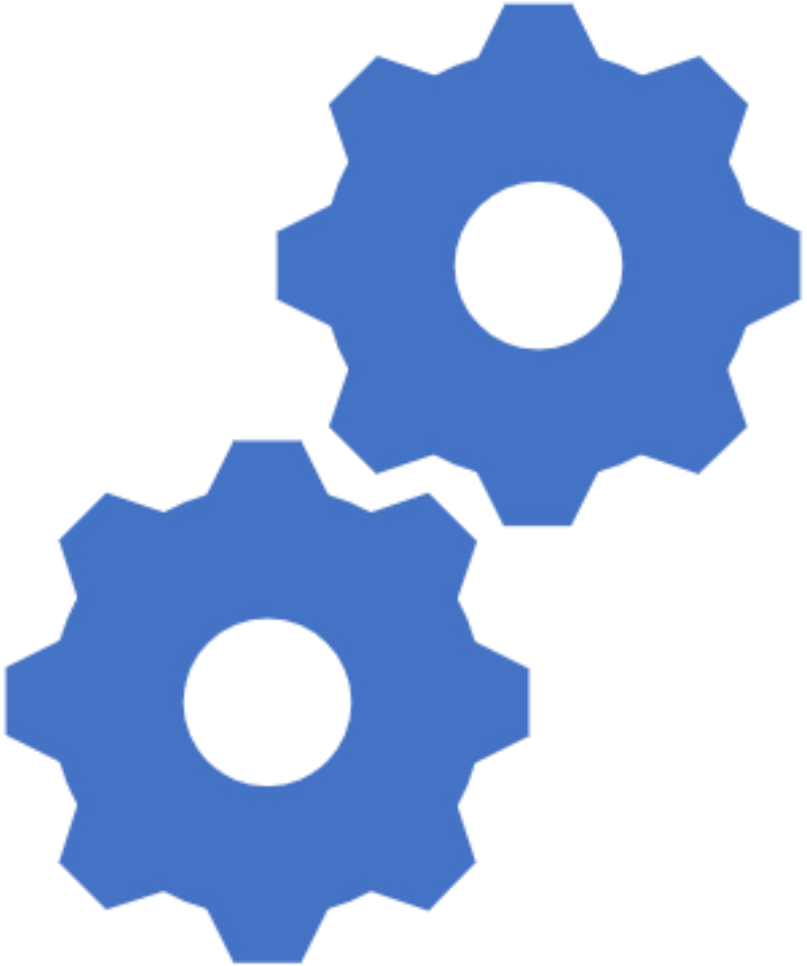
- Limited to Key-Value string pairs
- Accessed through a static ConfigurationManager Class
- Dependency Injection was not provided out of the box
- Transformation through difficult syntax
  - Slow Cheetah
- Hard to unit test
- Easy to leak secrets

XML, Static Classes, and Parsing

```
<appSettings>
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  <add key="Greeting" value="Hello, Everyone!" />
  <add key="CurrentMajorDotNetVersion" value="6" />
</appSettings>
<system.web>
  <compilation debug="true" targetFramework="4.7.2" />
  <httpRuntime targetFramework="4.7.2" />
</system.web>
```

```
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
  <connectionStrings>
    <add name="MyDB"
      connectionString="Data Source=ReleaseSQLServer..."
      xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
  </connectionStrings>
```

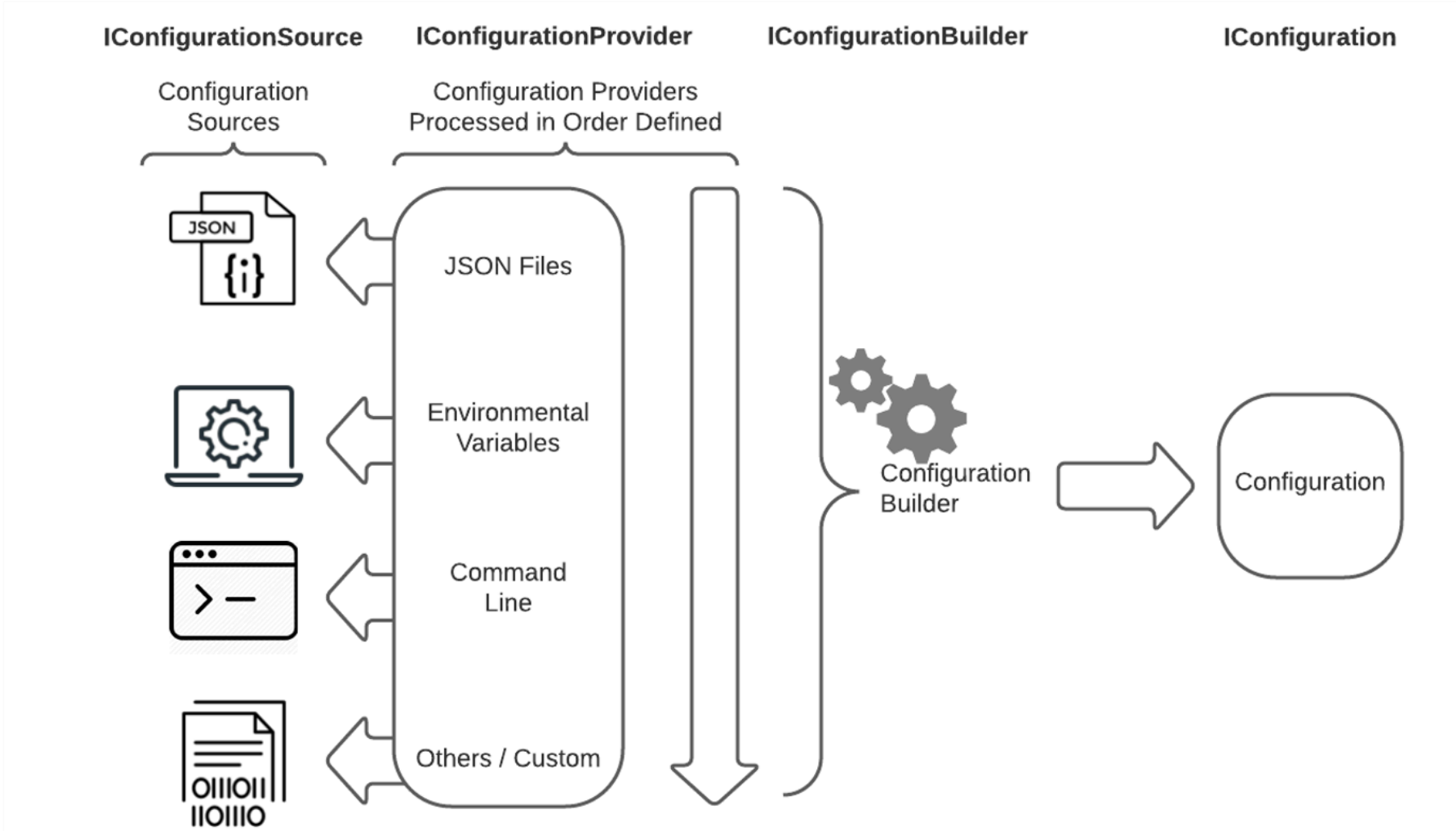
```
  private string greeting = "";
  private int majorDotNetVersion = 0;
  public HomeController()
  {
      greeting = ConfigurationManager.AppSettings["Greeting"];
      string ver = ConfigurationManager.AppSettings["CurrentMajorDotNetVersion"]
      majorDotNetVersion = Int32.Parse(ver);
  }
```



Modern .NET Configuration

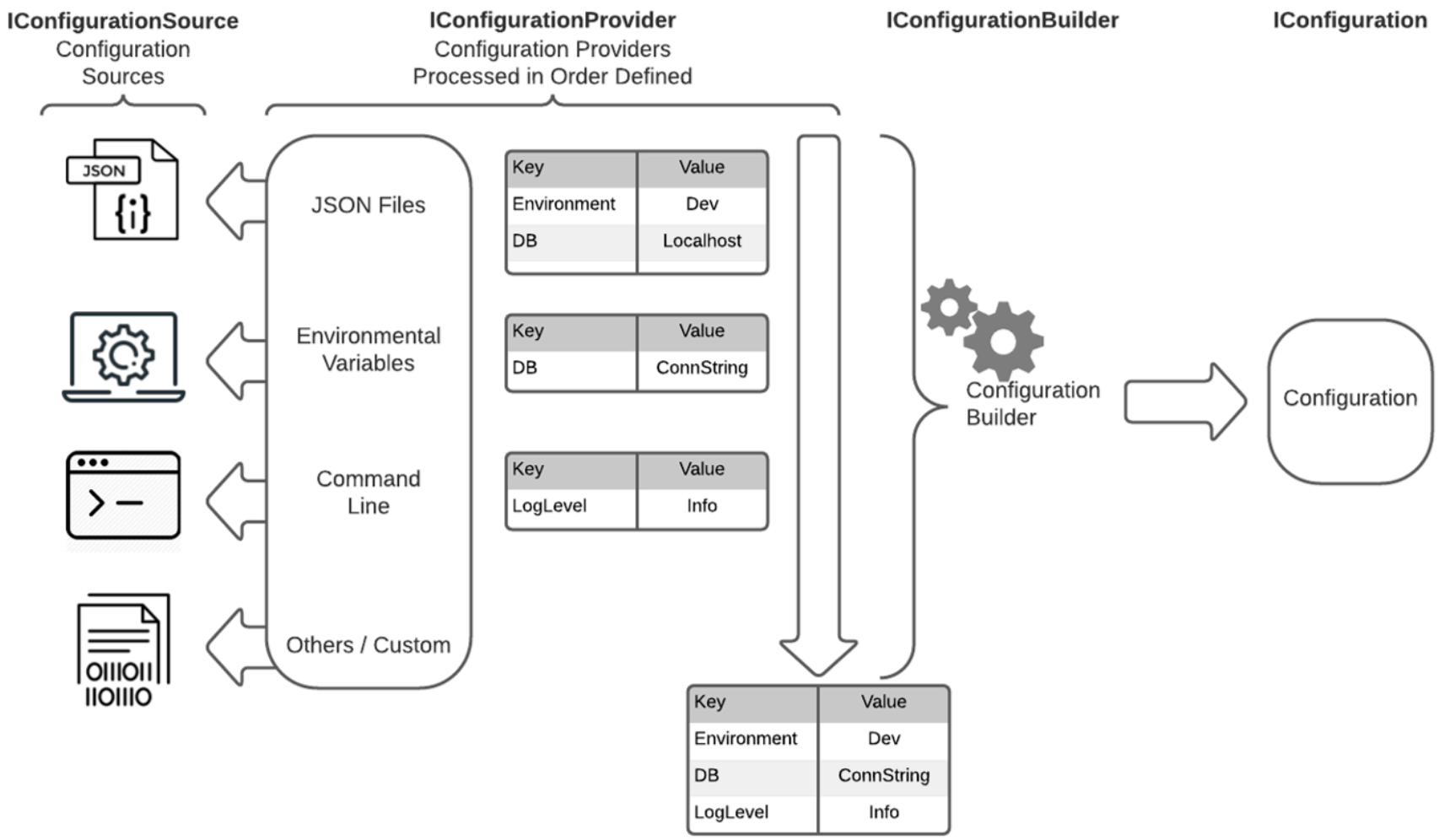
https://chris-ayers.com

Configuration Providers



https://chris-ayers.com

Order Matters



Binding

```
{
    "Settings": {
        "KeyOne": 1,
        "KeyTwo": true,
        "KeyThree": {
            "Message": "Oh, that's nice...",
            "SupportedVersions": {
                "v1": "1.0.0",
                "v3": "3.0.7"
            }
        }
    }
}
```

```
public sealed class Settings
{
    public required int KeyOne { get; set; }
    public required bool KeyTwo { get; set; }
    public required NestedSettings KeyThree { get; set; };
}
public sealed class NestedSettings
{
    public required string Message { get; set; };
}

IConfigurationRoot config = new ConfigurationBuilder()
    .AddJsonFile("appsettings.json")
    .Build();

Settings? settings =
  config.GetRequiredSection("Settings")
    .Get<Settings>();
```

Hierarchical Configuration Data

Keys are Flattened

```
{
  "Parent": {
    "FavoriteNumber": 7,
    "Child": {
      "Name": "Example",
      "GrandChild": {
        "Age": 3
      }
    }
  }
}
```

```
{
  "Parent:FavoriteNumber": 7,
  "Parent:Child:Name": "Example",
  "Parent:Child:GrandChild:Age": 3
}
```

Out of the Box

Console

- No Configuration

.NET Generic Host

- JSON
  - appsettings.json
  - appsettings.{Environment}.json
- User Secrets
- Environment Variables
- Command Line Variables

ASP.NET

- JSON
  - appsettings.json
  - appsettings.{Environment}.json
- User Secrets
- Environment Variables
- Command Line Variables

Configuration Providers

File-based

- JSON
- XML
- INI
- Key-per-file

Others

- Environment variables
- Command-line
- In-Memory
- User Secrets
- Azure Key Vault
- Azure App Configuration

Json Provider

```
<PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="8.0.0" />
```

```
IHostEnvironment env = builder.Environment;

builder.Configuration
  .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
  .AddJsonFile($"appsettings.{env.EnvironmentName}.json", true, true);
```

```
{
    "SecretKey": "Secret key value",
    "TransientFaultHandlingOptions": {
        "Enabled": true,
        "AutoRetryDelay": "00:00:07"
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        }
    }
}
```

XML Provider

```
<PackageReference Include="Microsoft.Extensions.Configuration.Xml" Version="8.0.0" />
```

```
IHostEnvironment env = builder.Environment;

builder.Configuration
  .AddXmlFile("appsettings.xml", optional: true, reloadOnChange: true)
  .AddXmlFile("repeating-example.xml", optional: true, reloadOnChange: true);
```

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <SecretKey>Secret key value</SecretKey>
  <TransientFaultHandlingOptions>
    <Enabled>true</Enabled>
    <AutoRetryDelay>00:00:07</AutoRetryDelay>
  </TransientFaultHandlingOptions>
  <Logging>
    <LogLevel>
      <Default>Information</Default>
      <Microsoft>Warning</Microsoft>
    </LogLevel>
  </Logging>
</configuration>
```

INI Provider

```
<PackageReference Include="Microsoft.Extensions.Configuration.Ini" Version="8.0.0" />
```

```
IHostEnvironment env = builder.Environment;

builder.Configuration
  .AddIniFile("appsettings.ini", optional: true, reloadOnChange: true)
  .AddIniFile($"appsettings.{env.EnvironmentName}.ini", true, true);
```

```
SecretKey="Secret key value"

[TransientFaultHandlingOptions]
Enabled=True
AutoRetryDelay="00:00:07"

[Logging:LogLevel]
Default=Information
Microsoft=Warning
```

Environment Variables

- Typically used to override settings found in appsettings.json or user secrets
- the : delimiter doesn't work for Hierarchical data on all platforms
- the `__` delimiter is used instead of `:`

```
public class TransientFaultHandlingOptions
{
    public bool Enabled { get; set; }
    public TimeSpan AutoRetryDelay { get; set; }
}
```

```
set SecretKey="Secret key from environment"
set TransientFaultHandlingOptions__Enabled="true"
set TransientFaultHandlingOptions__AutoRetryDelay="00:00:13"
```

Environment Variables

- There are built-in prefixes, like
  - ASPNETCORE_ for ASP.NET Core
  - DOTNET_ for .NET Core
- You can provide your own prefix

```
builder.Configuration.AddEnvironmentVariables(
  prefix: "MyCustomPrefix_");
```

```
set MyCustomPrefix_MyKey="My key with MyCustomPrefix_"
set MyCustomPrefix_Position__Title=Editor_with_custom
set MyCustomPrefix_Position__Name=Environment
dotnet run
```

Environment Variables for Environment

- `ASPNETCORE_ENVIRONMENT` (ASP.NET Core) — choose `Development` / `Staging` / `Production` per slot to drive `IWebHostEnvironment`.
- `DOTNET_ENVIRONMENT` (generic hosts, workers) — keep identical to `ASPNETCORE_ENVIRONMENT` so overrides stay predictable.
- `ASPNETCORE_ENVIRONMENT` via App Settings (Azure App Service) — one App Setting change flips every instance.
- `DOTNET_ENVIRONMENT` in containers (Docker/K8s) — bake into manifests so host + container diagnostics align.

Ports & URL Bindings

- `ASPNETCORE_URLS` (any host) — specify full scheme + IP; separate multiples with `;`.
- `HTTP_PORTS` / `HTTPS_PORTS` (.NET 8 containers) — shorthand feeding `ASPNETCORE_URLS`; defaults 8080/8443.
- `WEBSITES_PORT` (Azure App Service custom containers) — set via App Settings so routing targets the right port.
- `Kestrel__Endpoints__Http__Url` (fine-grained Kestrel) — use `__` hierarchy for per-endpoint bindings and cert wiring.

Diagnostics, GC & Container Flags

- `DOTNET_EnableDiagnostics` — set to `0` to block debugger/profiler attaches; combine with diagnostic ports for exceptions.
- `DOTNET_DiagnosticPorts` — define connect/listen/suspend behavior when you still need targeted traces.
- `DOTNET_RUNNING_IN_CONTAINER` — makes GC + ThreadPool honor cgroup limits inside containers.

DEMOS

The Options Pattern

Interface Segregation Principle (ISP): Scenarios (classes) that depend on configuration settings depend only on the configuration settings that they use.

Separation of Concerns : Settings for different parts of the app aren't dependent or coupled to one another.

An Options Class

- Must be non-abstract with a public parameterless constructor
- Contain public read-write properties to bind (fields are not bound)

```
public class FileOptions
{
    public string FileExtension { get; set; } ="";
    public string OutputDir { get; set; } ="";
    public string TemplateFile { get; set; } ="";
}
```

Types of IOptions

	Singleton	Reloading Support	Named Option Support
IOptions<T>	Yes	No	No
IOptionsSnapshot<T>	No	Yes	Yes
IOptionsMonitor<T>	Yes	Yes	Yes

Performance: IOptionsSnapshot vs IOptionsMonitor

IOptionsSnapshot<T>

- Scoped lifetime (per request)
- Recomputed each request
- Supports named options
- Best for web apps

```
public class MyController : Controller
{
    public MyController(IOptionsSnapshot<MyOptions> options)
    {
        // Fresh config per request
        _options = options.Value;
    }
}
```

IOptionsMonitor<T>

- Singleton lifetime
- Real-time change notifications
- Supports named options
- Best for background services

```
public class MyService : BackgroundService
{
    public MyService(IOptionsMonitor<MyOptions> monitor)
    {
        // React to config changes
        monitor.OnChange(OnConfigChanged);
    }
}
```

Testing Configuration

Configuration Testing Patterns

Unit Testing

```
[Test]
public void Service_Uses_Configuration_Correctly()
{
    // Arrange
    var config = new ConfigurationBuilder()
        .AddInMemoryCollection(new[]
        {
            new KeyValuePair<string, string>("ApiUrl", "https://test.api"),
            new KeyValuePair<string, string>("Timeout", "30")
        })
        .Build();

    var options = Options.Create(config.Get<ApiOptions>());
    var service = new ApiService(options);

    // Act & Assert
    Assert.That(service.BaseUrl, Is.EqualTo("https://test.api"));
}
```

Integration Testing

```
public class TestWebApplicationFactory<TProgram>
    : WebApplicationFactory<TProgram> where TProgram : class
{
    protected override void ConfigureWebHost(IWebHostBuilder builder)
    {
        builder.ConfigureAppConfiguration(config =>
        {
            config.AddInMemoryCollection(new[]
            {
                new KeyValuePair<string, string>("Database:ConnectionString",
                    "Server=localhost;Database=TestDb;"),
                new KeyValuePair<string, string>("ExternalApi:BaseUrl",
                    "https://mock-api.test")
            });
        });
    }
}
```

Configuration Validation

Data Annotations

```
public class DatabaseOptions
{
    [Required]
    [Url]
    public string ConnectionString { get; set; } = "";

    [Range(1, 300)]
    public int CommandTimeoutSeconds { get; set; } = 30;

    [Required]
    [RegularExpression(@"^[a-zA-Z0-9_]+$")]
    public string DatabaseName { get; set; } = "";
}
services.AddOptions<DatabaseOptions>()
    .Bind(configuration.GetSection("Database"))
    .ValidateDataAnnotations()
    .ValidateOnStart();
```

Custom Validation

```
public class DatabaseOptionsValidator : IValidateOptions<DatabaseOptions>
{
    public ValidateOptionsResult Validate(string name, DatabaseOptions options)
    {
        var failures = new List<string>();

        if (string.IsNullOrEmpty(options.ConnectionString))
            failures.Add("ConnectionString is required");
        if (options.CommandTimeoutSeconds <= 0)
            failures.Add("CommandTimeoutSeconds must be positive");
        if (!IsValidDatabaseName(options.DatabaseName))
            failures.Add("Invalid database name format");

        return failures.Count > 0
            ? ValidateOptionsResult.Fail(failures)
            : ValidateOptionsResult.Success;
    }
}

// Register validator
services.AddSingleton<IValidateOptions<DatabaseOptions>, DatabaseOptionsValidator>();
```

Validation at Startup

Fail Fast Pattern

```
// Program.cs
var builder = WebApplication.CreateBuilder(args);

// Configure options with validation
builder.Services.AddOptions<ApiOptions>()
    .Bind(builder.Configuration.GetSection("Api"))
    .ValidateDataAnnotations()
    .ValidateOnStart(); // Validates during app startup

builder.Services.AddOptions<DatabaseOptions>()
    .Bind(builder.Configuration.GetSection("Database"))
    .Validate(options => !string.IsNullOrEmpty(options.ConnectionString),
              "Connection string cannot be empty")
    .ValidateOnStart();

var app = builder.Build();
// App fails to start if validation fails
```

Benefits

- **Early Detection**: Catch configuration errors at startup
- **Clear Error Messages**: Know exactly what's wrong
- **Prevents Runtime Failures**: No surprises in production
- **Better DevEx**: Immediate feedback during development

```
services.AddOptions<MyOptions>()
    .Bind(configuration.GetSection("MySection"))
    .Validate(options =>
        { return options.ApiKey?.Length >= 10; },
        "ApiKey must be at least 10 characters")
    .ValidateOnStart();
```

DEMOS

Secrets Management Best Practices

⚠ Don't

- Store secrets in appsettings.json
- Commit secrets to source control
- Use production secrets in development
- Log configuration values containing secrets

✔ Do

- Use User Secrets for development
- Use Azure Key Vault for production
- Use environment variables for containers
- Implement proper secret rotation
- Validate secrets at startup

Secrets by Environment

Development

- User Secrets
  - Per-project secrets
  - Stored outside source control
  - Easy to manage locally

```
dotnet user-secrets set "ApiKey" "dev-key-123"
```

Staging/Production

- Azure Key Vault
  - Centralized secret management
  - Access policies and RBAC
  - Audit logging
  - Automatic rotation

```
builder.Configuration.AddAzureKeyVault(
  keyVaultUrl, credential);
```

Containers

- Environment Variables
  - Kubernetes secrets
  - Docker secrets
  - Service connection strings

```
docker run -e "ConnectionString=..." myapp
```

Environment-Specific Configuration Strategies

Layered Configuration

```
builder.Configuration
    .AddJsonFile("appsettings.json")
    .AddJsonFile($"appsettings.{env.EnvironmentName}.json", true)
    .AddEnvironmentVariables()
    .AddCommandLine(args);
```

Order matters! Later sources override earlier ones.

Environment Patterns

- **Development**: User Secrets + local files
- **Staging**: Environment variables + Key Vault
- **Production**: Key Vault + minimal env vars
- **Testing**: In-memory configuration

```
if (env.IsDevelopment())
{
    builder.Configuration.AddUserSecrets<Program>();
}
```

Configuration Security Considerations

Prevent Secret Leakage

- Never log IConfiguration directly
- Redact sensitive values in logs
- Use IOptionsSnapshot/IOptionsMonitor
- Implement custom configuration providers for sensitive data

Secure Logging

```
// ❌ DON'T - Exposes all configuration
logger.LogInformation("Config: {Config}",
    JsonSerializer.Serialize(configuration));

// ✅ DO - Log specific, non-sensitive values
logger.LogInformation("Database timeout: {Timeout}s",
    dbOptions.CommandTimeout);
```

.NET Aspire

.NET Aspire Configuration

Cloud-Native Configuration Made Simple

- **Orchestration**: Centralized service management through AppHost
- **Service Defaults**: Opinionated baseline for observability, health checks, and service discovery
- **Configuration Layering**: Hierarchical configuration across distributed services
- **Modern Patterns**: Built-in support for microservices and cloud-native apps

Aspire Configuration Architecture

AppHost Project

```
var builder = DistributedApplication.CreateBuilder(args);

var apiService = builder.AddProject<Projects.ApiService>("apiservice");

var workerService = builder.AddProject<Projects.WorkerService>("workerservice")
    .WithEnvironment("Api:BaseUrl", apiService.GetEndpoint("https"));

builder.Build().Run();
```

Service Defaults

```
public static class Extensions
{
    public static IHostApplicationBuilder AddServiceDefaults(this IHostApplicationBuilder builder)
    {
        builder.ConfigureOpenTelemetry();
        builder.AddDefaultHealthChecks();
        builder.Services.AddServiceDiscovery();

        builder.Services.ConfigureHttpClientDefaults(http =>
        {
            http.AddStandardResilienceHandler();
            http.UseServiceDiscovery();
        });

        return builder;
    }
}
```

Aspire Configuration Layering

Configuration Priority (Last Wins)

1. **SharedConfig** `appsettings.json` - Cross-service shared settings
2. **Service-specific** `appsettings.json` - Per-service configuration
3. **Environment-specific** `appsettings.{Environment}.json`
4. **User Secrets** (Development only)
5. **AppHost Environment Variables** - `WithEnvironment()` calls
6. **Command Line Parameters** - Runtime overrides

```
// AppHost parameter injection
var apiService = builder.AddProject<Projects.ApiService>("apiservice")
    .WithEnvironment("Api:InjectedMessage", builder.AddParameter("ApiBaseMessage"));
```

Questions?

Resources

GitHub Repo

- https://github.com/codebytes/dotnet-configuration-in-depth

Docs

- .NET Configuration Docs
- .NET Aspire Docs
- .NET Environment Variables

Contact

Chris Ayers

BlueSky: @chris-ayers.com
LinkedIn: - chris-l-ayers
Blog: https://chris-ayers.com/
GitHub: Codebytes
Mastodon: @Chrisayers@hachyderm.io
~~Twitter: @Chris_L_Ayers~~