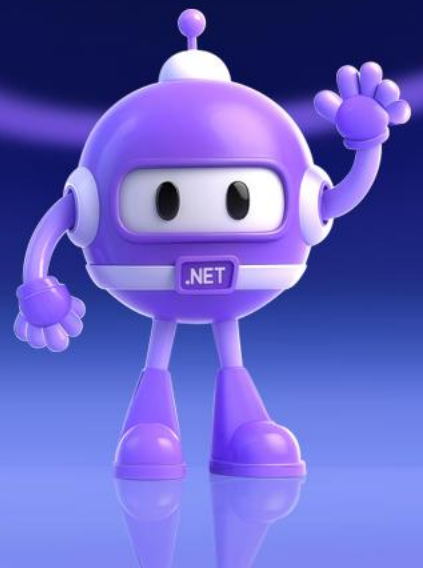
.NET Conf China 2025

改变世界 改变自己

2025 年 11 月 30 日 | 中国 上海

.NET Conf China 2025
改变世界 改变自己

依赖即代码：用 Testcontainers 重构.NET AI Agents 的测试体系



衣明志

微软最有价值专家(AI 平台与开发技术)

基普智能创始人

烟台易云创始人

现代软件 / AI Agent 开发交付周期

现代软件 / AI Agent 开发交付周期

现代软件 / AI Agent 开发交付周期

应用程序完整性的双重保障

单元测试

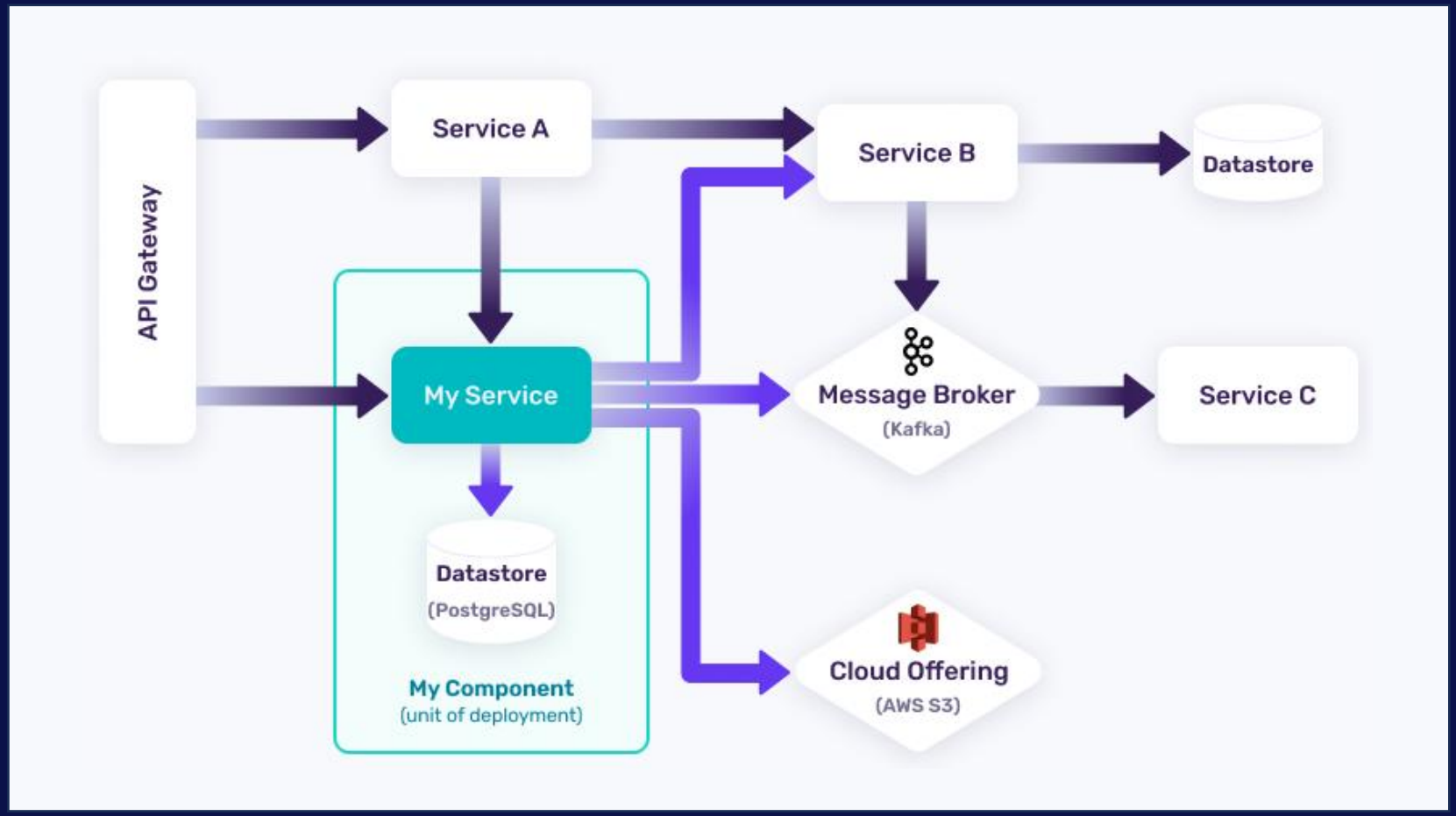
验证孤立的代码组件

集成测试

验证服务交互

你通常是怎样做单元测试的?

集成测试的挑战：一个简单的微服务



集成测试的挑战


预配置数据
确保基础设施启动并运行
基础设施准备
解决环境问题
维护集成测试环境
环境维护
测试执行
测试执行干扰
测试污染
并行测试
并行测试冲突
数据一致性问题

如何有效地测试与外部服务的交互？

模拟框架

复杂且难以管理

VS

本地设置

脆弱且难以复制

我们需要一种可以在真实的、
类似生产环境下运行测试的方法，无需手动设置的麻烦

Testcontainers是什么

Testcontainers 是一个库，它提供简单轻量级的 API，用于引导本地开发环境并测试依赖项，这些依赖项使用封装在 Docker 容器中的真实服务。

使用 Testcontainers，您可以编写依赖于与生产环境相同服务的测试，而无需使用模拟对象或内存服务。

Testcontainers 的特点

- **使用真实依赖项的单元测试**

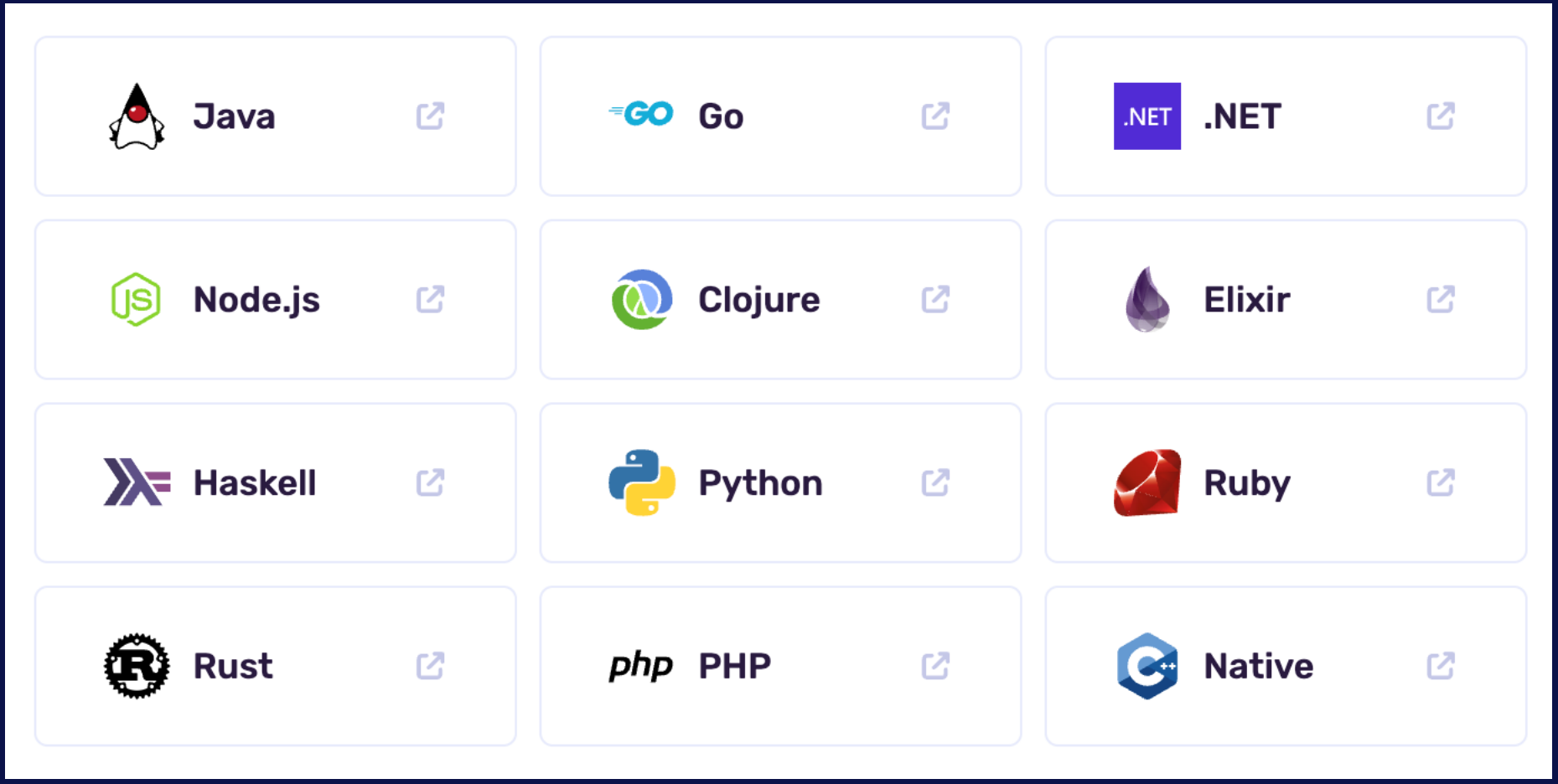
Testcontainers 是一个开源库，用于提供一次性、轻量级的数据库、消息代理、Web 浏览器实例，或者几乎任何可以在 Docker 容器中运行的东西。

- **以代码形式测试依赖项**

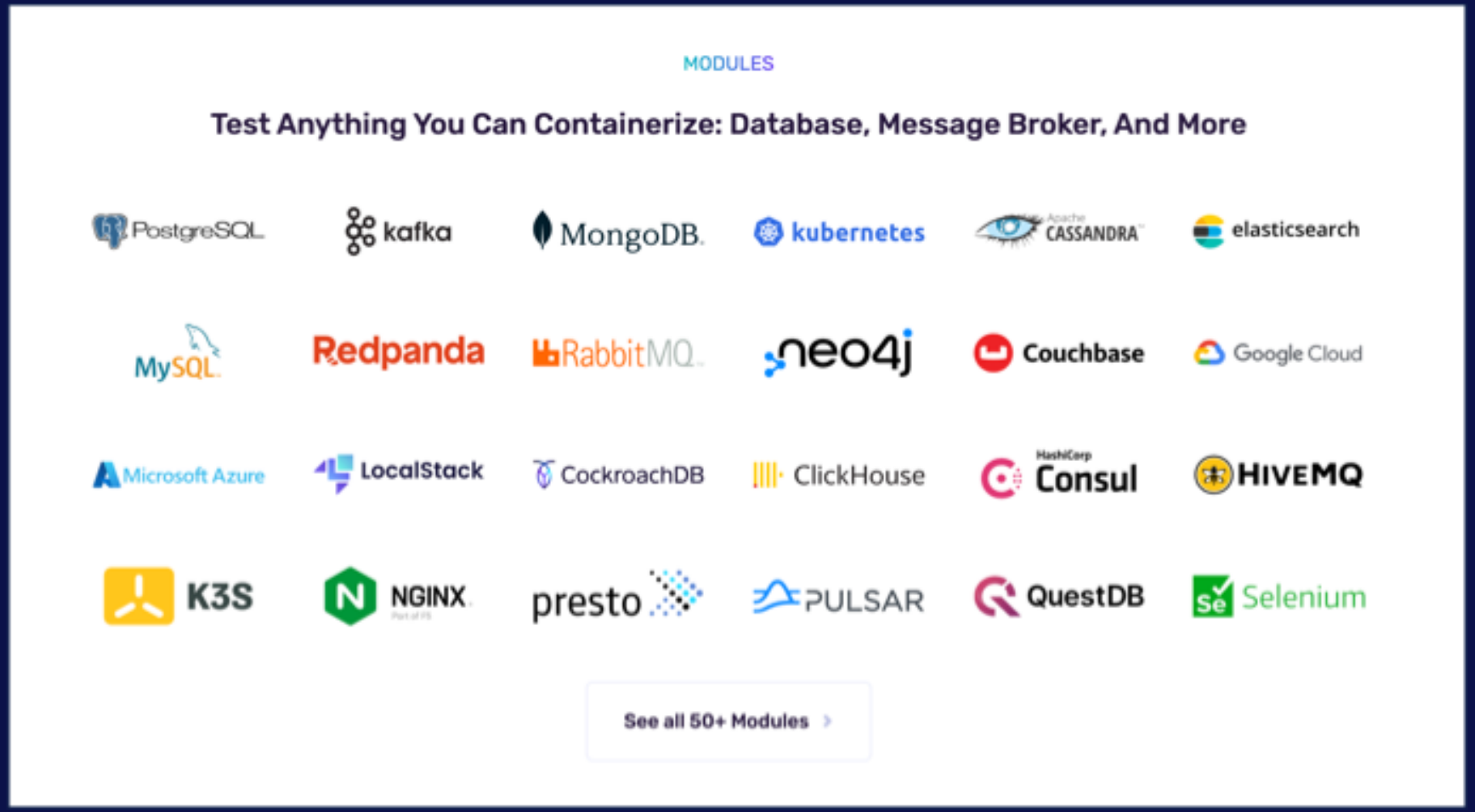
无需再使用模拟对象或复杂的环境配置。只需将测试依赖项定义为代码，然后运行测试，容器就会自动创建并在测试完成后自动删除。

Docker 支持多种语言和测试框架，你只需要它就够了。

Testcontainers支持10几种主流语言



提供了50多种测试的容器类型



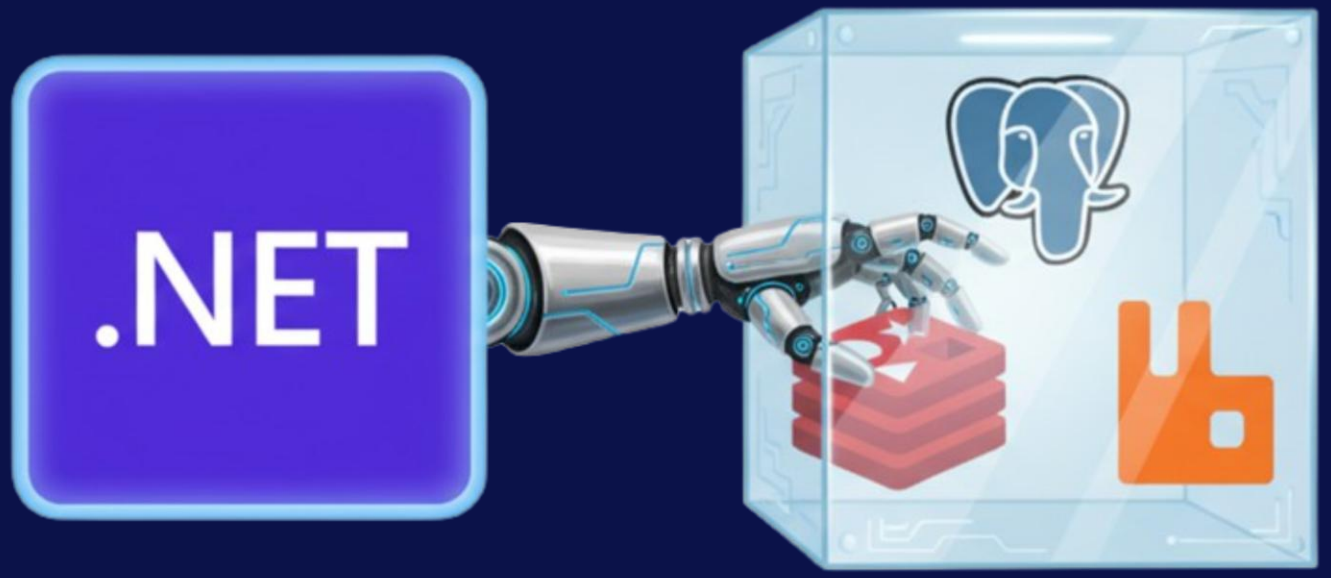
使用 Testcontainers 的优势

1. **真实的集成测试**
   与模拟不同，模拟测试如果模拟行为与真实服务不完全匹配可能导致误报，Testcontainers 允许你与真实服务进行测试。这让你有信心你的代码能在实时环境中运行。

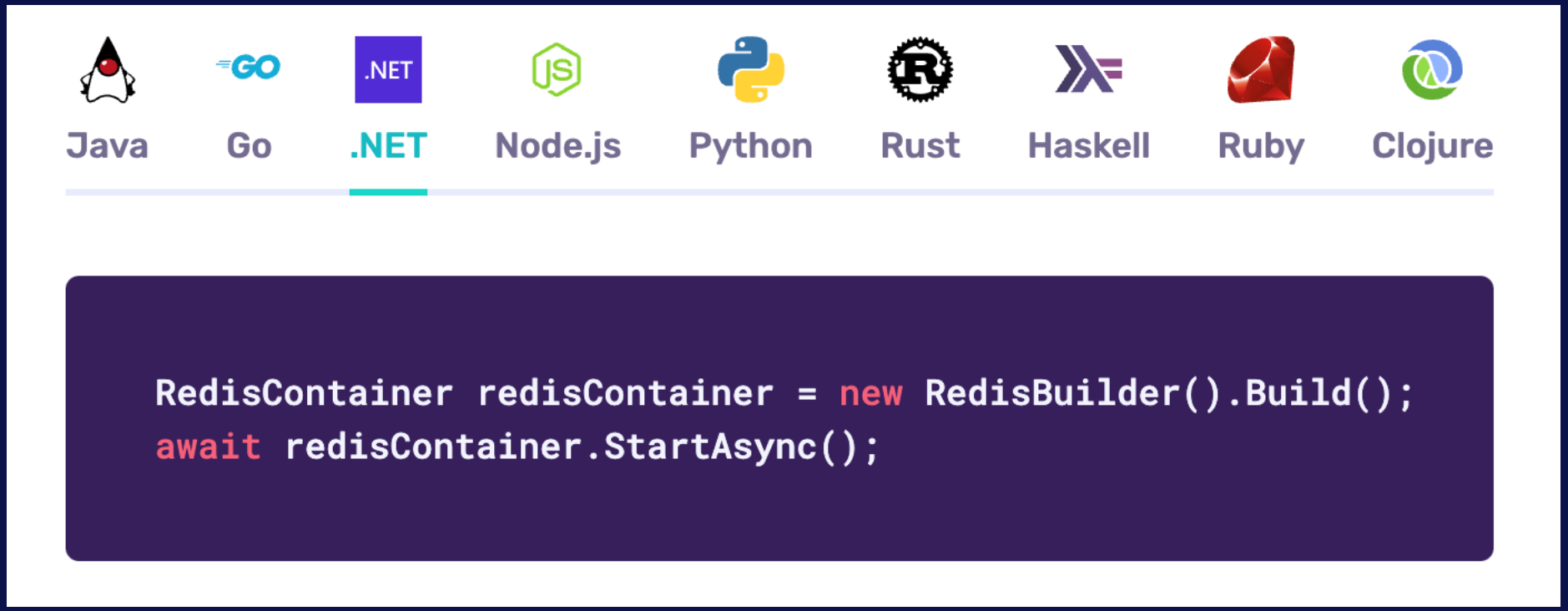
2. **隔离与一致性**
   每次测试都会有一个干净、隔离的容器。这意味着测试之间不会相互干扰，消除了共享状态或之前运行遗留数据带来的常见问题。你可以确定测试失败是因为代码问题，而不是环境问题。

3. **不再手动设置**
   别再去想安装和配置本地数据库的耗时过程了。使用 Testcontainer 时，你只需要一个运行的 Docker 守护进程。 **你的测试会自动拉取所需的图像并启动容器** 。这会让你的项目设置对新团队成员来说简单得多。

4. **语言中立**
   Testcontainers 也提供了涵盖多种语言的库，包括 Python、Go 和 .NET


.NET

Testcontainers 的 .NET 支持

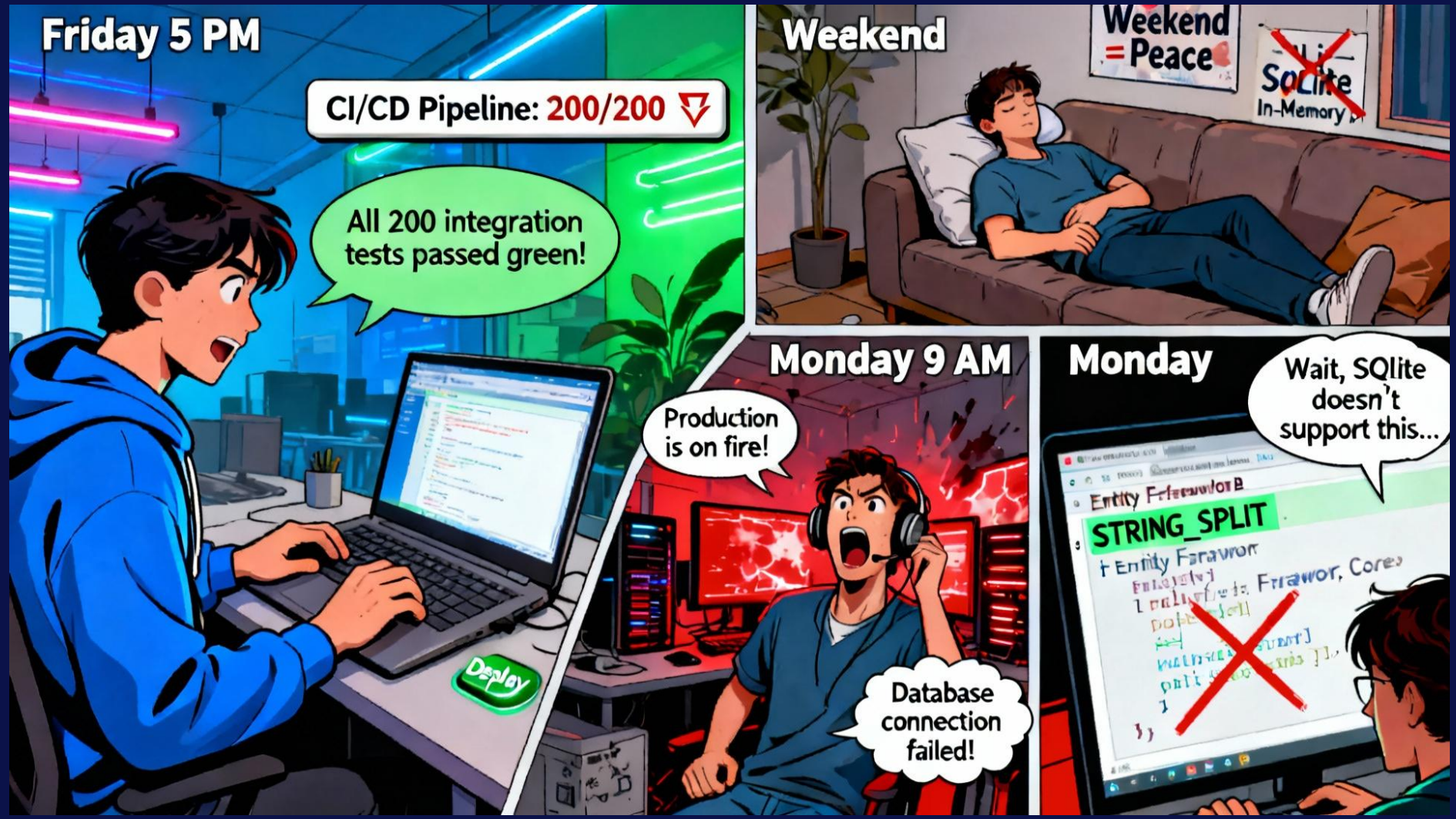


Testcontainers 的基本用法

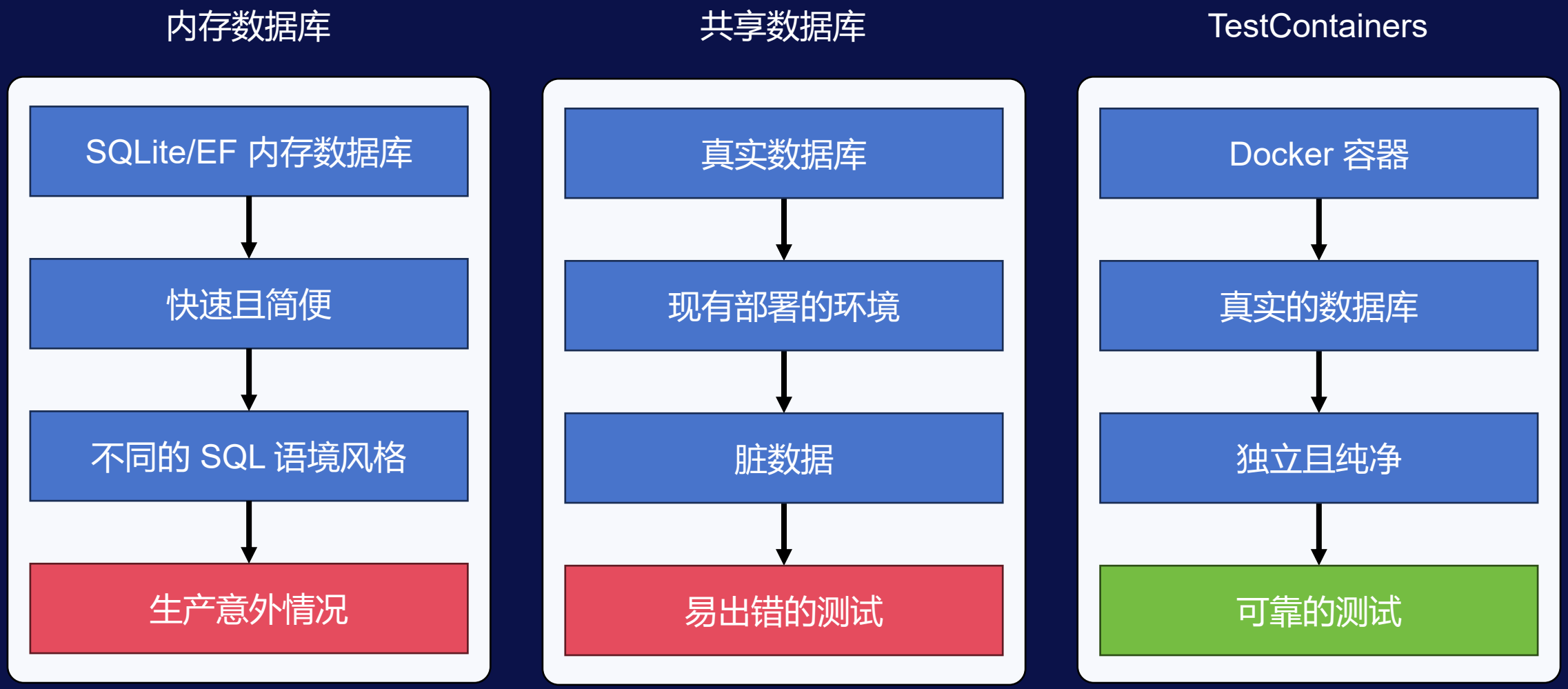
dotnet add package Testcontainers

```
// 创建容器实例。
var container = new ContainerBuilder()
  // 设置容器镜像为 "testcontainers/helloworld:1.3.0"。
  .WithImage("testcontainers/helloworld:1.3.0")
  // 将容器的 8080 端口映射到主机的随机端口。
  .WithPortBinding(8080, true)
  // 等待容器的 HTTP 端点可用。
  .WithWaitStrategy(Wait.ForUnixContainer().UntilHttpRequestIsSucceeded(r => r.ForPort(8080)))
  // 构建容器配置。
  .Build();

// 启动容器。
await container.StartAsync()
  .ConfigureAwait(false);

// 创建 HttpClient 实例以发送 HTTP 请求。
using var httpClient = new HttpClient();

// 构建请求 URI：指定协议、主机名、分配的随机主机端口和 "uuid" 端点。
var requestUri = new UriBuilder(Uri.UriSchemeHttp, container.Hostname, container.GetMappedPublicPort(8080), "uuid").Uri;

// 发送 HTTP GET 请求并以字符串形式获取响应。
var guid = await httpClient.GetStringAsync(requestUri)
  .ConfigureAwait(false);

// 确认检索到的 UUID 是有效的 GUID。
Debug.Assert(Guid.TryParse(guid, out _));
```

一个 EF 测试的小故事


Friday 5 PM
CI/CD Pipeline: 200/200
All 200 integration tests passed green!
Deploy
Weekend
Weekend = Peace
SQLite In-Memory
Monday 9 AM
Production is on fire!
Database connection failed!
Monday
Wait, SQlite doesn't support this...
STRING_SPLIT

以 EF 为例不同测试方案的对比



```
dotnet add package Testcontainers --version 4.9.0
dotnet add package Testcontainers.PostgreSql --version 4.9.0
# 或者其他特定数据库的包 比如 Testcontainers.SqlServer, Testcontainers.MySql
```

创建一个管理测试共享状态的 Fixture

```csharp
// PostgresFixture.cs
public class PostgresFixture : IAsyncLifetime
{
    private readonly IContainer _postgresContainer;
    public string ConnectionString { get; private set; }

    public PostgresFixture()
    {
        _postgresContainer = new PostgreSqlBuilder()
            .WithImage("postgres:17-alpine") // 始终使用具体的版本
            .WithDatabase("test_db")
            .WithUsername("test_user")
            .WithPassword("test_password")
            .Build();
    }

    public async Task InitializeAsync()
    {
        await _postgresContainer.StartAsync();
        ConnectionString = _postgresContainer.GetConnectionString();
    }

    public async Task DisposeAsync()
    {
        await _postgresContainer.StopAsync();
    }
}
```

```csharp
// PostgresFixture.cs
public class PostgresFixture : IAsyncLifetime
{
    private readonly IContainer _postgresContainer;
    public string ConnectionString { get; private set; }

    public PostgresFixture()
    {
        _postgresContainer = new PostgreSqlBuilder()
            .WithImage("postgres:17-alpine") // 始终使用具体的版本
            .WithDatabase("test_db")
            .WithUsername("test_user")
            .WithPassword("test_password")
            .Build();
    }

    public async Task InitializeAsync()
    {
        await _postgresContainer.StartAsync();
        ConnectionString = _postgresContainer.GetConnectionString();
    }

    public async Task DisposeAsync()
    {
        await _postgresContainer.StopAsync();
    }
}
```

定义我们的 Postgres 容器

```csharp
// PostgresFixture.cs
public class PostgresFixture : IAsyncLifetime
{
    private readonly IContainer _postgresContainer;
    public string ConnectionString { get; private set; }

    public PostgresFixture()
    {
        _postgresContainer = new PostgreSqlBuilder()
            .WithImage("postgres:17-alpine") // 始终使用具体的版本
            .WithDatabase("test_db")
            .WithUsername("test_user")
            .WithPassword("test_password")
            .Build();
    }

    public async Task InitializeAsync()
    {
        await _postgresContainer.StartAsync();
        ConnectionString = _postgresContainer.GetConnectionString();
    }

    public async Task DisposeAsync()
    {
        await _postgresContainer.StopAsync();
    }
}
```

在测试开始前启动容器

```csharp
// PostgresFixture.cs
public class PostgresFixture : IAsyncLifetime
{
    private readonly IContainer _postgresContainer;
    public string ConnectionString { get; private set; }

    public PostgresFixture()
    {
        _postgresContainer = new PostgreSqlBuilder()
            .WithImage("postgres:17-alpine") // 始终使用具体的版本
            .WithDatabase("test_db")
            .WithUsername("test_user")
            .WithPassword("test_password")
            .Build();
    }

    public async Task InitializeAsync()
    {
        await _postgresContainer.StartAsync();
        ConnectionString = _postgresContainer.GetConnectionString();
    }

    public async Task DisposeAsync()
    {
        await _postgresContainer.StopAsync();
    }
}
```

在所有测试完成后销毁容器

创建一个基于数据库操作的测试类

```
[Collection("Postgres Collection")]
public class OrderRepositoryTests
{
    private readonly PostgresFixture _fixture;

    public OrderRepositoryTests(PostgresFixture fixture)
    {
        _fixture = fixture;
    }

    [Fact]
    public async Task CreateOrder_ShouldSaveOrderToDatabase()
    {
        // Arrange
        // Use the ConnectionString provided by the fixture
        await using var dbContext = new AppDbContext(_fixture.ConnectionString);
        var repository = new OrderRepository(dbContext);
        var newOrder = new Order { ... };

        // Act
        await repository.CreateAsync(newOrder);

        // Assert
        var savedOrder = await dbContext.Orders.FindAsync(newOrder.Id);
        Assert.NotNull(savedOrder);
    }
}
```

```csharp
[Collection("Postgres Collection")]
public class OrderRepositoryTests
{
    private readonly PostgresFixture _fixture;

    public OrderRepositoryTests(PostgresFixture fixture)
    {
        _fixture = fixture;
    }

    [Fact]
    public async Task CreateOrder_ShouldSaveOrderToDatabase()
    {
        // Arrange
        // Use the ConnectionString provided by the fixture
        await using var dbContext = new AppDbContext(_fixture.ConnectionString);
        var repository = new OrderRepository(dbContext);
        var newOrder = new Order { ... };

        // Act
        await repository.CreateAsync(newOrder);

        // Assert
        var savedOrder = await dbContext.Orders.FindAsync(newOrder.Id);
        Assert.NotNull(savedOrder);
    }
}
```

测试前载入 Fixture

```
[Collection("Postgres Collection")]
public class OrderRepositoryTests
{
    private readonly PostgresFixture _fixture;

    public OrderRepositoryTests(PostgresFixture fixture)
    {
        _fixture = fixture;
    }

    [Fact]
    public async Task CreateOrder_ShouldSaveOrderToDatabase()
    {
        // Arrange
        // Use the ConnectionString provided by the fixture
        await using var dbContext = new AppDbContext(_fixture.ConnectionString);
        var repository = new OrderRepository(dbContext);
        var newOrder = new Order { ... };

        // Act
        await repository.CreateAsync(newOrder);

        // Assert
        var savedOrder = await dbContext.Orders.FindAsync(newOrder.Id);
        Assert.NotNull(savedOrder);
    }
}
```

连接动态创建的数据库

```
[Collection("Postgres Collection")]
public class OrderRepositoryTests
{
    private readonly PostgresFixture _fixture;

    public OrderRepositoryTests(PostgresFixture fixture)
    {
        _fixture = fixture;
    }

    [Fact]
    public async Task CreateOrder_ShouldSaveOrderToDatabase()
    {
        // Arrange
        // Use the ConnectionString provided by the fixture
        await using var dbContext = new AppDbContext(_fixture.ConnectionString);
        var repository = new OrderRepository(dbContext);
        var newOrder = new Order { ... };

        // Act
        await repository.CreateAsync(newOrder);

        // Assert
        var savedOrder = await dbContext.Orders.FindAsync(newOrder.Id);
        Assert.NotNull(savedOrder);
    }
}
```

真实的数据库操作

在 Github Actions 中定义测试执行

```yaml
name: Integration Tests
on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - name: Setup .NET
      uses: actions/setup-dotnet@v4
      with:
        dotnet-version: '9.x'
    - name: Run tests
      run: dotnet test
```

Testcontainers 的实践建议

1. 避免为 Docker 资源（例如容器、网络和卷）分配静态名称，以防命名冲突导致测试失败。
2. 用使用 _container.Hostname 属性从测试主机访问容器，而不是 IP 地址。
3. 根据需要适当重复使用容器

```
_ = new ContainerBuilder()
    .WithReuse(true);
```

4. 不要装你不需要（没用）的容器
5. 标记慢测试，且只在发布分支或夜间构建上运行。
6. 分担 CI 工作，快速的 PR 单元测试，按需提供全套测试。
7. 容器启动不了，会造成测试执行为0.
   如果启动就用90秒，最好做一些提前准备

```
_ = Wait.ForUnixContainer()
    .UntilMessageIsLogged("Server started", o => o.WithTimeout(TimeSpan.FromMinutes(1)));
```

8. 不是每个测试都需要容器，有些场景模拟测试更好。

```
var keepAlive = true;
var container = new OllamaBuilder()
    .WithImage("ollama/ollama:0.6.6")
    .WithBindMount("/Users/yimingzhi/.ollama", "/root/.ollama")
    .WithPortBinding(11434, true)
    .Build();

container.Started += OnContainerStarted;

await container.StartAsync();

while (keepAlive)
    await Task.Delay(100);

await container.StopAsync();
await container.DisposeAsync();

Console.WriteLine("---- DONE ----");
```

```
async void OnContainerStarted(object? sender, EventArgs e)
{
    var ollamaPort = container.GetMappedPublicPort(11434);
    var ollamaBaseUrl = new Uri($"http://{container.Hostname}:{ollamaPort}");
    Console.WriteLine($"Ollama: {ollamaBaseUrl}");
    var client = new OllamaApiClient(ollamaBaseUrl);
    await foreach (var msg in client.ChatAsync(new ChatRequest
    {
        Model = "qwen3:0.6b",
        Messages =
        [
            new Message(ChatRole.System, "Translate English to Chinese. Just translate, no explanation."),
            new Message(ChatRole.User, "hello world")
        ],
        Stream = false
    }))
        Console.Write(msg?.Message.Content);

    Console.WriteLine();
    keepAlive = false;
}
```

```
1   var keepAlive = true;
2   var container = new OllamaBuilder()
3       .WithImage("ollama/ollama:0.6.6")
4       .WithBindMount("/Users/yimingzhi/.ollama", "/root/.ollama")
5       .WithPortBinding(11434, true)
6       .Build();
7
8   container.Started += OnContainerStarted;
9
10  await container.StartAsync();
11
12  while (keepAlive)
13      await Task.Delay(100);
14
15  await container.StopAsync();
16  await container.DisposeAsync();
17
18  Console.WriteLine("---- DONE ----");
```

```
20  async void OnContainerStarted(object? sender, EventArgs e)
21  {
22      var ollamaPort = container.GetMappedPublicPort(11434);
23      var ollamaBaseUrl = new Uri($"http://{container.Hostname}:{ollamaPort}");
24      Console.WriteLine($"Ollama: {ollamaBaseUrl}");
25      var client = new OllamaApiClient(ollamaBaseUrl);
26      await foreach (var msg in client.ChatAsync(new ChatRequest
27      {
28          Model = "qwen3:0.6b",
29          Messages =
30          [
31              new Message(ChatRole.System, "Translate English to Chinese. Just translate, no explanation."),
32              new Message(ChatRole.User, "hello world")
33          ],
34          Stream = false
35      }))
36          Console.Write(msg?.Message.Content);
37
38      Console.WriteLine();
39      keepAlive = false;
40  }
41
```

```
var keepAlive = true;
var container = new OllamaBuilder()
    .WithImage("ollama/ollama:0.6.6")
    .WithBindMount("/Users/yimingzhi/.ollama", "/root/.ollama")
    .WithPortBinding(11434, true)
    .Build();

container.Started += OnContainerStarted;

await container.StartAsync();

while (keepAlive)
    await Task.Delay(100);

await container.StopAsync();
await container.DisposeAsync();

Console.WriteLine("---- DONE ----");
```

```
async void OnContainerStarted(object? sender, EventArgs e)
{
    var ollamaPort = container.GetMappedPublicPort(11434);
    var ollamaBaseUrl = new Uri($"http://{container.Hostname}:{ollamaPort}");
    Console.WriteLine($"Ollama: {ollamaBaseUrl}");
    var client = new OllamaApiClient(ollamaBaseUrl);
    await foreach (var msg in client.ChatAsync(new ChatRequest
    {
        Model = "qwen3:0.6b",
        Messages =
        [
            new Message(ChatRole.System, "Translate English to Chinese. Just translate, no explanation."),
            new Message(ChatRole.User, "hello world")
        ],
        Stream = false
    }))
        Console.Write(msg?.Message.Content);

    Console.WriteLine();
    keepAlive = false;
}
```

```
var keepAlive = true;
var container = new OllamaBuilder()
    .WithImage("ollama/ollama:0.6.6")
    .WithBindMount("/Users/yimingzhi/.ollama", "/root/.ollama")
    .WithPortBinding(11434, true)
    .Build();

container.Started += OnContainerStarted;

await container.StartAsync();

while (keepAlive)
    await Task.Delay(100);

await container.StopAsync();
await container.DisposeAsync();

Console.WriteLine("---- DONE ----");
```

```
async void OnContainerStarted(object? sender, EventArgs e)
{
    var ollamaPort = container.GetMappedPublicPort(11434);
    var ollamaBaseUrl = new Uri($"http://{container.Hostname}:{ollamaPort}");
    Console.WriteLine($"Ollama: {ollamaBaseUrl}");
    var client = new OllamaApiClient(ollamaBaseUrl);
    await foreach (var msg in client.ChatAsync(new ChatRequest
    {
        Model = "qwen3:0.6b",
        Messages =
        [
            new Message(ChatRole.System, "Translate English to Chinese. Just translate, no explanation."),
            new Message(ChatRole.User, "hello world")
        ],
        Stream = false
    }))
        Console.Write(msg?.Message.Content);

    Console.WriteLine();
    keepAlive = false;
}
```

```
var keepAlive = true;
var container = new OllamaBuilder()
    .WithImage("ollama/ollama:0.6.6")
    .WithBindMount("/Users/yimingzhi/.ollama", "/root/.ollama")
    .WithPortBinding(11434, true)
    .Build();

container.Started += OnContainerStarted;

await container.StartAsync();

while (keepAlive)
    await Task.Delay(100);

await container.StopAsync();
await container.DisposeAsync();

Console.WriteLine("---- DONE ----");
```

```
async void OnContainerStarted(object? sender, EventArgs e)
{
    var ollamaPort = container.GetMappedPublicPort(11434);
    var ollamaBaseUrl = new Uri($"http://{container.Hostname}:{ollamaPort}");
    Console.WriteLine($"Ollama: {ollamaBaseUrl}");
    var client = new OllamaApiClient(ollamaBaseUrl);
    await foreach (var msg in client.ChatAsync(new ChatRequest
    {
        Model = "qwen3:0.6b",
        Messages =
        [
            new Message(ChatRole.System, "Translate English to Chinese. Just translate, no explanation."),
            new Message(ChatRole.User, "hello world")
        ],
        Stream = false
    }))
        Console.Write(msg?.Message.Content);

    Console.WriteLine();
    keepAlive = false;
}
```

Program.cs > Program > <top-level-statements-entry-point>

```
1  var keepAlive = true;
2  var container = new OllamaBuilder()
3      .WithImage("ollama/ollama:0.6.6")
4      .WithBindMount("/Users/yimingzhi/.ollama", "/root/.ollama
5      .WithPortBinding(11434, true)
6      .Build();
7
8  container.Started += OnContainerStarted;
9
10 await container.StartAsync();
11
12 while (keepAlive)
13     await Task.Delay(100);
14
15 await container.StopAsync();
16 await container.DisposeAsync();
17
18 Console.WriteLine("---- DONE ----");
19
20 async void OnContainerStarted(object? sender, EventArgs e)
21 {
22     var ollamaPort = container.GetMappedPublicPort(11434);
23     var ollamaBaseUrl = new Uri($"http://{container.Hostname}
24     Console.WriteLine($"Ollama: {ollamaBaseUrl}");
25     var client = new OllamaApiClient(ollamaBaseUrl);
26     await foreach (var msg in client.ChatAsync(new ChatReques
27     {
28         Model = "qwen3:0.6b",
29         Messages =
30         [
31             new Message(ChatRole.System, "Translate English t
32             new Message(ChatRole.User, "hello world")
33         ],
34         Stream = false
35     }))
36         Console.Write(msg?.Message.Content);
37
38     Console.WriteLine();
39     keepAlive = false;
40 }
```

```
yimingzhi …/testcontainers-demo1 main !+ 08:54
```

ctop - 08:55:14 CST 4 containers

NAME	CID	CPU	MEM	NET RX/TX	IO R/W	PIDS	UPTIME
buildx_buildkit_multi-platefor	bc85380ca6a5	-	-	-	-	-	-4s
my-redis	688dd0e19fe2	-	-	-	-	-	-19h43m3s
postgres	d5f7631585be	-	-	-	-	-	-19h43m4s
smtp4dev	474370ee5919	-	-	-	-	-	-19h43m5s

向量数据库相关模块

Java Go .NET Node.js

PgVector

DEPENDENCY:

```
dotnet add package Testcontainers.PostgreSql
```

USAGE:

```
var pgVectorContainer = new PostgreSqlBuilder()
  .WithImage("pgvector/pgvector:pg16")
  .Build();
await pgVectorContainer.StartAsync();
```

Java Go .NET Node.js Python

Qdrant

DEPENDENCY:

Copy

```
dotnet add package Testcontainers.Qdrant
```

USAGE:

```
var qdrantContainer = new QdrantBuilder()
  .WithImage("qdrant/qdrant:v1.13.4")
  .Build();
await qdrantContainer.StartAsync();
```

Java Go .NET Node.js Python Rust

Redis

DEPENDENCY:

```
dotnet add package Testcontainers.Redis
```

USAGE:

```
var redisContainer = new RedisBuilder()
  .WithImage("redis:7.0")
  .Build();
await redisContainer.StartAsync();
```

OpenSearch

……

Elasticsearch

Typesense

.NET Conf China 2025

改变世界 改变自己

THANK YOU