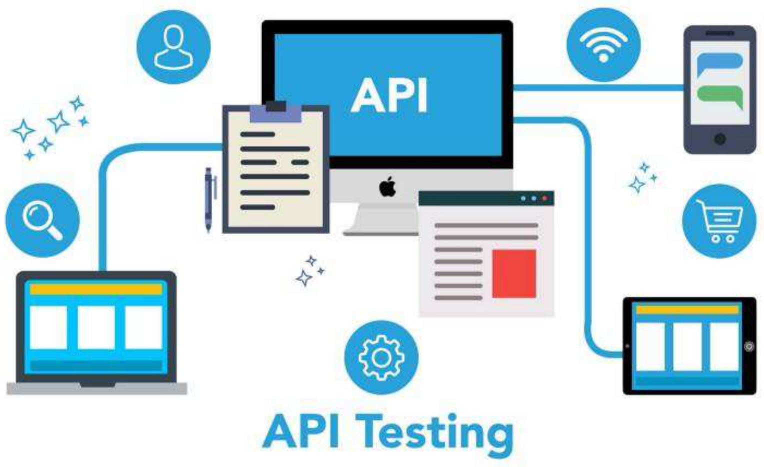
Integration Testing in Asp.Net Core 2.0

Aram Koukia Follow
Apr 22, 2018 · 3 min read



When you are building an API, and doing all the right unit tests and ad-hoc tests and etc, the next type of test in the test pyramid is Integration Test of API Tests, as in you make HTTP calls to your API and inspect the result.

Ideally you want a change to your code base to trigger all your unit tests as well as your integration tests to let the code change to be committed.

Now, one of the issues for API testing could be, you would need to deploy the code change to some environment and use an actual database and verify if the changes has broken anything.

But with .Net Core 2.0 and later, you don't need to host your API (with the new code changes) to an actual web server, and also you don't even need a real database and you can just run your tests as if they are unit tests!

Using Asp.Net Core Test Host

How? With Asp.Net Core Test Host package.

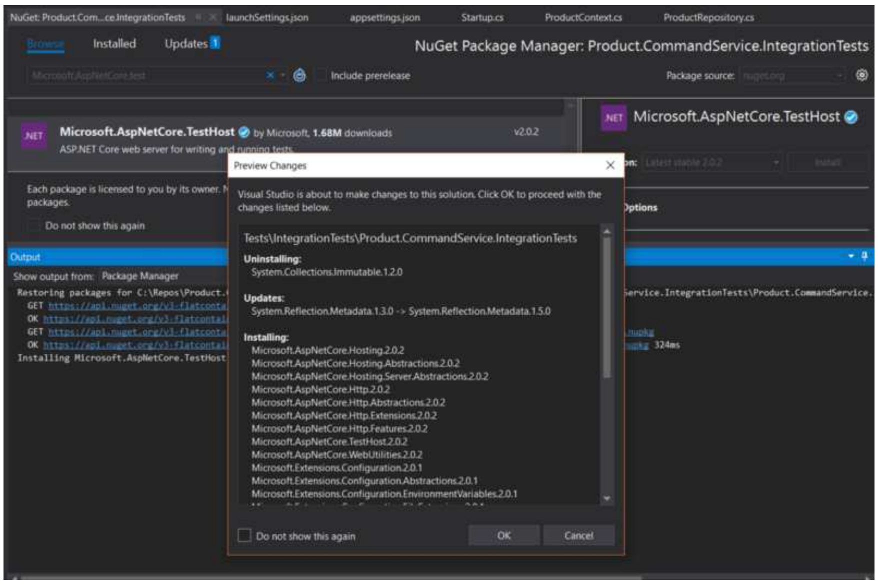
ASP.NET Core includes a test host that can be added to integration test projects and used to host ASP.NET Core applications, serving test requests without the need for a real web host.

Show me some code

Open your .Net Core solution and add a new test project.

Then add a reference to your .Net Core API project that has your Controller classes to the test project you just created.

In the test project, install the “Microsoft.AspNetCore.TestHost” nuget package.



Microsoft.AspNetCore.TestHost by Microsoft, **1.68M** downloads v2.0.2
ASP.NET Core web server for writing and running tests.

Preview Changes

Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.

Tests\IntegrationTests\Product.CommandService.IntegrationTests

Uninstalling:
System.Collections.Immutable.1.2.0

Updates:
System.Reflection.Metadata.1.3.0 -> System.Reflection.Metadata.1.5.0

Installing:
Microsoft.AspNetCore.Hosting.2.0.2
Microsoft.AspNetCore.Hosting.Abstractions.2.0.2
Microsoft.AspNetCore.Hosting.Server.Abstractions.2.0.2
Microsoft.AspNetCore.Http.2.0.2
Microsoft.AspNetCore.Http.Abstractions.2.0.2
Microsoft.AspNetCore.Http.Extensions.2.0.2
Microsoft.AspNetCore.Http.Features.2.0.2
Microsoft.AspNetCore.TestHost.2.0.2
Microsoft.AspNetCore.WebUtilities.2.0.2
Microsoft.Extensions.Configuration.2.0.1
Microsoft.Extensions.Configuration.Abstractions.2.0.1
Microsoft.Extensions.Configuration.EnvironmentVariables.2.0.1

Do not show this again

```
Restoring packages for C:\Repos\Product.
  GET https://api.nuget.org/v3-flatconta
  OK https://api.nuget.org/v3-flatcontai
  GET https://api.nuget.org/v3-flatconta
  OK https://api.nuget.org/v3-flatcontai
Installing Microsoft.AspNetCore.TestHost
```

This is going to give us access to Test Server class.

After you installed the `Microsoft.AspNetCore.TestHost` package in the test project, you can create and configure a `TestServer` to use in your integraion tests.

Now we are going to add a simple helper class to give us a Test Server instance. Let’s call it Test Server Fixture. Here is how it will look like:

```csharp
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.TestHost;
using Microsoft.Extensions.PlatformAbstractions;
using System;
using System.IO;
using System.Net.Http;

namespace Product.CommandService.IntegrationTests
{
    public class TestServerFixture : IDisposable
    {
        private readonly TestServer _testServer;
        public HttpClient Client { get; }

        public TestServerFixture()
        {
            var builder = new WebHostBuilder()
                   .UseContentRoot(GetContentRootPath())
                   .UseEnvironment("Development")
                   .UseStartup<Startup>();  // Uses Start u

            _testServer = new TestServer(builder);
            Client = _testServer.CreateClient();

        }

```

Now that we have our Test Server Fixture in place, we can go ahead and use it to build our first Integration Test.

Here is the actual Controller action that we want to add integration test for:

```csharp
namespace Product.CommandService.Controllers
{
    [Route("api/[controller]")]
    public class PingController : Controller
    {
        // GET api/ping
        [HttpGet]
        public string Get()
        {
```