Let's Talk HTTP in .NET Core

with IHttpClientFactory and Polly

Steve Gordon

@stevejgordon | stevejgordon.co.uk

https://bit.ly/dotnet-http

MVP Microsoft® Most Valuable Professional

.NET southeast

https://www.meetup.com/dotnetsoutheast

PLURALSIGHT

madgex

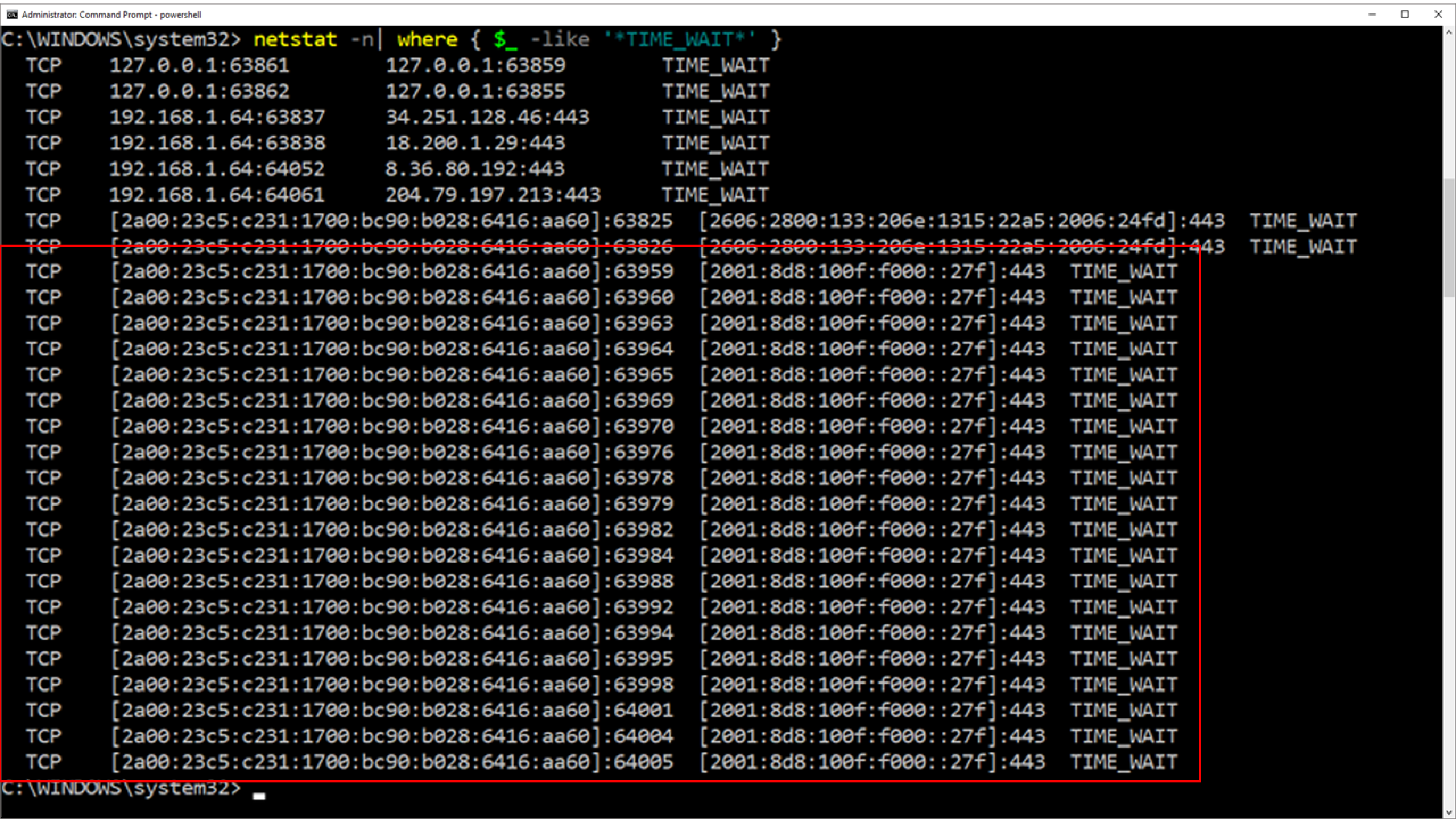
What we'll cover

- Why do we need IHttpClientFactory?
- How to use IHttpClientFactory
- Outgoing middleware
- Handling transient errors with Polly
- Patterns and recommendations
- HTTP improvements in .NET Core 2.1 to 3.1

NOTE: I'm generally speaking about .NET Core today.

```
public class GitHubController : ControllerBase
{
    [HttpGet]
    public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
    {
        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
            client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");

            var url = "https://api.github.com/orgs/aspnet/repos";
            var request = new HttpRequestMessage(HttpMethod.Get, url);

            var response = await _httpClient.SendAsync(request);

            var data = await response.Content.ReadAsStringAsync();
            return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
        }
    }
}
```

```
public class GitHubController : ControllerBase
{
   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      using (var client = new HttpClient())
      {
         client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
         client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");

         var url = "https://api.github.com/orgs/aspnet/repos";
         var request = new HttpRequestMessage(HttpMethod.Get, url);

         var response = await _httpClient.SendAsync(request);

         var data = await response.Content.ReadAsStringAsync();
         return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
      }
   }
}
```



Administrator: Command Prompt - powershell

```
C:\WINDOWS\system32> netstat -n| where { $_ -like '*TIME_WAIT*' }
  TCP    127.0.0.1:63861        127.0.0.1:63859        TIME_WAIT
  TCP    127.0.0.1:63862        127.0.0.1:63855        TIME_WAIT
  TCP    192.168.1.64:63837     34.251.128.46:443      TIME_WAIT
  TCP    192.168.1.64:63838     18.200.1.29:443        TIME_WAIT
  TCP    192.168.1.64:64052     8.36.80.192:443        TIME_WAIT
  TCP    192.168.1.64:64061     204.79.197.213:443     TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63825  [2606:2800:133:206e:1315:22a5:2006:24fd]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63826  [2606:2800:133:206e:1315:22a5:2006:24fd]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63959  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63960  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63963  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63964  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63965  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63969  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63970  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63976  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63978  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63979  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63982  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63984  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63988  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63992  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63994  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63995  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:63998  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:64001  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:64004  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
  TCP    [2a00:23c5:c231:1700:bc90:b028:6416:aa60]:64005  [2001:8d8:100f:f000::27f]:443  TIME_WAIT
C:\WINDOWS\system32>
```

WHAT'S THE SOLUTION?

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseMvc();
    }
}
```

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseMvc();
    }
}
```

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<HttpClient>();
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseMvc();
    }
}
```

```
public class GitHubController : ControllerBase
{
   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      using (var client = new HttpClient())
      {
         client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
         client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");

         var url = "https://api.github.com/orgs/aspnet/repos";
         var request = new HttpRequestMessage(HttpMethod.Get, url);

         var response = await client.SendAsync(request);

         var data = await response.Content.ReadAsStringAsync();
         return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
      }
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly HttpClient _httpClient;

   public GitHubController(HttpClient httpClient) => _httpClient = httpClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      using (var client = new HttpClient())
      {
         client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
         client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");

         var url = "https://api.github.com/orgs/aspnet/repos";
         var request = new HttpRequestMessage(HttpMethod.Get, url);

         var response = await client.SendAsync(request);

         var data = await response.Content.ReadAsStringAsync();
         return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
      }
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly HttpClient _httpClient;

   public GitHubController(HttpClient httpClient) => _httpClient = httpClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      using (var client = new HttpClient())
      {
         client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
         client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");

         var url = "https://api.github.com/orgs/aspnet/repos";
         var request = new HttpRequestMessage(HttpMethod.Get, url);

         var response = await client.SendAsync(request);

         var data = await response.Content.ReadAsStringAsync();
         return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
      }
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly HttpClient _httpClient;

   public GitHubController(HttpClient httpClient) => _httpClient = httpClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
      client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");

      var url = "https://api.github.com/orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await client.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly HttpClient _httpClient;

   public GitHubController(HttpClient httpClient) => _httpClient = httpClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "https://api.github.com/orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      request.Headers.Add("Accept", "application/vnd.github.v3+json");
      request.Headers.Add("User-Agent", "my-user-agent");

      var response = await client.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly HttpClient _httpClient;

   public GitHubController(HttpClient httpClient) => _httpClient = httpClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "https://api.github.com/orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      request.Headers.Add("Accept", "application/vnd.github.v3+json");
      request.Headers.Add("User-Agent", "my-user-agent");

      var response = await client.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly HttpClient _httpClient;

   public GitHubController(HttpClient httpClient) => _httpClient = httpClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "https://api.github.com/orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      request.Headers.Add("Accept", "application/vnd.github.v3+json");
      request.Headers.Add("User-Agent", "my-user-agent");

      var response = await _httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

BUT…

DNS
(Pre .NET Core 2.1)

IHttpClientFactory

- Manages the lifetime of HttpMessageHandlers
- Provides a central location for naming and configuring logical HttpClients
- Codifies the concept of outgoing middleware via delegating handlers
- Integrates with Polly for transient-fault handling

CONVERTING OUR CODE

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<HttpClient>();
        services.AddMvc();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseMvc();
    }
}
```

```
public class Startup
{
   public Startup(IConfiguration configuration)
   {
      Configuration = configuration;
   }

   public IConfiguration Configuration { get; }

   public void ConfigureServices(IServiceCollection services)
   {
      services.AddHttpClient();
      services.AddMvc();
   }

   public void Configure(IApplicationBuilder app, IHostingEnvironment env)
   {
      app.UseMvc();
   }
}
```

```csharp
public class GitHubController : ControllerBase
{
   private readonly HttpClient _httpClient;

   public GitHubController(HttpClient httpClient) => _httpClient = httpClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "https://api.github.com/orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      request.Headers.Add("Accept", "application/vnd.github.v3+json");
      request.Headers.Add("User-Agent", "my-user-agent");

      var response = await _httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly HttpClient _httpClient;

   public GitHubController(HttpClient httpClient) => _httpClient = httpClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "https://api.github.com/orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      request.Headers.Add("Accept", "application/vnd.github.v3+json");
      request.Headers.Add("User-Agent", "my-user-agent");

      var response = await _httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
    private readonly IHttpClientFactory _factory;

    public GitHubController(IHttpClientFactory factory) => _factory = factory;

    [HttpGet]
    public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
    {
        var url = "https://api.github.com/orgs/aspnet/repos";
        var request = new HttpRequestMessage(HttpMethod.Get, url);

        request.Headers.Add("Accept", "application/vnd.github.v3+json");
        request.Headers.Add("User-Agent", "my-user-agent");

        var response = await _httpClient.SendAsync(request);

        var data = await response.Content.ReadAsStringAsync();
        return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
    }
}
```

```
public class GitHubController : ControllerBase
{
    private readonly IHttpClientFactory _factory;

    public GitHubController(IHttpClientFactory factory) => _factory = factory;

    [HttpGet]
    public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
    {
        var url = "https://api.github.com/orgs/aspnet/repos";
        var request = new HttpRequestMessage(HttpMethod.Get, url);

        request.Headers.Add("Accept", "application/vnd.github.v3+json");
        request.Headers.Add("User-Agent", "my-user-agent");

        var response = await _httpClient.SendAsync(request);

        var data = await response.Content.ReadAsStringAsync();
        return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
    }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly IHttpClientFactory _factory;

   public GitHubController(IHttpClientFactory factory) => _factory = factory;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "https://api.github.com/orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      request.Headers.Add("Accept", "application/vnd.github.v3+json");
      request.Headers.Add("User-Agent", "my-user-agent");

      var httpClient = _factory.CreateClient();
      var response = await httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

NAMED CLIENTS

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient();
        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient("github", client =>
        {
            client.BaseAddress = new Uri("https://api.github.com/");
            client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
            client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");
        });
        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient("github", client =>
        {
            client.BaseAddress = new Uri("https://api.github.com/");
            client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
            client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");
        });
        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient("github", client =>
        {
            client.BaseAddress = new Uri("https://api.github.com/");
            client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
            client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");
        });
        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class GitHubController : ControllerBase
{
    private readonly IHttpClientFactory _factory;

    public GitHubController(IHttpClientFactory factory) => _factory = factory;

    [HttpGet]
    public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
    {
        var url = "https://api.github.com/orgs/aspnet/repos";
        var request = new HttpRequestMessage(HttpMethod.Get, url);

        request.Headers.Add("Accept", "application/vnd.github.v3+json");
        request.Headers.Add("User-Agent", "my-user-agent");

        var httpClient = _factory.CreateClient();
        var response = await httpClient.SendAsync(request);

        var data = await response.Content.ReadAsStringAsync();
        return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
    }
}
```

```csharp
public class GitHubController : ControllerBase
{
    private readonly IHttpClientFactory _factory;

    public GitHubController(IHttpClientFactory factory) => _factory = factory;

    [HttpGet]
    public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
    {
        var url = "orgs/aspnet/repos";
        var request = new HttpRequestMessage(HttpMethod.Get, url);

        request.Headers.Add("Accept", "application/vnd.github.v3+json");
        request.Headers.Add("User-Agent", "my-user-agent");

        var httpClient = _factory.CreateClient();
        var response = await httpClient.SendAsync(request);

        var data = await response.Content.ReadAsStringAsync();
        return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
    }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly IHttpClientFactory _factory;

   public GitHubController(IHttpClientFactory factory) => _factory = factory;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      request.Headers.Add("Accept", "application/vnd.github.v3+json");
      request.Headers.Add("User-Agent", "my-user-agent");

      var httpClient = _factory.CreateClient();
      var response = await httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly IHttpClientFactory _factory;

   public GitHubController(IHttpClientFactory factory) => _factory = factory;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var httpClient = _factory.CreateClient();
      var response = await httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
    private readonly IHttpClientFactory _factory;

    public GitHubController(IHttpClientFactory factory) => _factory = factory;

    [HttpGet]
    public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
    {
        var url = "orgs/aspnet/repos";
        var request = new HttpRequestMessage(HttpMethod.Get, url);

        var httpClient = _factory.CreateClient();
        var response = await httpClient.SendAsync(request);

        var data = await response.Content.ReadAsStringAsync();
        return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
    }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly IHttpClientFactory _factory;

   public GitHubController(IHttpClientFactory factory) => _factory = factory;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var httpClient = _factory.CreateClient("github");
      var response = await httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

TYPED CLIENTS

```
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      client.BaseAddress = new Uri("https://api.github.com/");
      client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
      client.DefaultRequestHeaders.Add("User-Agent", "my_user_agent");
      _httpClient = client;
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data);
   }
}
```

```
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      client.BaseAddress = new Uri("https://api.github.com/");
      client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
      client.DefaultRequestHeaders.Add("User-Agent", "my_user_agent");
      _httpClient = client;
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data);
   }
}
```

```
public class GitHubClient : IGitHubClient
{
    private readonly HttpClient _httpClient;

    public GitHubClient(HttpClient client)
    {
        client.BaseAddress = new Uri("https://api.github.com/");
        client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
        client.DefaultRequestHeaders.Add("User-Agent", "my_user_agent");
        _httpClient = client;
    }

    public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
    {
        var url = "orgs/aspnet/repos";
        var request = new HttpRequestMessage(HttpMethod.Get, url);

        var response = await _httpClient.SendAsync(request);

        var data = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data);
    }
}
```

```
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      client.BaseAddress = new Uri("https://api.github.com/");
      client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
      client.DefaultRequestHeaders.Add("User-Agent", "my_user_agent");
      _httpClient = client;
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data);
   }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient("github", client =>
        {
            client.BaseAddress = new Uri("https://api.github.com/");
            client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
            client.DefaultRequestHeaders.Add("User-Agent", "my-user-agent");
        });
        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient<IGitHubClient, GitHubClient>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

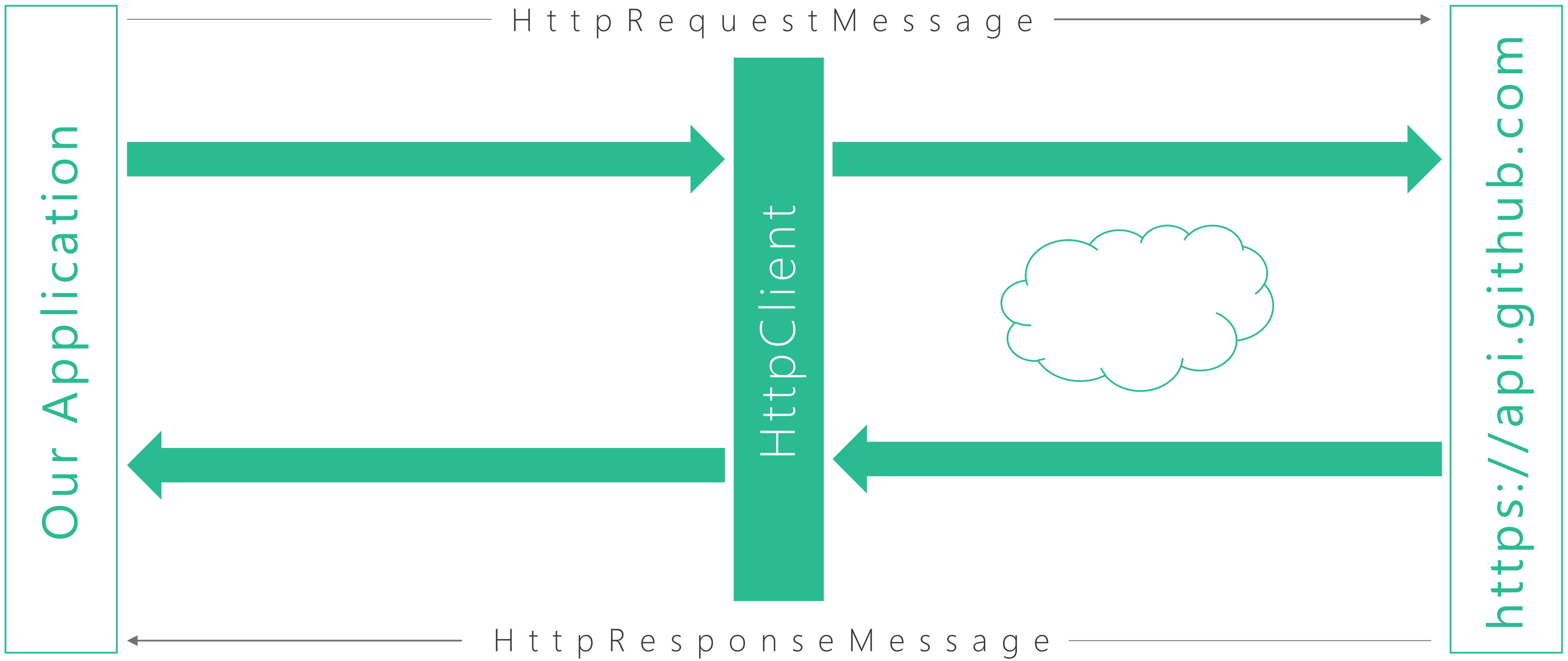
```csharp
public class GitHubController : ControllerBase
{
   private readonly IHttpClientFactory _factory;

   public GitHubController(IHttpClientFactory factory) => _factory = factory;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var httpClient = _factory.CreateClient("github");
      var response = await httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly IGitHubClient _gitHubClient;

   public GitHubController(IGitHubClient gitHubClient) => _gitHubClient = gitHubClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var httpClient = _factory.CreateClient("github");
      var response = await httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

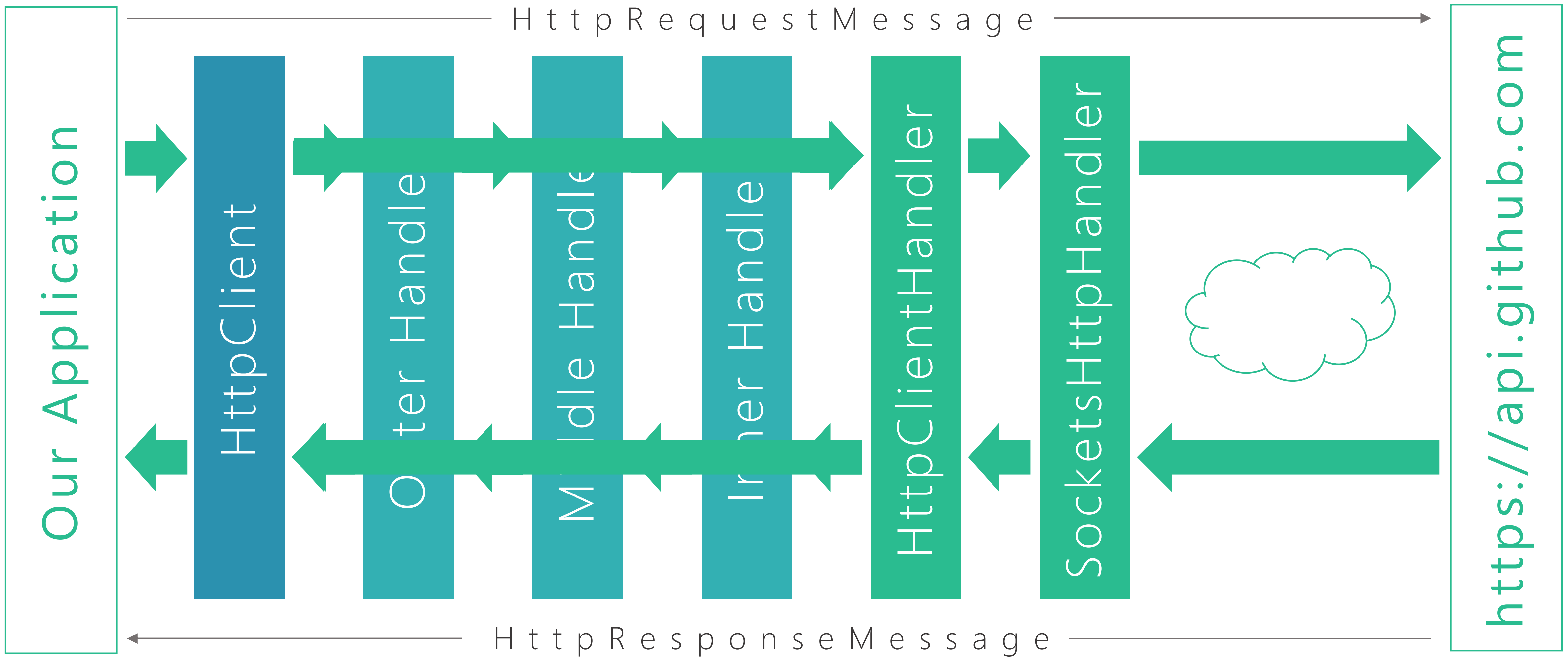
```csharp
public class GitHubController : ControllerBase
{
   private readonly IGitHubClient _gitHubClient;

   public GitHubController(IGitHubClient gitHubClient) => _gitHubClient = gitHubClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var httpClient = _factory.CreateClient("github");
      var response = await httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return Ok(JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data));
   }
}
```

```
public class GitHubController : ControllerBase
{
    private readonly IGitHubClient _gitHubClient;

    public GitHubController(IGitHubClient gitHubClient) => _gitHubClient = gitHubClient;

    [HttpGet]
    public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
    {
        var data = await _gitHubClient.GetAspNetReposAsync();
        return Ok(data);
    }
}
```

Outgoing "middleware"



Outgoing "middleware"



DELEGATING HANDLERS

```
public class StatusCodeMetricHandler : DelegatingHandler
{
   private readonly IMonitoringService _monitoringService;

   public StatusCodeMetricHandler(IMonitoringService monitoringService) =>
      _monitoringService = monitoringService;

   protected override async Task<HttpResponseMessage> SendAsync(
      HttpRequestMessage request,
      CancellationToken ct)
   {
      // note: we could do things with the request before passing it along

      var response = await base.SendAsync(request, ct);

      _monitoringService.RecordStatusCodeMetric((int)response.StatusCode);

      return response;
   }
}
```

```
public class StatusCodeMetricHandler : DelegatingHandler
{
   private readonly IMonitoringService _monitoringService;

   public StatusCodeMetricHandler(IMonitoringService monitoringService) =>
      _monitoringService = monitoringService;

   protected override async Task<HttpResponseMessage> SendAsync(
      HttpRequestMessage request,
      CancellationToken ct)
   {
      // note: we could do things with the request before passing it along

      var response = await base.SendAsync(request, ct);

      _monitoringService.RecordStatusCodeMetric((int)response.StatusCode);

      return response;
   }
}
```

```
public class StatusCodeMetricHandler : DelegatingHandler
{
   private readonly IMonitoringService _monitoringService;

   public StatusCodeMetricHandler(IMonitoringService monitoringService) =>
      _monitoringService = monitoringService;

   protected override async Task<HttpResponseMessage> SendAsync(
      HttpRequestMessage request,
      CancellationToken ct)
   {
      // note: we could do things with the request before passing it along

      var response = await base.SendAsync(request, ct);

      _monitoringService.RecordStatusCodeMetric((int)response.StatusCode);

      return response;
   }
}
```

```
public class StatusCodeMetricHandler : DelegatingHandler
{
   private readonly IMonitoringService _monitoringService;

   public StatusCodeMetricHandler(IMonitoringService monitoringService) =>
      _monitoringService = monitoringService;

   protected override async Task<HttpResponseMessage> SendAsync(
      HttpRequestMessage request,
      CancellationToken ct)
   {
      // note: we could do things with the request before passing it along

      var response = await base.SendAsync(request, ct);

      _monitoringService.RecordStatusCodeMetric((int)response.StatusCode);

      return response;
   }
}
```

```
public class StatusCodeMetricHandler : DelegatingHandler
{
   private readonly IMonitoringService _monitoringService;

   public StatusCodeMetricHandler(IMonitoringService monitoringService) =>
      _monitoringService = monitoringService;

   protected override async Task<HttpResponseMessage> SendAsync(
      HttpRequestMessage request,
      CancellationToken ct)
   {
      // note: we could do things with the request before passing it along

      var response = await base.SendAsync(request, ct);

      _monitoringService.RecordStatusCodeMetric((int)response.StatusCode);

      return response;
   }
}
```

```
public class StatusCodeMetricHandler : DelegatingHandler
{
   private readonly IMonitoringService _monitoringService;

   public StatusCodeMetricHandler(IMonitoringService monitoringService) =>
      _monitoringService = monitoringService;

   protected override async Task<HttpResponseMessage> SendAsync(
      HttpRequestMessage request,
      CancellationToken ct)
   {
      // note: we could do things with the request before passing it along

      var response = await base.SendAsync(request, ct);

      _monitoringService.RecordStatusCodeMetric((int)response.StatusCode);

      return response;
   }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient<IGitHubClient, GitHubClient>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IMonitoringService, MonitoringService>();
        services.AddTransient<StatusCodeMetricHandler>();

        services.AddHttpClient<IGitHubClient, GitHubClient>()
            .AddHttpMessageHandler<StatusCodeMetricHandler>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IMonitoringService, MonitoringService>();
        services.AddTransient<StatusCodeMetricHandler>();

        services.AddHttpClient<IGitHubClient, GitHubClient>()
            .AddHttpMessageHandler<StatusCodeMetricHandler>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

Polly

- Polly is a .NET resilience and transient-fault-handling library
- Integrates easily with IHttpClientFactory

- Retry
- Circuit-breaker
- Timeout
- Bulkhead isolation
- Cache
- Fallback
- PolicyWrap

HANDLING TRANSIENT ERRORS WITH POLLY

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IMonitoringService, MonitoringService>();
        services.AddTransient<StatusCodeMetricHandler>();

        services.AddHttpClient<IGitHubClient, GitHubClient>()
            .AddHttpMessageHandler<StatusCodeMetricHandler>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IMonitoringService, MonitoringService>();
        services.AddTransient<StatusCodeMetricHandler>();

        services.AddHttpClient<IGitHubClient, GitHubClient>()
            .AddTransientHttpErrorPolicy(builder =>
                builder.WaitAndRetryAsync(3, retryCount =>
                    TimeSpan.FromSeconds(Math.Pow(2, retryCount))))
            .AddHttpMessageHandler<StatusCodeMetricHandler>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IMonitoringService, MonitoringService>();
        services.AddTransient<StatusCodeMetricHandler>();

        services.AddHttpClient<IGitHubClient, GitHubClient>()
            .AddTransientHttpErrorPolicy(builder =>
                builder.WaitAndRetryAsync(3, retryCount =>
                    TimeSpan.FromSeconds(Math.Pow(2, retryCount))))
            .AddHttpMessageHandler<StatusCodeMetricHandler>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IMonitoringService, MonitoringService>();
        services.AddTransient<StatusCodeMetricHandler>();

        services.AddHttpClient<IGitHubClient, GitHubClient>()
            .AddTransientHttpErrorPolicy(builder =>
                builder.WaitAndRetryAsync(3, retryCount =>
                    TimeSpan.FromSeconds(Math.Pow(2, retryCount))))
            .AddHttpMessageHandler<StatusCodeMetricHandler>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IMonitoringService, MonitoringService>();
        services.AddTransient<StatusCodeMetricHandler>();

        services.AddHttpClient<IGitHubClient, GitHubClient>()
            .AddTransientHttpErrorPolicy(builder =>
                builder.WaitAndRetryAsync(3, retryCount =>
                    TimeSpan.FromSeconds(Math.Pow(2, retryCount))))
            .AddHttpMessageHandler<StatusCodeMetricHandler>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

```
public class Startup
{
    ...

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IMonitoringService, MonitoringService>();
        services.AddTransient<StatusCodeMetricHandler>();

        services.AddHttpClient<IGitHubClient, GitHubClient>()
            .AddTransientHttpErrorPolicy(builder =>
                builder.WaitAndRetryAsync(3, retryCount =>
                    TimeSpan.FromSeconds(Math.Pow(2, retryCount))))
            .AddHttpMessageHandler<StatusCodeMetricHandler>();

        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    { … }
}
```

OTHER HTTP TIPS

RESPONSE CONTENT ALLOCATIONS

```csharp
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      client.BaseAddress = new Uri("https://api.github.com/");
      client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
      client.DefaultRequestHeaders.Add("User-Agent", "my_user_agent");
      _httpClient = client;
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data);
   }
}
```

```
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      client.BaseAddress = new Uri("https://api.github.com/");
      client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
      client.DefaultRequestHeaders.Add("User-Agent", "my_user_agent");
      _httpClient = client;
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      var data = await response.Content.ReadAsStringAsync();
      return JsonConvert.DeserializeObject<IEnumerable<GitHubRepo>>(data);
   }
}
```

```
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      client.BaseAddress = new Uri("https://api.github.com/");
      client.DefaultRequestHeaders.Add("Accept", "application/vnd.github.v3+json");
      client.DefaultRequestHeaders.Add("User-Agent", "my_user_agent");
      _httpClient = client;
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      return await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>();
   }
}
```

HANDLING ERRORS

```
public class GitHubClient : IGitHubClient
{
   // ctor - hidden for brevity ...

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      return await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>();
   }
}
```

```
public class GitHubClient : IGitHubClient
{
    // ctor - hidden for brevity ...

    public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
    {
        var url = "orgs/aspnet/repos";
        var request = new HttpRequestMessage(HttpMethod.Get, url);

        var response = await _httpClient.SendAsync(request);

        response.EnsureSuccessStatusCode();

        return await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>();
    }
}
```

```
public class GitHubClient : IGitHubClient
{
   // ctor - hidden for brevity ...

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      return response.IsSuccessStatusCode
         ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>()
         : Array.Empty<GitHubRepo>();
   }
}
```

HANDLING CANCELLATION

```
public class GitHubClient : IGitHubClient
{
    private readonly HttpClient _httpClient;

    public GitHubClient(HttpClient client)
    {
        // hidden for brevity ...
    }

    public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync()
    {
        var url = "orgs/aspnet/repos";
        var request = new HttpRequestMessage(HttpMethod.Get, url);

        var response = await _httpClient.SendAsync(request);

        return response.IsSuccessStatusCode
            ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>()
            : Array.Empty<GitHubRepo>();
    }
}
```

```
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      // hidden for brevity ...
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync(CancellationToken ct = default)
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      var response = await _httpClient.SendAsync(request);

      return response.IsSuccessStatusCode
         ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>()
         : Array.Empty<GitHubRepo>();
   }
}
```

```csharp
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      // hidden for brevity ...
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync(CancellationToken ct = default)
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      ct.ThrowIfCancellationRequested();

      var response = await _httpClient.SendAsync(request);

      return response.IsSuccessStatusCode
         ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>()
         : Array.Empty<GitHubRepo>();
   }
}
```

```
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      // hidden for brevity ...
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync(CancellationToken ct = default)
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      ct.ThrowIfCancellationRequested();

      var response = await _httpClient.SendAsync(request, ct);

      return response.IsSuccessStatusCode
         ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>()
         : Array.Empty<GitHubRepo>();
   }
}
```

```
public class GitHubClient : IGitHubClient
{
   private readonly HttpClient _httpClient;

   public GitHubClient(HttpClient client)
   {
      // hidden for brevity ...
   }

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync(CancellationToken ct = default)
   {
      var url = "orgs/aspnet/repos";
      var request = new HttpRequestMessage(HttpMethod.Get, url);

      ct.ThrowIfCancellationRequested();

      var response = await _httpClient.SendAsync(request, ct);

      return response.IsSuccessStatusCode
         ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>(ct)
         : Array.Empty<GitHubRepo>();
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly IGitHubClient _gitHubClient;

   public GitHubController(IGitHubClient gitHubClient) => _gitHubClient = gitHubClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos()
   {
      var data = await _gitHubClient.GetAspNetReposAsync();
      return Ok(data);
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly IGitHubClient _gitHubClient;

   public GitHubController(IGitHubClient gitHubClient) => _gitHubClient = gitHubClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos(CancellationToken ct)
   {
      var data = await _gitHubClient.GetAspNetReposAsync(ct);
      return Ok(data);
   }
}
```

```
public class GitHubController : ControllerBase
{
   private readonly IGitHubClient _gitHubClient;

   public GitHubController(IGitHubClient gitHubClient) => _gitHubClient = gitHubClient;

   [HttpGet]
   public async Task<ActionResult<IEnumerable<GitHubRepo>>> GetAspNetRepos(CancellationToken ct)
   {
      var data = await _gitHubClient.GetAspNetReposAsync(ct);
      return Ok(data);
   }
}
```

HttpCompletionOption

```
public class GitHubClient : IGitHubClient
{
   // ctor - hidden for brevity ...

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync(CancellationToken ct = default)
   {
      var req = new HttpRequestMessage(HttpMethod.Get, "orgs/aspnet/repos");

      ct.ThrowIfCancellationRequested();

      var response = await _httpClient.SendAsync(req, ct);

      return response.IsSuccessStatusCode
         ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>(ct)
         : Array.Empty<GitHubRepo>();
   }
}
```

```
public class GitHubClient : IGitHubClient
{
   // ctor - hidden for brevity ...

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync(CancellationToken ct = default)
   {
      var req = new HttpRequestMessage(HttpMethod.Get, "orgs/aspnet/repos");

      ct.ThrowIfCancellationRequested();

      var response = await _httpClient.SendAsync(req, HttpCompletionOption.ResponseHeadersRead, ct);

      return response.IsSuccessStatusCode
         ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>(ct)
         : Array.Empty<GitHubRepo>();
   }
}
```

```
public class GitHubClient : IGitHubClient
{
   // ctor - hidden for brevity ...

   public async Task<IEnumerable<GitHubRepo>> GetAspNetReposAsync(CancellationToken ct = default)
   {
      var req = new HttpRequestMessage(HttpMethod.Get, "orgs/aspnet/repos");

      ct.ThrowIfCancellationRequested();

      var response = await _httpClient.SendAsync(req, HttpCompletionOption.ResponseHeadersRead, ct);

      try
      {
         return response.IsSuccessStatusCode
            ? await response.Content.ReadAsAsync<IEnumerable<GitHubRepo>>(ct)
            : Array.Empty<GitHubRepo>();
      }
      finally
      {
         response.Dispose();
      }
   }
}
```

WHEN SHOULD I DISPOSE?...

…IT DEPENDS!

When Should I Dispose?

- **HttpClient** (using IHttpClientFactory) – No effect
- **HttpClient** (without IHttpClientFactory) – Generally never
- **HttpRequestMessage** – Only has an effect (today) if sending StreamContent
- **HttpResponseMessage** – No effect unless using ResponseHeadersRead

SocketsHttpHandler

- Built on top of System.Net.Sockets
- Managed code
- Elimination of platform dependencies on libcurl (for Linux) and WinHTTP (for Windows)
- Consistent behaviour across platforms.
- Enabled by default in .NET Core 2.1+ (No HTTP/2 support until 3.0)

Connection Pooling

- PooledConnectionLifetime similar to ServicePointManager
- Set the max lifetime for a connection
- Also possible to set the PooledConnectionIdleTimeout
- Re-use the same HttpClient and handler chain without DNS concerns

.NET Core 3.0

- HTTP/2 is supported (primarily for gRPC scenarios)
- HTTP/2 is not enabled by default
- HttpClient now includes a Version property which applies to all messages sent via that client (HTTP/1.1 by default)

In Summary

- Use IHttpClientFactory in .NET Core 2.1+
- Use Polly for transient-fault handling
- Beware content.ReadAsStringAsync and client.GetStringAsync
- Expect and handle errors
- Pass cancellation tokens
- Dispose appropriately (generally not required)
- Go enjoy talking HTTP! ☺

Thanks for listening!

@stevejgordon | stevejgordon.co.uk

http://bit.ly/dotnet-http

https://www.meetup.com/dotnetsoutheast