

# Technologies

- [Http Client](#)

# Http Client

**“you should use either long-lived clients and set `PooledConnectionLifetime` (.NET Core and .NET 5+) or short-lived clients created by `IHttpClientFactory`.**

\*see [HttpClient guidelines for .NET - .NET | Microsoft Learn](#)

## How?

In the `Startup` file (or `Program.cs` file in cases, wherever you are doing dependency injection is where you want to do this), do

```
builder.Services.AddHttpClient();
```

This will allow you to dependency inject `IHttpClientFactory` in other classes, which you can use to create short lived `HttpClient` objects whose connections will be cleaned up immediately after disposal with the following code.

```
using HttpClient httpClient = HttpClientFactory.CreateClient();
```

For example in Atlantis Front-End of how we did this, reference the following files:

- [Program.cs - Repos \(azure.com\)](#)
- [BaseController.cs - Repos \(azure.com\)](#)

---

Another way to do this is by dependency injecting a singleton `HttpClient`. I (Quinn) had trouble getting the `IHttpClientFactory` to dependency inject in functions apps. I kept running into dependency issues with different nuget packages, but this approach worked great.

In your `startup.cs` add the following

```
builder.Services.AddSingleton(() => new HttpClient(  
    new SocketsHttpHandler { PooledConnectionLifetime = TimeSpan.FromMinutes(2) }  
));
```

Once you do this, you can add a `HttpClient` as a parameter to any constructor of a class that is dependency injected.

The approach this way is different than the one above because you aren't creating short lived `HttpClient`s, but instead, you have a singleton `HttpClient`. This is fine because most functions are thread safe, but you won't want to dispose of the `HttpClient` anywhere.

## Why?

In the Atlantis Front-End we learned this the hard way. We were spinning up a new `HttpClient` object every time a request was sent to Utopia or the Atlantis Back End (which happens for basically every request to the Atlantis Front End server). When we opted in all users to the new experience, we got a lot more traffic and starting seeing thousands of failed requests with the following error message:

*An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full. (utopia.revverdocs.com:443) An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.*

After researching a bit, we learned that each `HttpClient` object has it's own connection pool, and even when you dispose of the `HttpClient` object, it's connection pool stays alive and keeps all connections in the pool alive for 2 minutes (this time is configurable though). While the connection is still alive, it blocks another connection from being created on the same port, and your server does have a limited number of ports, so if you have a bunch of connections blocking all your ports from being used, you start to get the error above because there are no ports available to send you web request out on.