Getting Correct Scheme, Host, Port Behind a Proxy

When running an API behind a reverse proxy, such as NGINX, the service will not, by default, see the scheme and port of the incoming call. By default, the API service will see the scheme and port of the direct call to it, which will likely be http and some internal port (5000 maybe).

This is often because the reverse proxy is terminating SSL, and forwarding http requests to your API.

The problem with this, is that your API has no way to correctly compose Urls for redirection, because it doesn't know the scheme and port.

To ensure that your API can see the correct scheme, host, and port, you need to do the following things.

> NOTE: If your API service can be called from more than one NGINX server block, each with a different server_name, see this page for how to create a URIService instance from the forwarded info this page provides:
>
> URI Service Behind a Hostname Separated NGINX

NGINX Config

In the server block of your reverse proxy, verify you have the following three directives:

```
# These forward the scheme type (http or https), hostname, and listening port.
# These are used by the service, when generating redirect urls.
proxy_set_header X-Forwarded-Proto $scheme;
# NOTE: This line, explicitly, includes the port in the host string.
proxy_set_header Host $host:$server_port;
proxy_set_header X-Forwarded-Port $server_port;
```

Startup.cs

The dotnet service doesn't accept the forwarded headers, by default. So, we need to configure it to accept them.
This is done in the Russian doll logic of your Startup:Configure() routine.

Locate the Configure() of your Startup.cs, and find the UseRouting(); line.
Add the following block after UseRouting():

```
// Setup Header Forwarding, to accept headers from NGINX, for scheme, host, and port...
// This is needed if we are running behind a reverse proxy, and our service includes logic that composes redirecting URLs.
{
    // Accept the forwarded headers (scheme, host, port) from NGINX.
    // We do this, because we are running behind a reverse proxy, and don't directly know the scheme and port.
    // So, we have NGINX forward these to us.
    // And, this statement accepts them.
    // See this page for details: https://wiki.galaxydump.com/link/476

    // Create the options instance...
    var options = new ForwardedHeadersOptions();
    options.ForwardedHeaders = ForwardedHeaders.XForwardedProto |
                               ForwardedHeaders.XForwardedHost |
                               ForwardedHeaders.XForwardedFor;
    // These next four lines exist because, by default, ForwardedHeadersOptions.KnownProxies only allow proxy IP to be 127.0.0.1.
    // So, that prevents a reverse proxy from passing the headers we require.
    // You can restrict this to a specific IP of reverse proxy, but we leave it open, here.
    options.KnownNetworks.Clear();
    options.KnownProxies.Clear();
    options.KnownProxies.Add(IPAddress.Any);
    options.KnownNetworks.Add(new IPNetwork(IPAddress.Any, 0));
    // Apply our options...
    app.UseForwardedHeaders(options);
}
```

Usage

Once you have NGINX forwarding the correct scheme, host, and port, and your Startup.cs is accepting it, you can access them like this:

```
HttpRequest request = httpContext!.Request;

// Retrieve the scheme that was forwarded by NGINX...
// This is set by the following directive in NGINX:
//  proxy_set_header X-Forwarded-Proto $scheme;
var scheme = request.Scheme;
// Retrieve the host that was forwarded by NGINX...
// This is set by the following directive in NGINX:
//  proxy_set_header Host $host;
var hostname = request.Host.Host;
// Retrieve the port that was forwarded by NGINX...
// This is set by the following directive in NGINX:
//  proxy_set_header X-Forwarded-Port $server_port;
var port = request.Host.Port;
```

A further example of usage, is to correctly compose a base URI from what NGINX told us about the request.

The following will retrieve the scheme, host, and port.
And, it will compose a base URI that matches what NGINX saw from the original call.

```
HttpRequest request = httpContext!.Request;

// Retrieve the scheme that was forwarded by NGINX...
// This is set by the following directive in NGINX:
//  proxy_set_header X-Forwarded-Proto $scheme;
var scheme = request.Scheme;
// Retrieve the host that was forwarded by NGINX...
// This is set by the following directive in NGINX:
//  proxy_set_header Host $host;
var hostname = request.Host.Host;
// Retrieve the port that was forwarded by NGINX...
// This is set by the following directive in NGINX:
//  proxy_set_header X-Forwarded-Port $server_port;
var port = request.Host.Port;

// In case the port was not set, we will accept defaults, based on scheme...
if (request.Host.Port.HasValue)
    port = request.Host.Port.Value;
else
```

```
{
    if (scheme == "http")
        port = 80;
    else if (scheme == "https")
        port = 443;
}

bool nondefaultport = false;
if(scheme == "http" && port != 80)
    nondefaultport = true;
if(scheme == "https" && port != 443)
    nondefaultport = true;

// Compose the base url string...
string baseUri = scheme + "://" + hostname;
// If the port is not default, add it explicitly...
if(nondefaultport)
    baseUri = baseUri + ":" + port.ToString();

// Create the uri service from the above base url string...
var svc = new UriService(baseUri);
```

The above can be passed to a URIService instance, so it can help compose redirects and pagination.

```
// Create the uri service from the above base url string...
var svc = new UriService(baseUri);
```

Revision #12
Created 28 September 2025 22:57:45 by glwhite
Updated 29 September 2025 01:05:32 by glwhite