Simple Consul with Service Discovery

Step 1: Install Consul

1. Download from: https://www.consul.io/downloads
2. Extract to `C:\consul` and add to PATH
3. Run: `consul agent -dev -ui`

Step 2: Install NuGet Package

Run in Package Manager Console for ALL 3 projects:

```
Install-Package Consul
```

Step 3: Order API - Add ONE File

Create `ConsulRegistration.cs`:

csharp

```csharp
using Consul;

public class ConsulRegistration : IHostedService
{
    private readonly IConfiguration _config;
    private readonly IConsulClient _consul;
    private string _serviceId;

    public ConsulRegistration(IConfiguration config)
    {
        _config = config;
        _consul = new ConsulClient(c => c.Address = new Uri("http://localhost:8500"));
    }

    public async Task StartAsync(CancellationToken cancellationToken)
    {
        var uri = new Uri(_config["urls"] ?? "http://localhost:5002");
        _serviceId = $"order-{Guid.NewGuid()}";

        var registration = new AgentServiceRegistration
        {
            ID = _serviceId,
            Name = "order-api",
            Address = uri.Host,
            Port = uri.Port,
            Check = new AgentServiceCheck
            {
                HTTP = $"{uri}health",
                Interval = TimeSpan.FromSeconds(10)
            }
        };

        await _consul.Agent.ServiceRegister(registration);
    }

    public async Task StopAsync(CancellationToken cancellationToken)
    {
        await _consul.Agent.ServiceDeregister(_serviceId);
    }
}
```

Update `Program.cs` (add 2 lines):

```csharp
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Add these 2 lines
builder.Services.AddHealthChecks();
builder.Services.AddHostedService<ConsulRegistration>();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthorization();

// Add this line
app.MapHealthChecks("/health");

app.MapControllers();
app.Run();
```

Step 4: Product API - Add ONE File

Create `ConsulRegistration.cs`:

csharp

```csharp
using Consul;

public class ConsulRegistration : IHostedService
{
    private readonly IConfiguration _config;
    private readonly IConsulClient _consul;
    private string _serviceId;

    public ConsulRegistration(IConfiguration config)
    {
        _config = config;
        _consul = new ConsulClient(c => c.Address = new Uri("http://localhost:8500"));
    }

    public async Task StartAsync(CancellationToken cancellationToken)
    {
        var uri = new Uri(_config["urls"] ?? "http://localhost:5001");
        _serviceId = $"product-{Guid.NewGuid()}";

        var registration = new AgentServiceRegistration
        {
            ID = _serviceId,
            Name = "product-api",
            Address = uri.Host,
            Port = uri.Port,
            Check = new AgentServiceCheck
            {
                HTTP = $"{uri}health",
                Interval = TimeSpan.FromSeconds(10)
            }
        };

        await _consul.Agent.ServiceRegister(registration);
    }

    public async Task StopAsync(CancellationToken cancellationToken)
    {
        await _consul.Agent.ServiceDeregister(_serviceId);
    }
}
```

Update `Program.cs` (add same 2 lines as Order API)

Step 5: Gateway API - Add ONE File

Create `ConsulConfig.cs`:

csharp

```
using Consul;
using Yarp.ReverseProxy.Configuration;
using Microsoft.Extensions.Primitives;

public class ConsulConfig : BackgroundService, IProxyConfigProvider
{
    private readonly IConsulClient _consul = new ConsulClient(c => c.Address = new Uri("http://
    private volatile Config _config = new Config(new List<RouteConfig>(), new List<ClusterConfi

    public IProxyConfig GetConfig() => _config;

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            var services = await _consul.Agent.Services();

            var routes = new List<RouteConfig>
            {
                new RouteConfig
                {
                    RouteId = "product-route",
                    ClusterId = "product-cluster",
                    AuthorizationPolicy = "AdminPolicy",
                    Match = new RouteMatch { Path = "/api/product/{**catch-all}" }
                },
                new RouteConfig
                {
                    RouteId = "order-route",
                    ClusterId = "order-cluster",
                    AuthorizationPolicy = "UserPolicy",
                    Match = new RouteMatch { Path = "/api/order/{**catch-all}" }
                }
            };

            var clusters = new List<ClusterConfig>();

            // Find product services
            var productServices = services.Response.Values.Where(s => s.Service == "product-api
            if (productServices.Any())
            {
                clusters.Add(new ClusterConfig
                {
```

```
                ClusterId = "product-cluster",
                Destinations = productServices.ToDictionary(
                    s => s.ID,
                    s => new DestinationConfig { Address = $"http://{s.Address}:{s.Port}/"
                )
            });
        }

        // Find order services
        var orderServices = services.Response.Values.Where(s => s.Service == "order-api");
        if (orderServices.Any())
        {
            clusters.Add(new ClusterConfig
            {
                ClusterId = "order-cluster",
                Destinations = orderServices.ToDictionary(
                    s => s.ID,
                    s => new DestinationConfig { Address = $"http://{s.Address}:{s.Port}/"
                )
            });
        }

        var oldConfig = _config;
        _config = new Config(routes, clusters);
        oldConfig.SignalChange();

        await Task.Delay(5000, stoppingToken);
    }
}

private class Config : IProxyConfig
{
    private readonly CancellationTokenSource _cts = new CancellationTokenSource();

    public Config(IReadOnlyList<RouteConfig> routes, IReadOnlyList<ClusterConfig> clusters)
    {
        Routes = routes;
        Clusters = clusters;
        ChangeToken = new CancellationChangeToken(_cts.Token);
    }

    public IReadOnlyList<RouteConfig> Routes { get; }
    public IReadOnlyList<ClusterConfig> Clusters { get; }
    public IChangeToken ChangeToken { get; }
```

```
        public void SignalChange() => _cts.Cancel();
    }
}
```

Update Gateway `Program.cs` (add 3 lines after builder):

csharp

```
var builder = WebApplication.CreateBuilder(args);

// Add these 3 lines
builder.Services.AddSingleton<ConsulConfig>();
builder.Services.AddSingleton<IProxyConfigProvider>(p => p.GetRequiredService<ConsulConfig>());
builder.Services.AddHostedService(p => p.GetRequiredService<ConsulConfig>());

// Change this line (remove LoadFromConfig)
builder.Services.AddReverseProxy();

// Rest stays the same...
```

REMOVE the static routes from Gateway's `appsettings.json`:

json

```
{
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*",
    "Authentication": {
        "Key": "QGuiAdpxokd4Uoqz6d1rMHNuvEDoQR7YG4YYVLH/YqE=",
        "Issuer": "http://localhost:5000",
        "Audience": "http://localhost:5000"
    }
}
```

That's It! Test Dynamic Discovery

1. Start Consul: `consul agent -dev -ui`
2. Start all services normally
3. Test changing ports:
   - Stop Order API
   - Change port in `launchSettings.json` to 5003
   - Start Order API again
   - Gateway will automatically route to new port!

Check Consul UI: http://localhost:8500

The gateway now discovers services dynamically from Consul. No static configuration!